UNISYS

A Series
Disk Subsystem
Administration and Operations Guide

Release 3.9.0

September 1991

U S America
8600 0668–000

Priced Item

UNISYS

A Series
Disk Subsystem
Administration and Operations Guide

Release 3.9.0 September 1991

U S America
Priced Item 8600 0668–000

The names, places, and/or events used in this publication are not intended to correspond to any individual, group, or association existing, living, or otherwise. Any similarity or likeness of the names, places, and/or events with the names of any individual, living or otherwise, or that of any group or association is purely coincidental and unintentional.

NO WARRANTIES OF ANY NATURE ARE EXTENDED BY THE DOCUMENT. Any product and related material disclosed herein are only furnished pursuant and subject to the terms and conditions of a duly executed Program Product License or Agreement to purchase or lease equipment. The only warranties made by Unisys, if any, with respect to the products described in this document are set forth in such License or Agreement. Unisys cannot accept any financial or other responsibility that may be the result of your use of the information in this document or software material, including direct, indirect, special or consequential damages.

You should be very careful to ensure that the use of this information and/or software material complies with the laws, rules, and regulations of the jurisdictions with respect to which it is used.

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

Correspondence regarding this publication may be forwarded using the Product Information card at the back of the manual, or may be addressed directly to Unisys, Product Information, 25725 Jeronimo Road, Mission Viejo, CA 92691.

Page Status

Page	Issue
iii	–000
iv	Blank
v through xvii	–000
xviii	Blank
xix	–000
xx	Blank
1–1 through 1–18	–000
2–1 through 2–11	–000
2–12	Blank
3–1 through 3–12	–000
4–1 through 4–25	–000
4–26	Blank
5–1 through 5–17	–000
5–18	Blank
6–1 through 6–3	–000
6–4	Blank
7–1 through 7–13	–000
7–14	Blank
8–1 through 8–10	–000
9–1 through 9–9	–000
9–10	Blank
10–1 through 10–9	–000
10–10	Blank
11–1 through 11–5	–000
11–6	Blank
12–1 through 12–13	–000
12–14	Blank
A–1 through A–5	–000
A–6	Blank
Glossary–1 through 17	–000
Glossary–18	Blank
Bibliography–1	–000
Bibliography–2	Blank
Index–1 through 10	–000

Unisys uses an 11-digit document numbering system. The suffix of the document number (1234 5678–*xyz*) indicates the document level. The first digit of the suffix (*x*) designates a revision level; the second digit (*y*) designates an update level. For example, the first release of a document has a suffix of –000. A suffix of –130 designates the third update to revision 1. The third digit (*z*) is used to indicate an errata for a particular level and is not reflected in the page status summary.

About This Guide

Purpose

This guide is intended to provide a reference to the disk subsystem that runs on A Series systems. Included in the discussions are

- Explanations of the basic concepts, structure, and organization of the disk subsystem on Unisys A Series systems
- Descriptions of how to plan, install, and use the disk subsystem
- Descriptions of various related systems and subsystems, including the archive, catalog, and disk resource control (DRC) systems.

Scope

This guide emphasizes the software related to the disk subsystem on A Series systems. Discussion of hardware is limited to general concepts that are helpful in understanding the operation of the software, and cases where differences among Unisys A Series systems affect the way that the software manages the disk subsystem.

The disk subsystem software is basically the same for all A Series systems.

Audience

This guide is intended for use by operations center managers, senior operators, and system programmers to aid them in setting up the disk subsystem and operating it efficiently.

The guide is designed to be used by both new and experienced users of Unisys A Series systems.

Prerequisites

Users of this guide should have experience in operating Unisys A Series systems and should be familiar with system commands and Work Flow Language (WFL) statements.

How to Use This Guide

The subject matter in this guide is organized by topic. If you are not familiar with disk subsystem concepts, you should read each section in the order it is presented. If you are familiar with disk subsystem concepts, you can refer to the discussions in the order that you need them.

The sections titled "Disk Subsystem Concepts" and "Disk Initialization and Operation" explain the concepts, structure, and operation of the disk subsystem. If you are not familiar with Unisys A Series systems, these sections introduce you to the subsystem. If you are an experienced user of A Series systems, these sections can help familiarize you with the terminology used in the guide.

The sections titled "Planning and Installation," "Safety Mechanisms," "Disk Resource Control System," and "Recovery" provide an overview of how to plan, install, and use the disk subsystem.

The sections titled "Volume Library and Volume Directory," "Archiving Disk Files," "Cataloging," and "Comparing the Archive and Catalog Subsystems" discuss subsystems that enable you to backup and track files more efficiently.

The sections titled "Mirrored Disk Feature" and "Memory Disk Feature" describe these features and provide guidelines for their implementation and use.

Any of the sections in this guide can be read independently of the others. Most procedures and operations are explained in general terms. The guide often does not explain the formal syntax of system commands and WFL statements. If you need complete syntax information, refer to the *A Series System Commands Operations Reference Manual* and the *A Series Work Flow Language (WFL) Programming Reference Manual*. The online help text for the Menu-Assisted Resource Control (MARC) interface also provides information about some subjects covered in this guide.

Organization

The individual sections and the appendix are described in the following outline. In addition, a glossary, a bibliography, and an index appear at the end of this guide.

Section 1. Disk Subsystem Concepts

This section explains the components of the disk subsystem, the structure and functions of disks and disk packs, disk families, disk files, and disk file access. This section is intended for readers who are not familiar with disk concepts or Unisys A Series systems.

Section 2. Disk Initialization and Operation

This section explains how disks are prepared for use and how the disk subsystem is operated. This section expands upon the concepts presented in the previous section and includes the following topics: preparing a disk for use, identifying the different types of disks used on Unisys A Series systems, creating multidisk families, identifying online

and offline disks, releasing a disk from system use, saving disk units, and using system commands.

Section 3. Volume Library and Volume Directory

This section describes optional, independent subsystems that enable you to track the status of tape and disk volumes. Topics include tape security and how you can use it, maintenance of volumed tape and disk media, and the LISTVOLUME utility.

Section 4. Archiving Disk Files

This section describes the archive subsystem and its commands. Included in the discussion are explanations of the AUTORESTORE feature, the WFL *ARCHIVE* statements, and the archive file selection processes. Modification of the selector procedure in the standard archive support library is also discussed.

Section 5. Cataloging

This section explains the system function cataloging, which keeps track of backup copies of files in the system. Topics covered include how cataloging works, how cataloging affects system performance, how to set up and operate a cataloging system, how to rebuild a catalog, how to make and use backup copies of the catalog file, and how to replace a damaged disk on a cataloging system.

Section 6. Comparing the Archive and Catalog Subsystems

This section summarizes the differences and similarities that exist between the archive and catalog subsystems.

Section 7. Planning and Installation

This section provides system file requirements. It explains how to allocate files for efficient system performance and ease of recovery from system problems. It also describes techniques for the startup of the system, and discusses disk file headers. This section can be used when you first install a system or when you want to improve the performance of a system that is already operating.

Section 8. Safety Mechanisms

This section explains how to duplicate vital system files such as directories. It also describes how to make alternate halt/load families.

Section 9. Disk Resource Control System

This section explains what the DRC system does, and how to use it.

Section 10. Mirrored Disk Feature

This section explains the benefits, requirements, and options of mirrored disks. It also explains how to initiate the mirrored disk option, how to move mirrored sets within or between systems, and how to deallocate mirrored disks.

Section 11. Memory Disk Feature

This section explains how to create, initiate, and recover a memory disk and provides operational restrictions and considerations. It also explains some system commands.

Section 12. Recovery

This section explains how to correct problems caused by a damaged or destroyed disk, a faulty disk drive, disk I/O errors, directory errors, or corrupted or missing archive directory records.

Appendix A. Layout of Archive Directory Records

This appendix provides examples of archive directory records and describes the characteristics they share.

Related Product Information

***A Series CANDE Operations Reference Manual* (form 8600 1500)**

This manual describes how CANDE operates to allow generalized file preparation and updating in an interactive, terminal-oriented environment. This manual is written for a wide range of computer users who work with text and program files.

***A Series DMSII Utilities Operations Guide* (form 8600 0759)**

This guide describes how to maintain relationships between data elements in a DMSII database. This guide is written for database administrators and programmers who are responsible for database integrity and recovery.

A Series File Attributes Programming Reference Manual* (form 8600 0064). Formerly the *A Series I/O Subsystem Programming Reference Manual

This manual contains information about each file attribute and each direct I/O buffer attribute. The manual is written for programmers and operations personnel who need to understand the functionality of a given attribute. The *A Series I/O Subsystem Programming Guide* is a companion manual.

***A Series GETSTATUS/SETSTATUS Programming Reference Manual* (form 8600 0346)**

This manual explains how to use the various GETSTATUS and SETSTATUS calls used in the DCALGOL programming language. This manual is written for experienced ALGOL programmers who are involved with data communications.

A Series I/O Subsystem Programming Guide* (form 8600 0056). Formerly the *A Series I/O Subsystem Programming Reference Manual

This guide contains information about how to program for various types of peripheral files and how to program for interprocess communication, using port files. This guide is written for programmers who need to understand how to describe the characteristics of a file in a program. The *A Series File Attributes Programming Reference Manual* is a companion manual.

***A Series Security Administration Guide* (form 8600 0973)**

This guide describes systems-level security features and suggests how to use them. It provides administrators with the information necessary to set and implement effective security policy. This guide is written for system administrators, security administrators, and those responsible for establishing and implementing security policy.

***A Series System Commands Operations Reference Manual* (form 8600 0395)**

This manual gives a complete description of the system commands used to control system resources and work flow. This manual is written for systems operators and administrators.

***A Series System Configuration Guide* (form 8600 0445)**

This guide describes how to organize a complex computer system into different hardware configurations. It also describes the steps required to dynamically change the system from one configuration to another. This guide is written for experienced system administrators and system operators.

***A Series System Messages Support Reference Manual* (form 8600 0429)**

This manual presents operating system error messages and explains the most likely cause of each error and the most effective response to each message. This manual is written for operators and programmers responsible for the operation of A Series systems, and for the resolution of error conditions on those systems.

***A Series System Operations Guide* (form 8600 0387)**

This guide describes the basic concepts and procedures required to operate Micro A through A 6 systems and, more generally, all A Series systems. This guide is written for A Series operators, especially those with little or no experience.

***A Series System Software Support Reference Manual* (form 8600 0478)**

This manual describes a number of facilities used for system monitoring and debugging, including BARS, DUMPANALYZER, LOGANALYZER, and LOGGER. It also describes the format of the SUMLOG file. This manual is written for system support personnel and operators.

***A Series System Software Utilities Operations Reference Manual* (form 8600 0460)**

This manual provides information on the system utilities, such as DCSTATUS, FILECOPY, and DUMPALL. This manual is written for applications programmers and operators.

A Series Task Attributes Programming Reference Manual* (form 8600 0502). Formerly the *A Series Work Flow Administration and Programming Guide

This manual describes all the task attributes available on A Series systems. It also gives examples of statements for reading and assigning task attributes in various programming languages. The *A Series Task Management Programming Guide* is a companion manual.

A Series Task Management Programming Guide* (form 8600 0494). Formerly the *A Series Work Flow Administration and Programming Guide

This guide explains how to initiate, monitor, and control processes on an A Series system. It describes process structures and process family relationships, introduces the uses of many task attributes, and gives an overview of interprocess communication techniques. The *A Series Task Attributes Programming Reference Manual* is a companion manual.

***A Series Work Flow Language (WFL) Programming Reference Manual* (form 8600 1047)**

This manual presents the complete syntax and semantics of WFL. WFL is used to construct jobs that compile or run programs written in other languages and that perform library maintenance such as copying files. This manual is written for individuals who have some experience with programming in a block-structured language such as ALGOL and who know how to create and edit files using CANDE or the Editor.

Contents

Section 3. Volume Library and Volume Directory

Section 4. Archiving Disk Files

Section 5. Cataloging

Section 6. Comparing the Archive and Catalog Subsystems

Section 7. Planning and Installation

Section 11. Memory Disk Feature

Section 12. Recovery

Figures

Tables

Section 1
Disk Subsystem Concepts

This section provides an overview of disk subsystem concepts and terminology that are referred to throughout this guide. The section is intended for readers who are not familiar with disk concepts or Unisys A Series systems. This section explains the following topics:

- Disk subsystem components
- Disks and disk packs
- The structure of data on disks
- Families and multidisk families
- Disk files
- Disk file access

Disk Subsystem Components

The disk subsystem on Unisys A Series systems consists of the following components:

- The disk or disk pack media
- The disk drives, which are also called units or peripheral devices
- The disk drive controller and exchange, which controls the disk drive units and transfers information between the host system and the disk drive units
- The I/O controller, which provides the interface between the host system and the disk drive controller
- The software that controls the structure and operation of the disk subsystem and provides the structure for the information stored on the subsystem

Disks and Disk Packs

A disk is a data storage device that consists of one or more circular metal plates that are often called platters. These platters rotate at high speed around a spindle that is mounted in a disk drive.

One or both faces of each platter are coated with a thin film of magnetic material, and data is recorded on the disk as magnetic changes in this material. Data on the platter is accessed by a disk drive component called the read/write head.

Data is stored on the platters in concentric circles that are called tracks. These tracks are very narrow and there can be hundreds of them per inch. All the tracks on a disk that have the same radius form a cylinder.

Disks can be divided into two main types that have different physical characteristics. The two types of disks are disk packs and head-per-track disks. Various models of disks are available that have different capacities and performance capabilities, but Unisys A Series system software treats the models as logically identical.

Disk packs have multiple platters that are mounted on a central spindle. Disk packs have one or more movable read/write heads for each recording surface. These read/write heads move from track to track to access data. The movement of the read/write heads to the designated track is called a *seek*. Disk packs can be removable or nonremovable depending on the disk pack model. Nonremovable disk packs operate with smaller tolerances, so that tracks can be closer together and the disk pack can store more data.

Head-per-track disks have fixed read/write heads. Each track on each recording surface has one read/write head so that the read/write head does not need to be moved to the desired track. Physical head-per-track disks are no longer supported on Unisys A Series systems. However, memory disk is treated as though it were a head-per-track disk. For details, refer to Section 11, "Memory Disk Feature," later in this guide.

The Structure of Data on Disks

Disk tracks are physically divided into portions known as sectors. A sector is 30 words (180 bytes) long and is the smallest portion that can be read from or written onto a disk. Sectors are also called segments. Segments, as they refer to disk storage, are different from application program segments. This guide uses the term *sector* when referring to disk segments rather than *segment* to avoid confusion. Each sector on a disk has a unique address that the system uses to identify the location of the sector.

When you create a disk file or add to it, the data is not necessarily stored in one contiguous sequence of sectors. Instead, the data is stored in portions called areas that each contain an equal number of sectors. An area is the contiguous group of sectors allocated for that portion of the file. An *area* is also referred to as a *row*. The file attributes you designate refer to the term *area*, while the Master Control Program (MCP) refers to the term *row* in its processing. An area is measured in terms of logical records, where a logical record is the amount of data that is accessed by the execution of one read or write statement in a program. A row is measured in terms of sectors. All the areas on a disk do not need to be the same size, but all the areas of a single file must be the same size. Figure 1-1 illustrates the structure of disk tracks and sectors.

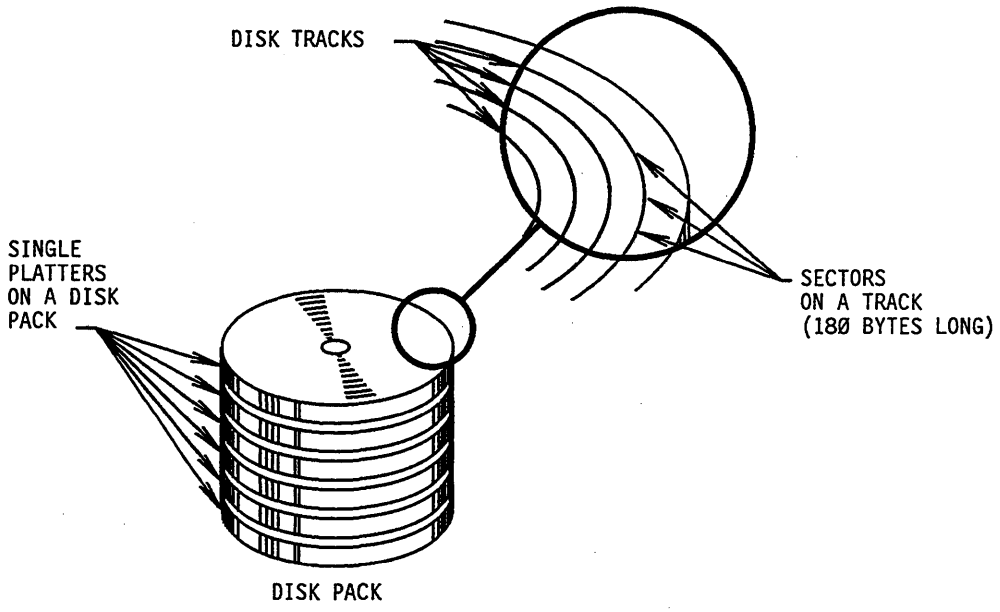

Figure 1–1. The Physical Structure of a Disk

The system accesses each area of a file by obtaining the physical address of the area from a special structure called the disk file header. Each disk file has a header, which also includes information such as the name of the file, the length of its records, its creation date, and so on. Figure 1-2 illustrates the structure of the disk file header.

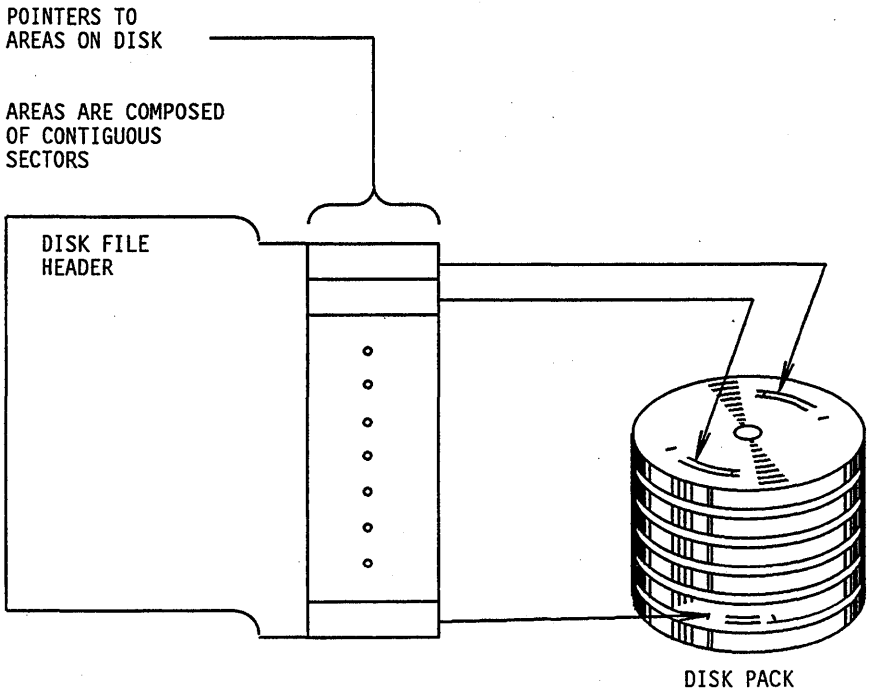

Figure 1–2. A Disk File Header

The disk file headers of all the permanent files on a disk are stored in a special file called the flat directory. When the system needs to access a file, it locates the disk file header in the flat directory for the appropriate disk and then uses the physical addresses in the header to access the individual areas of the file. The flat directory is also referred to as the system directory. Figure 1-3 illustrates the structure of the flat directory.

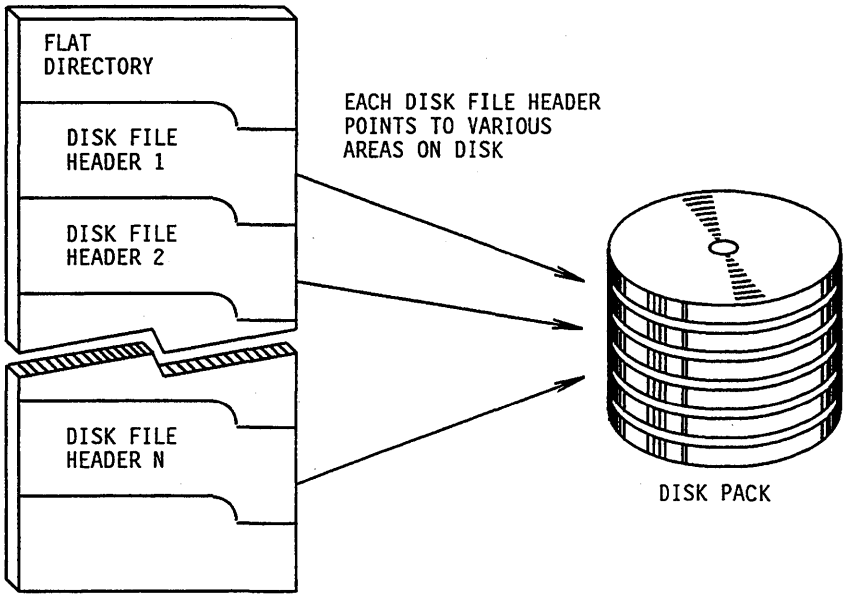

Figure 1–3. The Flat Directory

Families and Multidisk Families

A family consists of one or more disks that are logically grouped together and treated as a single entity by the system. Each family has a family name, which can consist of up to 17 alphanumeric characters (letters and digits). When you access a file, you designate both the file name and the family name as *<file name> ON <family name>*. The file name and the family name, when designated together, are called the file title.

The family name of a disk and other information about the disk is stored in a structure called the label. The label is stored on the first 28 sectors of the disk. The label also contains the serial number that your site assigns to the disk, and a pointer to the flat directory of the disk, if there is a flat directory. When a program needs to access or allocate a file, the system uses the family name to locate the disk that contains the file. The system uses the serial number to distinguish disks from each other when they are online. For more information about the label, family name, and serial number, refer to Section 2, "Disk Initialization and Operation."

The flat directory is stored on only one member of the family, although other disks in the family can have duplicate copies of the flat directory. The family member that contains the flat directory that the system is currently using to access the family is called the base pack.

Sometimes one disk is not large enough to store all the data you want to put on it. The system allows you to have several disks in a family so that files can extend across several the members of the family. A family that consists of more than one disk is known as a

multidisk family. Although a multidisk family can be mixture of different disk pack models, it cannot be a mixture of memory disks and disk packs.

The first disk that you enter into a family is the base pack; it contains the flat directory for the family. You can add more disks to the family as continuation packs. The system logically links the family members together so that they are treated as a single entity. A family can have up to 254 continuation packs. Continuation packs do not necessarily contain a flat directory file. Refer to Section 2, "Disk Initialization and Operation," for information about how to create a multidisk family.

The system usually allocates different areas of a particular file on different members of the same family. You have the option of designating which family members are to receive areas of the file by using the FAMILYINDEX file attribute. Refer to the *A Series I/O Subsystem Programming Reference Manual* for more information about the FAMILYINDEX attribute. The system cannot spread a file over disks that are in different families. Figure 1-4 illustrates the structure of a multidisk family.

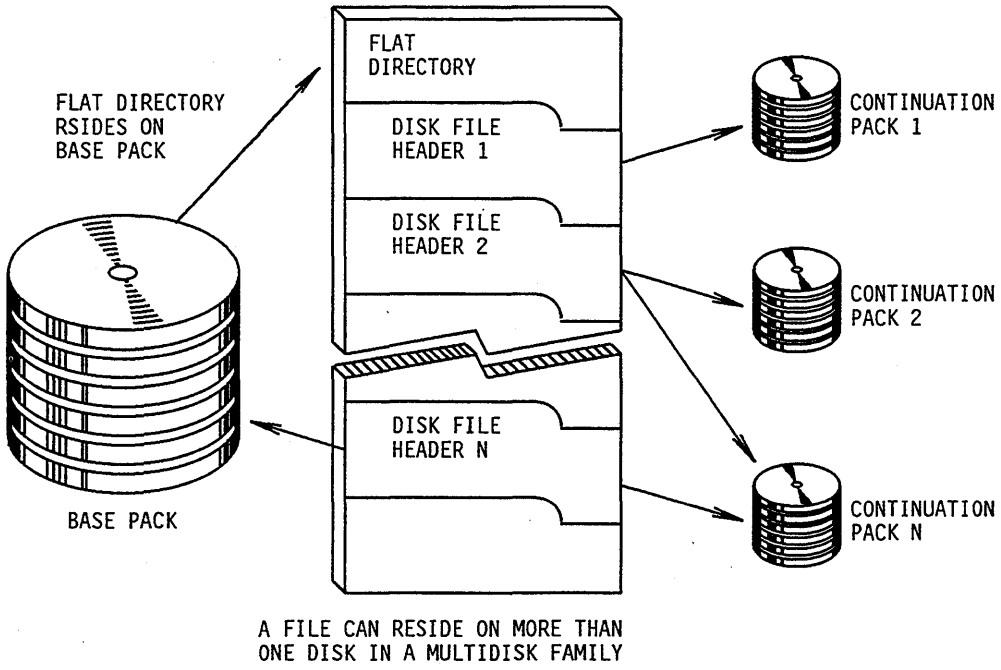

Figure 1–4. A Multidisk Family

It is important to remember that disks with the same family name are not necessarily in the same family. If you try to use a disk that has the same family name as another disk on the system and the two disks are not in the same family, the system can issue an error or RSVP message.

When the system needs to access a multidisk family, the base pack and any continuation pack that contains a copy of the flat directory must be online. If the file to be accessed is spread over several family members, continuation packs must be brought online when areas are needed that are stored on those continuation packs.

The system assigns a family index number to each member of a family when that member is first added to the family. The family index number is incremented by one for each new member. The initial base pack is assigned family index 1, the first continuation pack is assigned family index 2, and so on. Programs can use the FAMILYINDEX file attribute to distribute files or areas of files to particular disks in the family. There is no connection between the serial number of a disk and the family index number of the disk.

The file name of the flat directory is SYSTEMDIRECTORY/<family index number>, where <family index number> is the three-digit family index number of the disk on which the flat directory is stored.

You can use the PER PK version of the PER (Peripheral Status) system command to examine the family name, serial number, and family index number of all the online disks on the system.

Disk Files

A disk file is a named collection of data that is stored on disk. All disk files have disk file headers that describe the file. The header contains the physical addresses where the file is stored on the family and various file attributes. The header for a permanent file is stored in the flat directory of the family on which the file is stored; the header for a temporary file is stored in main memory.

File Attributes

File attributes are specifications included with each file that define basic information about that file. Some of these attributes identify the file, while others describe the structure of the file. File attributes that identify the file include FILENAME, CYCLE, and VERSION. File attributes that describe the structure of the file include AREAS, AREALENGTH, BLOCKSIZE, and MAXRECSIZE. Many of the file attributes for a disk file are stored in the header of the file.

When someone at your installation wants to create a new disk file, that person must decide what values to assign to file attributes. File attributes can be assigned either within a program or with file equations at compilation or execution time.

Creating a New Disk File with Fixed-Length Records

Use the following procedure to assign attributes to files with fixed-length records:

1. Assign to KIND the value DISK.
2. Assign to FAMILYNAME the name of the family on which you want the file to be stored. The default value of FAMILYNAME is DISK. If you use family substitution, the family name can be changed to the family designated by the family specification. Refer to "Family Substitution" in Section 7, "Planning and Installation," for more information on this subject.
3. Assign to FILENAME the name you choose for the file.
4. Assign to NEWFILE the value TRUE.

5. Assign to FILEKIND the mnemonic value that describes the internal structure and purpose of the file. The default value of FILEKIND is DATA.
6. Assign to FRAMESIZE a value of 4, 8, or 48 to indicate how many bits are to be transferred as one unit of data during an I/O operation. If you use a value of 4, data is transmitted in 4-bit units, which are equal to hexadecimal characters. If you use a value of 8, data is transmitted in 1-byte units, which are equal to EBCDIC or ASCII characters. If you use the value 48, data is transmitted as full words (48 bits). Other file attributes, such as AREALENGTH, BLOCKSIZE, and MAXRECSIZE are expressed in the units assigned in FRAMESIZE.
7. A logical record is the amount of data accessed by one execution of a read or write statement in a program. Determine how many words or characters each logical record is to contain and assign that value to MAXRECSIZE. The default value of MAXRECSIZE is 30 words, or 180 EBCDIC characters.
8. If your program accesses records sequentially, it can reduce I/O operation time by reading or writing several physically adjacent records in one block. The number of logical records accessed as one block is determined by the attribute BLOCKSIZE. The default value of BLOCKSIZE is the value of MAXRECSIZE, which means that only one logical record is accessed by each I/O operation. When determining the block size, keep in mind that if you use large blocks, the I/O operations are efficient, but you are tying up a large amount of main memory. On the other hand, if you use very small blocks, you must perform more I/O operations.
9. Determine the size, in FRAMESIZE units, that you want to allocate for each area of the file and assign that value to AREALENGTH. The default value of AREALENGTH is MAXRECSIZE multiplied by 1000 and then rounded up so that the value can be evenly divided by the value of BLOCKSIZE. Areas that are too small or too large have disadvantages. If the areas are too small, they limit the number of records in the file. If the areas are too large, it is more difficult for the system to find the contiguous disk sectors needed to store each area.

Using the AREAS Attribute

The AREAS attribute designates how many areas can be allocated for the file; the default value of AREAS is 20. When the file is created, the number of areas designated by AREAS is not automatically allocated. Instead, the system leaves an empty entry in the disk file header for each possible area. The system allocates an area to the file the first time a program attempts to read or write records that must be placed in that area. When an area is allocated, the system places a pointer in the header to that area.

If all the areas for a file have been allocated and you need to expand the file, the system increases the value assigned to AREAS and automatically allocates new areas if the attribute FLEXIBLE is assigned its default value of TRUE. A file can contain up to 1000 areas.

Opening the File

After you have finished the above procedure to assign the various file attributes, you can open the file. When you open the file, the system creates a header for the file and stores the values of the various file attributes in the header. The system also enters values

automatically for other file attributes, such as CREATIONDATE and CREATIONTIME. Then you can write records to the new file. When you have finished processing the file, you can close it so that it becomes a permanent file. The system then updates the values of certain file attributes, such as LASTRECORD, and places the header in the flat directory of the disk. Refer to "Temporary and Permanent Files" in this section for more information on this subject.

Accessing a Permanent File

To access an existing permanent file, you do not need to reassign all the above attributes. If you assign the value TRUE to the DEPENDENTSPECS file attribute before you access the file, you must designate only the FILENAME, FAMILYNAME, and KIND; the file retains certain previous attribute values.

Many other file attributes can be selected, and these attributes are also stored in the disk file header. Refer to the *A Series I/O Subsystem Programming Guide* for more information about file attributes.

File Generations on Noncataloging Systems

In certain application programs, it is convenient to create different generations of the same file, such as a new generation for each time the program runs. This section discusses generations for noncataloging systems only. Refer to "File Generations" in Section 5, "Cataloging," for more information about this subject.

There are two ways to differentiate the generations of a file on a noncataloging system.

Distinguishing Generations with Unique File Names

The first way to differentiate the generations of a file on a noncataloging system is to assign a different file name to each generation. An example of this approach would be PAYABLE/001, PAYABLE/002, PAYABLE/003, and so on. All the generations can be online at once because they have different file names.

Distinguishing Generations with the CYCLE and VERSION Attributes

The second way to differentiate the generations of a file on a noncataloging system is to use the CYCLE and VERSION attributes to distinguish each generation. CYCLE and VERSION are integer values; the higher the value of CYCLE, and the higher the value of VERSION within that CYCLE, the better the genealogy of the generation is said to be, relative to other generations. Only one generation can be online at one time, because the file names are the same. If you do not assign values to CYCLE and VERSION, the default value is 1 for CYCLE and 0 for VERSION. You must assign CYCLE in order to use VERSION, but assigning VERSION is optional when you use CYCLE.

The following example illustrates how the system determines the genealogy of each generation when CYCLE and VERSION are used on a noncataloging system. In this example, the first entry is the generation with the best genealogy and the last entry is the generation with the worst genealogy.

Cycle	Version
4	0
3	3
3	2
2	3
2	2
2	1
2	0
1	5
1	0

CYCLE and VERSION function in the following manner when you try to access an existing file on noncataloging systems:

- If you designate CYCLE and VERSION when you want to access the file, the system locates the generation that has that exact CYCLE and VERSION. If a file with the proper file name is not online or does not exist, the system displays a "NO FILE" message on the ODT.

  If a file with the proper file name, but the wrong CYCLE and VERSION, is online, the system displays an "UNMATCHED GENEALOGY" message on the ODT.

- If you specify CYCLE, but not VERSION, when you want to access the file, the system locates the generation with that exact CYCLE and the VERSION equal to 0. If the generation that is online does not have that CYCLE and VERSION, the system displays an "UNMATCHED GENEALOGY" message on the ODT.

  If no generation of the file is online, the system displays a "NO FILE" message on the ODT.

- If you do not specify CYCLE and VERSION when you want to access the file, the system locates the generation that is online. If no generation of the file is online, the system displays a "NO FILE" message on the ODT.

Resident and Nonresident Files

Files are often referred to as *resident* or *nonresident*. A file is resident if it is the primary copy of the file (as opposed to a backup copy) and it is stored on disk, regardless of whether or not the disk is online. A file is nonresident if there is no copy of the file on the disk family, but there are one or more backup copies of the file on library maintenance tapes, other disk families, or both. Backup copies of files are tracked by the archive subsystem and the catalog subsystem. The terms resident and nonresident in this guide do not pertain to the file attribute RESIDENT.

Temporary and Permanent Files

Disk files can be either temporary or permanent. The header of a temporary file header is not stored in the flat directory of the family on which the file is located. When a program closes a temporary file, the contents of the file are no longer available, and the disk space of the file is returned to the system. A file is permanent when it has an entry in the flat directory of the family on which the file is stored. When a program closes a permanent file, the contents of the file remain stored on disk.

A temporary file is created when the NEWFILE file attribute is assigned the value TRUE and a program opens the file. The program can then write data in the file and access it if necessary. If steps are not taken to make the file permanent, the disk space of the file is returned to the system when that file is closed.

You can make a temporary file into a permanent file in one of the the following ways, depending on your program:

- In an ALGOL program, use the LOCK or CRUNCH options in the CLOSE statement that closes the file. Both options make the file permanent; CRUNCH also causes the unused portion of the last area of the file to be returned to the system.
- In a COBOL or COBOL74 program, use the LOCK, CRUNCH, or SAVE options in the CLOSE statement that closes the file. All three options make the file permanent. LOCK also marks the file so that it cannot be reopened during that execution of the program. CRUNCH also causes the unused portion of the last area of the file to be returned to the system.

When a new file is opened, it is made permanent immediately if you specify one of the following file attributes:

- Assign to the PROTECTION file attribute either the value SAVE or the value PROTECTED. When the file is opened, it becomes a permanent file.
- Assign the to SENSITIVEDATA file attribute the value TRUE. When the file is opened, it becomes a permanent file.

After a file is permanent, it remains permanent unless you remove it. You can remove a permanent file when you perform one of the following actions:

- Use the WFL statement REMOVE to remove the disk file header from the flat directory. If the file is not in use, its disk space is returned immediately to the system.
- In a COBOL, COBOL74, or ALGOL program, use the PURGE option in the CLOSE statement that closes the file.
- Create another permanent file with the same name in the same family. If the OP+AUTORM version of the OP (Options) system command has been designated, the system automatically removes the old file with the duplicated name from the family.

If you remove a permanent file while it is still being used by other programs, the file becomes a temporary file. The file can still be used by the other programs that had

already opened it. The disk space is not returned to the system until the file is closed by the last program using it.

When you make a file permanent on a cataloging system, the system places an entry for that file in the catalog. If you remove a permanent cataloged file and no backup copies of the file exist, the system deletes the catalog entry for the file. If you remove a permanent cataloged file and backup copies of the file exist, the system changes the catalog entry to indicate that there are no resident generations of the file.

Disk File Access

Computer performance depends to a large degree upon the ability to efficiently perform I/O operations. Unisys has developed a very efficient and reliable method for accessing disk files so that the system can operate at its full potential. This access method consists of two parts:

- The disk access structure used by the entire system
- The flat directory on each disk family

Refer to "The Structure of Data on Disks" in this section for more information about the flat directory.

The system has a special disk file called the disk access structure file or the catalog file. This disk file is stored on one family in the system. Use the system command DL CATALOG to designate the family on which the catalog file is to be stored. The catalog disk file contains several components, which include the pack access structure table (PAST) and the file access structure table (FAST). On tape security subsystems, the catalog disk file contains the volume directory. On cataloging systems, the catalog disk file contains the catalog and the volume library.

Whenever a disk family comes online, the system uses the disk file family name to look up the PAST entry for that family. The PAST entry is used to locate the FAST entries for the family whenever the system needs to locate a permanent disk file. The system looks up the file name in the FAST entries to determine the location of the disk file header for the requested file in the flat directory.

The access structure functions differently on cataloging and noncataloging systems. The access structure for a cataloging system contains entries for all available versions of all resident and backup files of all online and volumed disk families. The access structure for a noncataloging system contains only entries for the files that are currently resident on an online disk family on the system. The file name of the access structure on noncataloging systems is SYSTEM/ACCESS/<family index number>. The file name of the access structure on cataloging systems is SYSTEM/CATALOG/<family index number>. The <family index number> indicates which member of the catalog family contains the access structure.

The MCP constructs the access structure the first time the system is initialized so that the access structure contains entries for each family that is online at that time. Each time a family is brought online, entries for the files of that family are entered into the access structure. When a family is removed with the system commands CLOSE (Close

Pack) or POWER (Power Up/Down), references to its files are removed from the access structure unless it is a cataloging system and the family has an entry in the volume library as the result of the WFL statement VOLUME. Refer to Section 5, "Cataloging," for more information about this subject.

Pack Access Structure Table (PAST)

The PAST contains pointers into the FAST. These pointers indicate where in the FAST the entry for each family is located. The PAST is not accessed each time the MCP needs to locate a disk file. Instead, the MCP reads the PAST entries into a table in main memory when the system is initialized. The MCP then can access the FAST directly.

Each time a family is brought online, the MCP checks the PAST to see if there is an entry for the family. If there is an entry but it is not up-to-date, or if the family is being brought online for the first time and thus does not have an entry, the MCP creates PAST and FAST entries for the family and stores a copy of the the PAST entry in main memory.

File Access Structure Table (FAST)

When the MCP needs to access a disk file, it uses the FAST to locate the disk file header in the flat directory of the appropriate family. The FAST is a sorted table that can be logically considered as a hierarchical tree structure. The FAST contains a pointer to the disk file header or catalog record of each disk file on the system. This allows the MCP to read the FAST entry for the family and then use the pointer to quickly locate the disk file header in the unsorted flat directory. Figure 1-5 illustrates how the FAST corresponds to the flat directory of one family.

Using the FAST and flat directories to access files is efficient, and it provides a safety mechanism in case either the FAST or the flat directory experiences a problem.

The FAST entries for a family are logically organized as a tree structure, but the structure is unusable if it experiences data corruption or some other problem. However, the system can circumvent the problem by reconstructing the FAST entries for each family by reading the flat directory of that family. Refer to "Family Rebuilds" in this section for more details about this subject.

The flat directory is an unsorted structure that is accessed randomly through the FAST. Thus, if there is a problem with a record in the flat directory, the problem is restricted to one record and, at most, one file header. The rest of the flat directory, which contains records for other headers, can be successfully accessed.

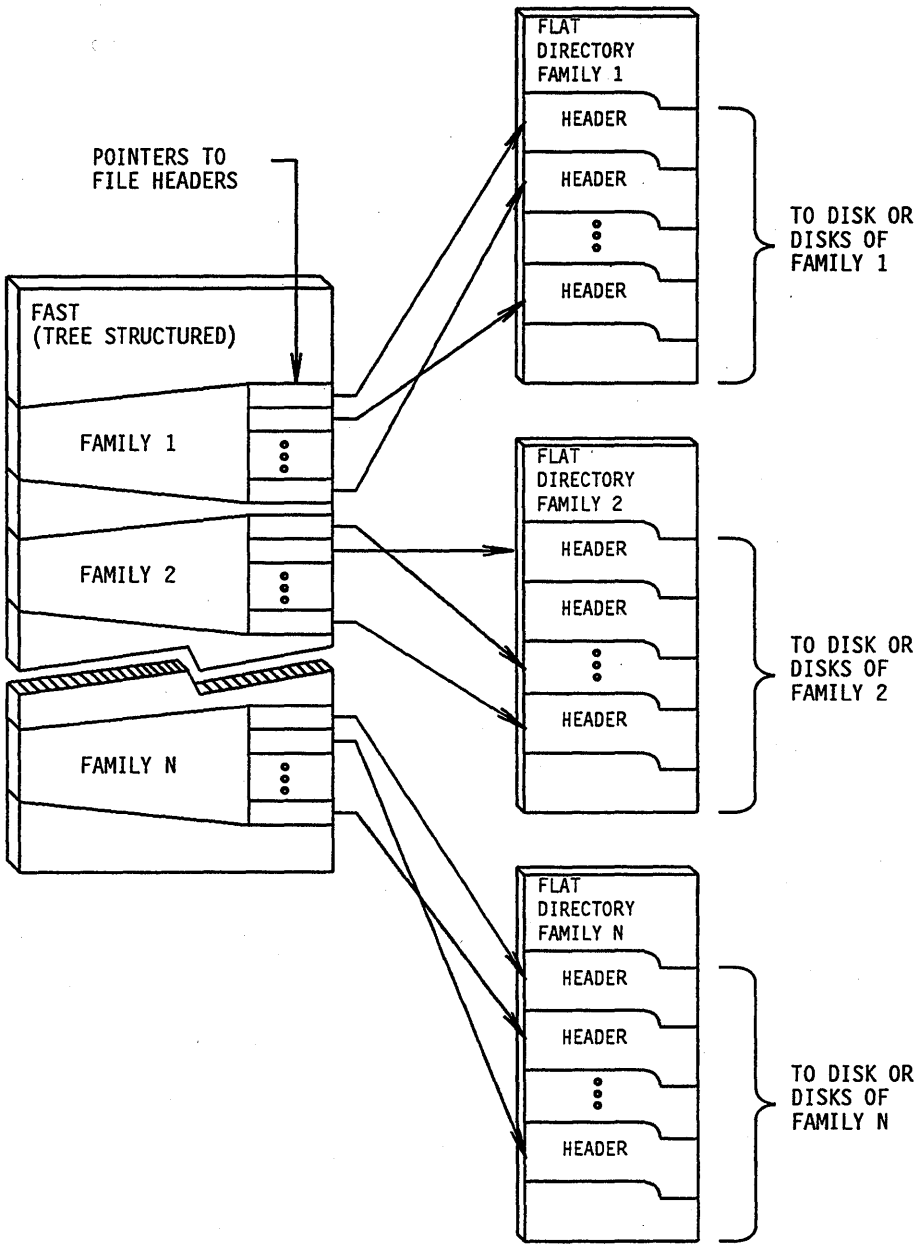

Figure 1–5. The File Access Structure Table (FAST)

Process of Accessing Disk Files

When you access a disk file by designating its file title (the file name and the family name), the following procedure occurs:

1. The MCP uses the family name to locate the proper family entry in the FAST.
2. The MCP reads the file entries for the family in the FAST and locates the pointer to the disk file header for the file with that file name.
3. The MCP reads the disk file header from the flat directory.
4. The MCP obtains from the header the physical disk addresses of the area or areas that contain the disk file.
5. The MCP accesses the data of the file at those physical addresses.

Figure 1-6 illustrates how the components of the access structure and the flat directory allow efficient access of disk files.

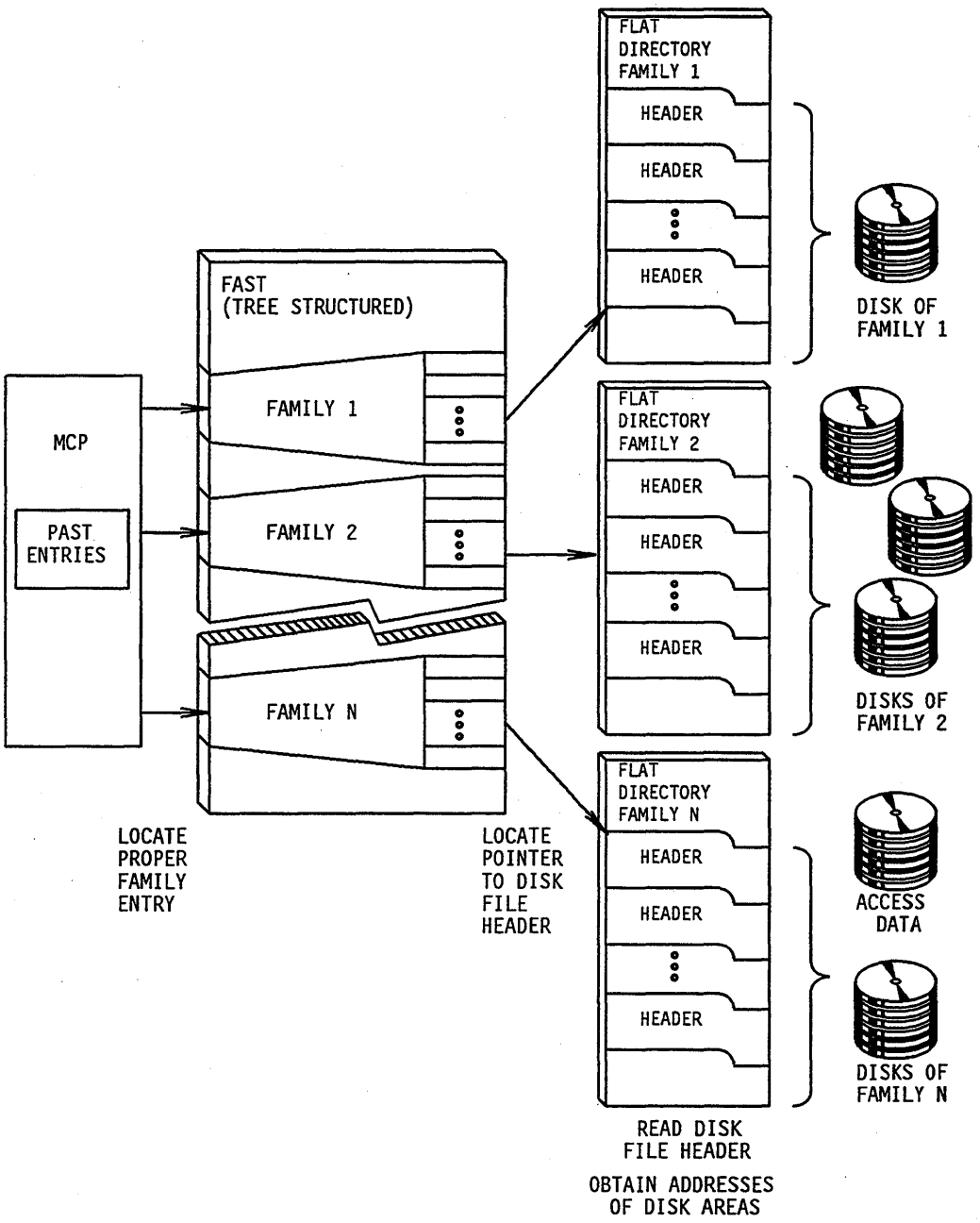

Figure 1–6. The Process of Accessing Disk Files

Archive Directories and the Archive Access Structure Table

For each disk family at your installation, the system creates and maintains an archive directory on the DL CATALOG family. These archive directories keep track of backup

copies of disk files made with the WFL *ARCHIVE FULL*, *ARCHIVE INCREMENTAL*, *ARCHIVE DIFFERENTIAL*, *ARCHIVE ROLLOUT*, and *ARCHIVE MERGE* statements. The system stores a record in an archive directory for each file backed up by an *ARCHIVE* WFL statement. The system retrieves records from the archive directory by the names of the files that are backed up. The system uses a special set of index records stored in each archive directory to rapidly locate archive records for the file. The special index structure in an archive directory is called an archive access structure table (AAST).

Family Rebuilds and Archive Rebuilds

The MCP automatically handles file access structures and archive access structures. The only time your installation is aware of these access structures is during the processes known as family rebuilding and archive rebuilding. A family rebuild is the process the system uses to construct or reconstruct the FAST entries for a family by sequentially reading its flat directory. Similarly, an archive rebuild is the process the system uses to construct or reconstruct the AAST entries for a family by sequentially reading its archive directory.

Family Rebuilds

Family rebuilds occur in the following situations:

- Automatically when you bring a base pack online on a noncataloging system and no local access structure table (LAST) is found on the base pack
- Automatically to recover from certain directory errors when they occur
- When you use the RB (Rebuild Access) system command to initiate a rebuild

If during a family rebuild, two disk file headers are found with the same file name, the rebuild process bypasses the second file. If you do not take action, it is not possible to access the second copy of the file. However, if you use the system REMOVE or CHANGE command to remove or rename the first copy of the file, the next family rebuild reveals the second copy of the file.

Archive Rebuilds

Archive rebuilds occur in the following situations:

- Automatically to recover from certain directory errors or from certain directory actions interrupted by halt/loads.
- When you use the RB (Rebuild Access) system command to initiate a rebuild.

Local Access Structure Table (LAST)

To eliminate the need for a family rebuild on a noncataloging system each time a pack is brought online, the system uses the LAST to restore the FAST, instead of rebuilding the FAST from scratch. The LAST is written to the base pack whenever the base pack is

closed (with the system command CLOSE), freed (with the system command FREE), or powered off (with the system command PO). When an operator readies the pack, the system attempts to restore the FAST from the LAST. If the restoration is successful, a family rebuild is not carried out.

The LAST, however, is *not* created in the following conditions:

- The system is a cataloging system.
- The pack is not write-enabled when closed with the CLOSE command.
- The pack has no room for the LAST when the pack is closed with the CLOSE command.
- The pack is manually turned off without first entering the system commands CLOSE, FREE, or PO.
- I/O errors are encountered during the creation of the LAST.
- The pack is mirrored.

Available Disk Space

When a disk is ready at halt/load time or when a family is brought online, the MCP needs to know what space is available on each disk. The MCP keeps track of unused disk space by means of tables stored in main memory that list available disk space. The process of constructing or reconstructing tables of available disk space is called directory complementing.

The MCP starts with the assumption that the entire disk (with the exception of the label area) is available and then reads the disk file headers in the flat directory. Each time the MCP reads a header, it removes the space allocated to that file from the tables of available space. After the last header is read and processed, the remaining space on the disk is listed in the tables of available disk space as being available for use. The tables of available space are different from the master available table (MAT), which contains pointers to the disk sectors that are not defective so that no defective sectors are used.

Section 2
Disk Initialization and Operation

This section describes the procedures for setting up and using disks and disk families. This section covers the following topics:

- Preparing a disk for use
- Identifying types of disks used on A Series systems
- Creating multidisk families
- Identifying online and offline disks
- Releasing a disk from system use
- Saving disk units
- Using the RES, SQUASH, and XD commands

Preparing a Disk for Use

Before a disk can be used for the first time, the following two steps must be performed to prepare it:

1. Perform the initialize, verify, and relocate (IVR) operation.
2. Issue the RC (Reconfigure Disk) system command.

Performing the IVR Operation

Before Unisys ships a disk to a customer, it uses the IVR operation to write sector boundaries and the label on the disk. The IVR operation also creates the master available table (MAT), which contains pointers to the disk sectors that are usable. If a sector is defective, the IVR operation deletes the pointer to the bad sector for the MAT so that it cannot be used. The MAT is different from the available disk table, which keeps track of unallocated or unassigned disk space.

Although the IVR operation is available on some systems, it should never be used except under the direction of your Unisys field engineer.

Using the RC Command to Create a Disk Family

Before your installation uses a disk, you must use the RC command to prepare the disk for system use. In the RC command, you designate the family name and serial number you want to assign to the disk. The family name and serial number are stored on the disk label. You can use the OL (Display Labels and Paths) system command to examine disk label information.

To create a base back, reconfigure the disk that you want to be the base pack of the family. The base pack contains the flat directory and must always be online when the family is accessed. Use the following RC command syntax:

```
RC PK <unit number> NAME <family name>
```

The variable <unit number> is the unit number of the disk drive on which the base pack is mounted, and the variable <family name> is the name that you want to call the family. The family name must be different from the name of any family that is online to the system.

The family name can be up to 17 alphanumeric characters (letters and digits). When a program needs to access or allocate a file, the system uses the family name to locate the correct disk.

When you reconfigure a disk with the RC command or purge it with the PG (Purge) system command, any files on that disk are made permanently inaccessible. Care should be taken so that a disk is not reconfigured by accident. The OWNER clause of the RC command acts as a safety feature to ensure that you actually want to reconfigure the disk. When you first reconfigure the disk, you can assign to the OWNER clause any name of up to 14 alphanumeric characters. Then, if you ever try to reconfigure that disk again, the system displays a message on the ODT that specifies the value of OWNER and asks the operator if the reconfiguration is permissible. Refer to the *A Series System Commands Operations Reference Manual* for the proper syntax of the RC command.

Identifying the Types of Disks Used on A Series Systems

Disks to be used on A Series systems are logically organized in one of two ways:

- Native-mode disks
- Interchange disk packs

Descriptions for these disk types are provided in the following table:

Disk Type	Description
Native-mode disk	This disk is used on all A Series systems. A native-mode disk cannot be transferred from an A Series system to other Unisys systems such as a B 1000, B 2900, B 3900, or B 4900 system. When this guide uses the term *disk*, it is referring to native-mode disks unless otherwise indicated.
Interchange disk pack	This disk pack has a directory format that enables files to be transferred from a Unisys A Series system to other Unisys systems such as a B 1000, B 2900, B 3900, or B 4900 system. Interchange disk packs have many limitations and are not discussed in this guide.

Creating Multidisk Families

A multidisk family consists of several disks that have the same family name and are logically linked together so that they are treated by the system as a single entity. The disks in the family, other than the base pack, are called continuation packs. Multidisk families allow much more data to be stored on one family than is possible with a single disk family. Different models of disk packs can be members of the same family, but a family cannot be a mixture of disk packs and memory disks. For an overview of disk family concepts, refer to "Families and Multidisk Families" in Section 1, "Disk Subsystem Concepts."

Assigning a disk the same family name as another disk that is already online to the system does not automatically make the disks members of the same disk family. To create a multidisk family, you must use a special form of the RC system command to logically connect the disks together. Adding a continuation pack to an existing family with the special form of the RC command does not harm the files on the existing family, but it makes all the old files inaccessible on the disk that is reconfigured.

Whenever you want to add a continuation pack to a family, use the RC command with the following syntax:

```
RC PK <unit number> BP <serial number> NAME <family name>
```

The variable <unit number> is the unit number of the disk drive on which the new continuation pack is mounted, the variable <serial number> is the serial number of the base pack, and the variable <family name> is the family name of the base pack. The base pack must be online when you reconfigure the continuation pack.

Each disk also has a serial number, and when you reconfigure a disk with an RC (Reconfigure Disk) system command, you can optionally change the serial number. The serial number is a 6-digit number that you choose. The system uses the serial number to distinguish disks from each other. Installations can use serial numbers to give each disk a unique, permanent identification to aid in keeping track of disk use, as well as logging errors or other problems. Some installations use the serial number that is listed on the bottom of the disk by the manufacturer. Others choose a reference number such as the date that the disk was first used. Some installations give nonremovable disks the same serial number as the unit number of the disk drive on which the disk is mounted. The serial number of each online disk on a system must be unique, and Unisys suggests that serial numbers be assigned in a systematic fashion.

Disk serial numbers are used for three separate purposes by the system:

- The serial number is used to identify individual disk volumes. In this respect, the serial numbers appear in the PER PK display and system log entries. Note especially that two separate disks with the same serial number cannot be online at the same time, with the exception of disks in a a mirrored disk set.
- The serial number assigned to a disk volume is used by the system to link it into its proper disk family. For a multiple disk family, each member in the family must have a unique serial number. For example, suppose that there is a two-volume disk family, where the first volume has serial number 111 and the second volume has serial number 222. Suppose that the second volume is temporarily out of order. After it is fixed, the second volume cannot be reconfigured back into the family with the serial number 222. Otherwise, the following error message would be issued:

  ```
  PKnn SERIAL NO. ALREADY IN FAMILY AS FAMILYINDEX 2
  ```

  To place the disk into the family again, the operator would have to either designate a new serial number in the RC command or invoke the FAMILYINDEX clause in the RC command.
- At cataloging installations, the serial number of the first disk volume in the family is used to identify if a disk family is volumed so that it can contain cataloged files. Refer to "Handling Volume Libraries and Volume Directories" in Section 5, "Cataloging."

Identifying Online and Offline Disks

A disk is online to the system if all the following conditions are true:

- It is mounted on a unit that has been logically connected to the system with the ACQUIRE (Acquire Resource) system command.
- It is mounted on a unit that has not been reserved with the UR (Unit Reserved) system command.
- It has not been logically detached from the system with the CLOSE (Close Pack) or FREE (Free Resource) system command.
- Its label and its flat directory (if it has one) have been read successfully. Refer to "Families and Multidisk Families" in Section 1, " Disk Subsystem Concepts," for more information about the flat directory.

A disk is offline to a particular system if any of the above criteria have not been met. An online continuation pack cannot be accessed unless there is an online base pack for the family.

Releasing a Disk from System Use

After the system has successfully brought a disk online, the disk can be used by the system and user programs. While a program or system function is using the disk, that online disk cannot arbitrarily be taken offline or removed from the system.

To place the disk offline, you must enter a specific system command such as CLOSE (Close Pack), UR (Unit Reserved), or SV (Save).

After the system successfully brings a disk online, user programs and system processes can access the disk and use the files on the disk. An in-use, online disk does not become offline even if it experiences multiple I/O errors or becomes *not ready* or *hung*. To stop or prevent usage of an online disk, use the system command CLOSE or force the disk offline in some other way.

To place a disk offline, you can issue one of the following commands:

- FREE (Free Resource)
- CLOSE (Close Pack)
- PO (Power Off)
- Certain forms of the RES (Reserve) command

You can use the SV (Save) system command to prevent more files from being opened on the disk, but the SV command does not place the disk offline. Refer to "Saving Disk Units" in this section.

The following commands also take a disk offline, alter its status, and return it to the online state if it has not been saved with the SV command:

- RC (Reconfigure Disk)
- LB (Relabel Pack)
- PG (Purge)

If the system or a user program is using a disk, the system rejects all the above commands except SV and RES. Using the FREE, CLOSE, PO, RC, LB or PURGE commands would return the following message:

```
PK<unit number> UNIT IN USE
```

For instructions on using the RES command, refer to "Using the RES, XD, and SQUASH Commands" in this section.

The system considers a disk to be in-use for any of the following reasons:

- The disk contains one or more areas of files that are open and are in-use by the system or a user program. The in-use files can be either permanent files that are listed in the directory or new and temporary files that do not appear in the directory. You can use the system command PD (Print Directory) to list permanent files.
- The disk is being accessed by an MCP function such as a WFL *REMOVE* or *CHANGE* statement or a system RES or SQUASH command.
- The disk is the current base pack of the family or contains an active duplicate directory (refer to the system command DD), and the family has one or more files that are open or a disk that is in-use.

- The disk is a member of either an active halt/load family or the active DL JOBS, DL LOG, or DL OVERLAY families.
- The disk is being processed by an MCP status procedure such as RY (Ready), CLOSE (Close Pack), RC (Reconfigure Disk), LB (Relabel Pack or HCU), or PG (Purge).
- On certain systems the disk contains an active bootcode file.

To remove a disk from use (such as to power it off or to reconfigure it), you must wait for all active users of the disk to release the disk.

You can try several methods to get users off the disk:

- Move files from one member of a family to other members of the family by using the system command RES PK<unit number>.
- Remove a duplicated directory by using the system command DD–.
- Discontinue the MCP RY procedure by using the system command CL PK<unit number>.

However, it is not always possible to locate and terminate all users of a disk. For instance, the RES command cannot move files that are in-use, DD– is rejected if the system is using the disk as the basepack, and members of the halt/load and DL JOBS family are always in-use.

If it is absolutely necessary to remove a disk from use by the system, issue the system command SV PK<unit number> and halt/load the system. This technique works for all disks except for the halt/load disk and disks that contain active bootcode files.

Another method for taking a pack offline is to create a mirror copy of the pack and then to release the original pack. This method is the least disruptive to the operational environment. For further information, refer to Section 10 in this guide.

Saving Disk Units

The system command SV PK<unit number> does not test the in-use status of the disk and does not place the disk completely offline. That is, system and program usage of an online disk that is subsequently saved with the SV command is not stopped. However, attempts to open files and use the disk after it has been saved can be stopped.

You can perform the following commands on a disk that is saved with the SV command:

- RC (Reconfigure Disk)
- LB (Relabel Pack or HCU)
- PG (Purge)
- CLOSE (Close Pack)

Sometimes you must reconfigure or change the status of a disk before the system can bring the disk online. For example, if the disk is part of the halt/load family and the

disk is turned on, an RC system command cannot be executed for the disk. As soon as the disk becomes ready, the system reads its label and recognizes that it is part of the halt/load family; disks in the halt/load family are always considered to be in-use.

Use the following procedure to reconfigure or change the status of a disk that is part of the halt/load family:

1. Issue the system command SV PK<unit number> for the unit.
2. Turn on the disk drive or halt/load the system.
3. Issue the RC (or LB or PG) command for the disk as needed.
4. Issue the system command RY PK<unit number> for the unit.

Using the RES, XD, and SQUASH Commands

You can use the system commands RES PK <unit number>, SQUASH PK <family name>, and XD PK <unit number> to change and control the physical location (sector addresses) of disk file areas. The RES PK <unit number> command moves the areas of disk files away from a designated section. The SQUASH PK <family name> command moves the areas of disk files together so that the unallocated space on the disk is consolidated. The XD PK <unit number> command marks the designated section of the disk as not to be used. For example, you can use the XD command if you suspect that a section of a disk is defective and prone to parity errors.

Only one RES, XD, or SQUASH procedure can be active on the system at any particular time.

Moving Allocated Disk File Areas with the RES and XD Commands

You can use the RES (Reserve Disk) command to move disk files from a particular section of a disk volume or from the entire disk volume to other parts of the disk or other disks on the family. After the RES command has moved all the allocated files from the designated section, it marks the section of disk or disk volume with the status specified implicitly or explicitly in the RES command.

The RES procedure cannot ordinarily move files that are actively being used by programs or the system. However, the RES procedure often can move certain in-use files, such as code files and directories. During its processing, the RES procedure reports each in-use file that it cannot move with one of the following messages:

```
PK<unit number> WAIT ON PERMANENT FILE <file name>

PK<unit number> WAIT ON TEMP FILE <file name>

PK<unit number> WAIT ON TEMP FILE <file name>, CREATED BY JOB<mix no>

PK<unit number> WAITING FOR JOB/SESSION <mix number>
```

Then the RES procedure waits either for some operator action such as DS or QT or for the file to be closed by the programs that are using it.

Designate the section of the disk to be cleared by the RES command in one of the following three ways:

- As a specific disk sector address range
- As an entire family member or disk volume
- On memory disks, as the section of disk controlled by specific lock-out switches

Information follows that explains how to perform each option.

When the entire disk volume is to be cleared, a special situation exists. Areas allocated with the FAMILYINDEX attribute or the SINGLEPACK attribute on that family member cannot be moved to other members of the family. The RES procedure issues messages that list all files in the FAMILYINDEX and SINGLEPACK set that cannot be moved:

```
PK<unit number> FAMILYINDEX FILE BLOCKING RESERVE <file name>
PK<unit number> SINGLEPACK FILE BLOCKING RESERVE <file name>
```

The files listed in the above message format would have to be removed from the disk before the RES procedure could complete.

After the RES procedure moves all file areas out of the designated disk section, the system establishes the disposition of the disk or section that is reserved in one of the following three ways, based on the action that you take:

- If you use an ADDRESS, SECTOR, or SEGMENT specification in the RES command, the system covers the section of reserved disk with a BADDISK file. The reserve procedure performs the XD command on the reserved section of the disk.

  The system does not use sectors of disk covered by a BADDISK file to allocate disk files. Using RES with an ADDRESS, SECTOR, or SEGMENT specification is a good choice if you suspect that a section of disk is defective and is prone to I/O parity errors.

  Using RES with an ADDRESS, SECTOR, or SEGMENT specification differs from the XD (Bad Disk) command in one respect. RES automatically moves files from the designated section of disk, but XD only succeeds if no files are allocated in the designated section to begin with.

  An eliminated section can be returned for use by removing the BADDISK/<file name> file that covers it. For example, the command RES PK97 SEGMENT 3020 FOR 10 moves any file areas out of the sector range 3020-3029 and then creates a BADDISK file that covers that area with the following file name:

  ```
  BADDISK/FMLYINX2/UNIT97/ADB68H.
  ```

  You can use the RES command to delete a continuation pack from a family by designating the address range that covers the entire disk. The following command is an example:

  ```
  RES PK<unit number> SEGMENT 28 FOR "<size of disk> - 28"
  ```

The first 28 segments contain the disk label and cannot be reserved. After you delete the disk from the family, you should purge the disk by using the PG command. You can then reconfigure the disk into another family. If you skip the purge step after the RC command is executed, the disk will again contain the BADDISK file that covers the entire volume that was created by the original RES command.

- If you want to reserve the entire disk volume (no ADDRESS, SECTOR, or SEGMENT specifications exist) and you designate AS MAINT, the reserve procedure marks the disk unit as saved.

  The system moves all disk file areas except BADDISK areas from the disk volume to other volumes in the family and marks the unit as saved. You can use this particular form of the RES command for one of several reasons:

  - Use RES PK<unit number> AS MAINT to terminate usage of the disk so that you can close it (with the CLOSE command) and unload it so that you can load another disk volume onto the unit.
  - Use AS MAINT prior to running maintenance tests on the unit. That is, reserve the unit with the system command UR (Unit Reserved), and then the peripheral test driver (PTD) can use the disk.

  You can make the unit and disk volume available for use again by issuing the system command RY PK<unit number>. For example, the command RES PK97 AS MAINT moves all file areas on pack 97 to other disks in the same family (a base pack of a family can never be reserved as MAINT) and marks the unit as saved.

Consolidating Disk Space with the SQUASH Command

Often, files on a disk are continually being added, expanded, or removed. As the files on a disk are changed, it is not always possible to fit areas of new files between areas of other files; there might be only small areas available between in-use areas. If a large number of files (especially many small files) are on the disk, a large number of unused sectors might be scattered throughout the disk. The total number of available sectors could be large, but the system might not be able to use those sectors, because contiguous groups of sectors are too small for an area of a file to be stored there. This situation is known as *checkerboarding*.

The SQUASH (Consolidate Disk Allocation) system command can be used to consolidate files and to increase the number of contiguous usable sectors. The SQUASH operation moves allocated areas of files from one place to another so as to reduce the amount of checkerboarding. That is, the SQUASH procedure tries to move or group allocated areas of files to one or two parts of the disk and thereby arrange them so that the unallocated sectors on the disk are contiguously grouped together. This operation facilitates allocating new areas and files on the disk because after a SQUASH operation, it is more likely that large groups of contiguous sectors (as required for files that specify large area sizes) will be available.

The SQUASH procedure is bypassed if one of the following conditions exists:

- There is not enough space on the disk (unallocated disk sectors) to make the SQUASH operations useful.
- The majority of available space on the disk is already grouped together.

Either of these conditions causes SQUASH to issue the following message:

```
PK<unit number> DISK SPACE ALREADY CONSOLIDATED, SQUASH BYPASSED
```

Otherwise, the SQUASH operation proceeds to analyze the location of allocated areas on the disk and attempts to move some of those allocated areas so as to group together the available sectors on the disk. However, the SQUASH procedure cannot successfully move all allocated areas. In particular, SQUASH does not try to move areas that belong to the following files:

- XD files, that is files with FILEKIND = BADDISK. These are not true files but are place holders used by the system to mark or prevent usage of sectors on the disk that are suspected of being faulty.
- Files with very large AREASIZEs (greater than the largest available area on the disk prior to the beginning of the SQUASH procedure). The size of the largest available area can be displayed with the system command DU (Disk Utilization). SQUASH cannot move areas of a file unless it can find another available area in which the allocated area would fit.
- Files that are open and in-use at the time of the SQUASH operation. These files would include both permanent disk files (those whose names appear in the directory of the disk), temporary disk files, and WFL job files (those whose names appear on the active DL JOBS family). Occasionally there are specific cases where SQUASH can sometimes move areas that belong to open, in-use files that include the following:
  - SYSTEMDIRECTORY files
  - SYSTEM/ACCESS or SYSTEM/CATALOG files
  - The JOBDESC file
  - The active MCP code file
  - Code files

The general restriction that SQUASH cannot move open, in-use disk files means that the SQUASH operation might not always be useful if it is employed at times when the disk contains many open and in-use files.

If, prior to moving data, the SQUASH procedure determines that there are so many files blocking the operation that the procedure cannot increase the extent of the largest group of available sectors on the disk, the SQUASH procedure bypasses the disk with the following message:

```
PK<unit number> DISTRIBUTION OF INUSE, XD, AND LARGE FILES MAKES
SQUASH NOT FEASIBLE
```

When the SQUASH operation is performed on a multidisk family, the system consolidates areas on the base pack first, and then consolidates areas on each continuation pack. Areas are not moved from one family member to another.

The SQUASH operation is a time-consuming process, and Unisys suggests that you use SQUASH at night or at some other time when system usage is minimal. You can use the FILEDATA utility to produce a listing that shows the checkerboarding for a disk; this

listing shows the in-use portions of the disk and the unused space that surrounds those portions. You can also use the DU (Disk Utilization) system command to examine the space that is available on a disk family. Refer to the information on FILEDATA in the *A Series System Software Utilities Operations Reference Manual* for more information about FILEDATA. Refer to the *A Series System Commands Operations Reference Manual* for more information about the SQUASH and DU commands.

Section 3
Volume Library and Volume Directory

The system supports two optional, independent subsystems that keep track of the status of tape volumes and disk volumes. These two subsystems store volume information in special records in the central catalog directory.

In noncataloging installations, these records are stored in the SYSTEM/ACCESS directory. In cataloging installations, they are stored in the SYSTEM/CATALOG directory.

One set of these records is called the volume directory. It exists only if the installation uses the system option SECOPT TAPECHECK=AUTOMATIC. The volume directory keeps track of the status of tape volumes only.

The other set of records is called the volume library. It exists only if the installation runs with the system option OP + CATALOGING turned on. The volume library keeps track of the status of both tape and disk volumes.

The volume directory and volume library keep track of only those tapes and disks that have been *volumed*—that is, tapes and disks that have been included in the volume directory or volume library. You include a tape or disk in the volume directory or library by executing a WFL *VOLUME ADD* statement for the tape or disk. In the command, you must specify the name of the volume, the attribute KIND (tape or disk), and the serial number of the volume as shown in the following example:

```
VOLUME ADD SCRATCH (TAPE, SERIALNO = "X55612")
```

After a tape or disk has been volumed, you cannot change its serial number with the system command SN (Serial Number) unless you first issue a WFL *VOLUME DELETE* statement to delete the entry for the tape from the volume library and directory. However, you can use the SN command to change the density of the volumed tape as long as you respecify the original serial number of the volume.

Using Tape Security

The volume directory is a data structure in the tape security subsystem. This directory stores information about tape volumes at your installation by tape serial number, and includes the following information for each volumed tape:

- Current volume name
- Tape volume ownership
- Tape file security attributes

The system uses the volume directory when it either readies or purges a tape volume and when it opens a tape file for input or output. The tape volume directory is a feature that is available only with InfoGuard security-enhancement software.

The tape volume directory has the following two sections:

- The data section contains data records that describe individual tape volume families.
- The key section contains a structure used by the system to rapidly locate individual data records in the data section.

You can use the TV (Tape Volume) system command to determine if a particular tape is volumed. You can also use the LISTVOLUME utility to generate a printer listing of all the volumes that are in the volume directory. For more information, refer to "Using the LISTVOLUME and LISTVOLUMELIB Utilities" later in this section.

The system frequently changes volume directory entries to track the changing status of the tape version. An operator can issue commands to rebuild the volume directory if it becomes corrupted. If you designate SECOPT TAPECHECK = NONE, the system does not create, reference, or update the volume directory data structure. To have volume directory capability, you must enter the following command:

```
SECOPT TAPECHECK=AUTOMATIC
```

When you enter this form of the command, it does not become effective until the next time you halt/load the system.

Volume Library

The volume library is a component in the SYSTEM/CATALOG file that keeps track of all volumed disks and tapes used by the system. A volumed disk or tape is one for which a descriptive entry has been added to the volume library with the WFL statement VOLUME ADD. Note that disks and tapes often are referred to as volumes, whether or not they are entered in the volume library. The volume library contains an entry for each volumed disk or tape family. A multidisk family has only one entry.

You can use the PV (Print Volume) system command to determine if a particular disk or tape is volumed. You can also use the LISTVOLUMELIB utility to generate a printer listing of all the volumes that are in the volume library. For more information, refer to "Using the LISTVOLUME and LISTVOLUMELIB Utilities" later in this section.

The system uses a special structure called the volume access structure table (VAST) to access volume library entries. The system handles the VAST automatically.

How the System Handles Volumed Tapes and Disks

The information that follows describes how the system handles volumed tapes and volumed disks.

Handling Volumed Tapes

The following subsections provide general information and an example procedure for handling volumed tapes.

General Information

For volumed tapes, the system automatically updates the volume library and the volume directory when you change the status of a volume. When you purge a volumed tape with the PURGE (Purge) system command or place a new file on the tape, the system automatically updates the volume library, the volume directory, or both. However, automatic updates such as these do not take place under the following conditions:

- The tape is purged while cataloging or the tape security subsystem is not in use.
- The tape is purged on another system.
- A new file is stored on the tape while cataloging or the tape security subsystem is not in use.
- A new file is stored on the tape while the tape is being used on another system.

Example Procedure for Handling Volumed Tapes

The following example procedure shows how a cataloging system or a tape security subsystem handles these tasks for volumed tapes:

- Assigns a serial number to a new tape.
- Enters the tape into the volume library, the volume directory, or both.
- Adds a file to the tape.
- Purges the tape.
- Assigns a new serial number to the tape.

In this example, the unit number of the tape drive on which the tape is mounted is 118, and the serial number of the tape is 555. The tape is a single reel family, so some system messages in this example contain #1 to indicate that this reel is the first (and, in this case, only) reel in the family.

1. Use the SN (Serial Number) system command to purge the tape and assign the serial number 555:

   ```
   SN MT 118 555
   ```

   The system then displays the following messages on the ODT:

   ```
   MT 118 WILL BE SN-ED
   MT 118 PURGED
   ```

2. Use the WFL statement VOLUME ADD to place an entry for the tape into the volume library and the volume directory as a scratch tape:

```
VOLUME ADD SCRATCH (KIND = TAPE, SERIALNO = 555)
```

   If you are using cataloging, the system then displays the following message on the ODT:

```
VOLUME FAMILY SCRATCH (MT) [555] #1 ENTERED
```

   If you are using the tape security subsystem, the system displays the following message on the ODT:

```
VOLUME DIRECTORY ADD SCRATCH [000555] OK
```

3. Use the WFL statement COPY to copy the file PAYABLE to the tape and change the name of the tape from SCRATCH to ACCOUNTS:

```
COPY PAYABLE TO ACCOUNTS (SERIALNO = 555)
```

   The cataloging system then displays the following messages on the ODT:

```
VOLUME FAMILY SCRATCH (MT) [555] #1 DELETED
VOLUME FAMILY ACCOUNTS (MT) [555] #1 ENTERED
```

   The tape security subsystem does not display a message when a tape is changed.

4. Use the PV (Print Volume) or TV (Type Volume) system command to display information from the volume library or volume directory respectively about tape 555:

   a. Enter the following syntax for the PV command:

```
PV MT 555
```

      The system then displays the following information from the volume library on the ODT:

```
-----VOLUME LIBRARY ENTRY FOR (MT) [000555]-----

SERIALNO 000555, #0001, PETAPE
FAMILY NAME: ACCOUNTS
FAMILY CREATED ON: 3/27/85
FAMILY EXPIRATION DATE: 4/27/85
FAMILY CREATION SITE: 281
ONE MEMBER IN THIS FAMILY
```

   b. Enter the following syntax for the TV command:

```
TV MT 555
```

The system then displays the following information from the volume directory on the ODT:

```
-----VOLUME DIRECTORY ENTRY FOR (MT) [ØØØ555]-----

FAMILY NAME: ACCOUNTS
FAMILY OWNER: *
FAMILY CREATED ON: 3/27/85
FAMILY EXPIRATION DATE: 4/27/85
FAMILY CREATION SITE: 281
FAMILY VOLUME TYPE: LIBRARY MAINTENANCE
UPDATE TIMESTAMP: TUESDAY JUNE Ø3, 1986 (86154) AT 1Ø:2Ø:3Ø
SECURITY: PRIVATE - USAGE: READ/WRITE
GUARDFILE: NONE
FAMILY STRUCTURE #1 ONLY
  (MT) [ØØØ555] #ØØØ1, TEMPORARY, NOT-RESTRICTED
```

5. Use the PG (Purge) system command to purge the tape (the PG command uses the unit number, 118, not the serial number, 555):

   ```
   PG MT 118
   ```

   The system purges the tape, automatically updates the volume library and/or volume directory entry to indicate that the tape is now a scratch tape, and displays the following cataloging messages on the ODT:

   ```
   VOLUME FAMILY ACCOUNTS (MT) [555] #1 DELETED
   VOLUME FAMILY SCRATCH (MT) [555] #1 ENTERED
   MT 118 PURGED
   ```

   The tape security subsystem does not display any messages when the volume directory entry for the tape is updated in this manner.

6. Use the SN (Serial Number) system command to purge the tape and attempt to assign the serial number 666:

   ```
   SN MT 118 666
   ```

   Instead of purging the tape, the cataloging system displays the following message on the ODT:

   ```
   MT 118 VOLUME DELETE REQUIRED
   ```

   If you are using the tape security subsystem, the following message is displayed:

   ```
   VOLUME DIRECTORY MT 118 SCRATCH [ØØØ555] NOT SNED,
   REQUIRES VOLUME DELETE
   ```

   The system does not allow the serial number to be changed until the existing volume library entry and/or volume directory entry for the tape has been deleted.

7. Use the WFL statement VOLUME DELETE to delete the existing volume library entry for the tape:

   ```
   VOLUME DELETE SCRATCH (KIND = TAPE, SERIALNO = 555)
   ```

The cataloging system then displays the following message on the ODT:

```
VOLUME FAMILY SCRATCH (MT) [555] #1 DELETED
```

If you are using the tape security subsystem, the following message is displayed:

```
VOLUME DIRECTORY DELETE SCRATCH [ØØØ555] OK
```

8. Enter the SN command again to purge the tape and assign the serial number 666:

```
SN MT 118 666
```

The system then displays the following messages on the ODT:

```
MT 118 WILL BE SN-ED
MT 118 PURGED
```

9. Use the VOLUME ADD statement to place an entry for the tape back into the volume library and volume directory:

```
VOLUME ADD SCRATCH (KIND = TAPE, SERIALNO = 666)
```

The cataloging system then displays the following message on the ODT:

```
VOLUME FAMILY SCRATCH (MT) [666] #1 ENTERED
```

If you are using the tape security subsystem, the following message is displayed:

```
VOLUME DIRECTORY ADD SCRATCH [ØØØ666] OK
```

Handling Volumed Disks (Volume Library Only)

The following subsections provide general information and an example procedure for handling volumed disks.

General Information

The system handles volumed disks differently from volumed tapes. When you want to rename a disk or disk family, use the RC (Reconfigure Disk) or LB (Relabel Pack) system command. When you want to purge the files from a disk or disk family, use the PG (Purge) system command. However, when you use these commands on a volumed disk, first use the WFL statement VOLUME DELETE to remove the entry for the disk family from the volume library. After you reconfigure or rename the disk, you can place a new entry for it into the volume library with the WFL statement VOLUME ADD.

To use the VOLUME ADD statement with a multipack disk family, designate the name of the family and the serial number of the pack with the family index number 1. Do not use the VOLUME ADD statement with the other packs in the family, even if the unit bearing family index number 1 is not currently online.

Special handling might be necessary for the family serial number of a volumed disk family with duplicated directories that is created with the system command DD (Directory Duplicate). The serial number of the first member of the family identifies whether the family is volumed. (The VOLUME ADD and VOLUME DELETE

commands issued for other disks in the family do not affect the status of the family; the system ignores those entries.)

Even if you remove the directory from family member 1 with a DD– command, you should still use the serial number of family member 1 when you issue a VOLUME ADD statement for the family. If you delete from a family the first disk in that family by issuing a DD– command and you later reconfigure that first disk as another family, the original serial number of that disk remains the serial number that is used to identify the family in the volume library. The serial number of the original family member 1 is displayed as the second serial number in the ODT displays for the system commands *PER PK* and *OL PK <unit number>*. You can use these commands to determine the serial number to use in VOLUME commands for the family.

You should observe a special precaution if you delete from the family the first disk in that volumed family and later reconfigure the disk as (or into) a new disk family. If the original family is to continue to be usable and volumed, you should not use the serial number of the old family again. This precaution is necessary so that the catalog information for the old family can remain linked to that family with the original family serial number. So, if you delete and reconfigure the first disk in a volumed family, you should give the disk a new serial number.

Example Procedure for Handling Volumed Disks

The procedure that follows illustrates how to reconfigure a disk, add it to the volume library, and change the name of the disk. In this example, the family name of the disk is SHIPPING, the serial number of the disk is 032785, and the unit number of the disk drive on which the disk is mounted is 211. The SHIPPING disk is the only member of the family, so some system messages in this example contain #1 to indicate that this disk is the first (and, in this case, only) disk in the family.

1. Use the RC (Reconfigure Disk) system command to reconfigure and name the disk:

   ```
   RC PK 211 NAME=SHIPPING SERIAL=Ø32785
   ```

2. Use the WFL statement VOLUME ADD to enter SHIPPING into the volume library:

   ```
   VOLUME ADD SHIPPING (SERIALNO=Ø32785, KIND = PACK)
   ```

   The system displays the following message on the ODT:

   ```
   VOLUME FAMILY SHIPPING (PK) [Ø32785] #1 ENTERED
   ```

3. Use the PV (Print Volume) system command to display information from the volume library about disk 032785:

   ```
   PV PK Ø32785
   ```

The system then displays the following information on the ODT:

```
-----VOLUME LIBRARY ENTRY FOR (PK) [Ø32785]-----

SERIALNO Ø32785, #ØØØ1, PACK, BASE
FAMILY NAME: SHIPPING
FAMILY CREATED ON: 3/27/85
FAMILY CREATION SITE: 281
ONE MEMBER IN THIS FAMILY
```

4. Use the system command CLOSE (Close Pack) to place the family offline:

   ```
   CLOSE PK211
   ```

   The system then displays the following message on the ODT:

   ```
   PK211 UNIT CLOSED
   ```

5. Use the WFL statement VOLUME DELETE to delete the existing volume library entry for the disk:

   ```
   VOLUME DELETE SHIPPING (KIND = PACK, SERIALNO = Ø32785)
   ```

   The system then displays the following message on the ODT:

   ```
   VOLUME FAMILY SHIPPING (PK) [Ø32785] #1 DELETED
   ```

6. Enter the following LB (Relabel Pack) command syntax to change the disk name to INVENTORY:

   ```
   LB PK211 NAME = INVENTORY OLDNAME = SHIPPING
   ```

   The system then displays the following message on the ODT:

   ```
   PK211 UNIT RELABELLED
   ```

7. Use the WFL statement VOLUME ADD to enter INVENTORY into the volume library:

   ```
   VOLUME ADD INVENTORY (KIND = PACK, SERIALNO = Ø32785)
   ```

   The system then displays the following message on the ODT:

   ```
   VOLUME FAMILY INVENTORY (PK) [Ø32785] #1 ENTERED
   ```

Using the LISTVOLUME and LISTVOLUMELIB Utilities

LISTVOLUME is a utility in InfoGuard that reports on the status of volumes in both the volume library and the volume directory. Only privileged users can use this utility.

The LISTVOLUMELIB utility reports on the status of volumes in the volume library only.

To obtain information about the volume library, the CATALOGING option must be designated (OP + CATALOGING). To obtain information about the volume

directory your installation must have set the SECOPT TAPECHECK option equal to AUTOMATIC.

The following subsections provide examples of the parameters that you can use to run these utility programs. Note, however, that they might require a large amount of disk space, depending on the size of the volume library or the volume directory.

Running the LISTVOLUME Utility

To run the LISTVOLUME utility, enter *RUN SYSTEM/LISTVOLUME* followed by one of the parameters shown in the examples that follow.

If you use one of the following examples, the utility reports on whichever structures exist on the current system (volume library or volume directory). If both exist, both are reported.

```
RUN SYSTEM/LISTVOLUME ("");
RUN SYSTEM/LISTVOLUME (" ");
RUN SYSTEM/LISTVOLUME ("PRINT");
```

To report only volume library entries, use the following WFL statement:

```
RUN SYSTEM/LISTVOLUME ("PRINT VOLUMELIB");
```

To report only volume directory entries, use the following WFL statement:

```
RUN SYSTEM/LISTVOLUME ("PRINT VOLUMEDIR");
```

To report entries for both the volume library and the volume directory, use the following WFL statements:

```
RUN SYSTEM/LISTVOLUME ("PRINT VOLUMELIB, VOLUMEDIR");
RUN SYSTEM/LISTVOLUME ("PRINT VOLUMEDIR, VOLUMELIB");
```

If you enter the previous example and your installation does not have both the volume library and the volume directory, the system displays an error message to let you know that only one structure exists.

The report generated for the volume library contains the following sections:

- The date that the LISTVOLUME utility was run and the disk unit number that contains the volume library structure.
- Scratch volumes, sorted by serial number within type.
- Expired volumes, sorted by serial number within type (DISK, TAPE, PACK, PETAPE). The LISTVOLUME utility computes the expiration date of each volume by using the SAVEFACTOR and CREATIONDATE attributes of each volume.
- Valid volumes that contain files, sorted by serial number within type. These volumes have not yet expired and are assumed to have valid file data.

- Expired volumes, sorted by title. The following information is printed for each volume:

```
Volume name= KIND = <disk, tape, pack, petape>

             CREATED: <creation date>
             EXPIRES: <expiration date>
             BASE or EMPTY
             CREATED ON SYSTEM: <system serial number>

             [volume serial number] reel number, ...
```

- Valid volumes, sorted by title. The reporting format is as follows:

```
Volume name= KIND = <disk, tape, pack, petape>

             CREATED: <creation date>
             EXPIRES: <expiration date>
             BASE or EMPTY
             CREATED ON SYSTEM: <system serial number>

             [volume serial number] reel number, ...
```

- Volume library statistics. This part of the report shows disk space consumed by the volume library and used by the disk. The format is as follows:

```
nnnn=ALLOCATED SEGMENTS
nnnn=AVAILABLE SEGMENTS
nnnn=VOLUME BLOCKS
nnnn=VOLUME ENTRIES
nnnn=SCRATCH ENTRIES
nnnn=CONTINUATION BLOCKS
nnnn=BASE ENTRIES
```

The report generated for the volume directory contains the following sections:

- Scratch volumes, sorted by serial number within type.
- Volumes with the MATCHONLYSERIALNO attribute, sorted by serial number within type. These volumes are recognized by serial number alone; the system does not check the family names.
- Expired volumes, sorted by serial number within type (TAPE or PETAPE). The LISTVOLUME utility computes the expiration date of each volume by using the the SAVEFACTOR and CREATIONDATE attributes of each volume.
- Valid volumes that contain files, sorted by serial number within type. These volumes have not yet expired and are assumed to have valid file data.
- Scratch volumes, sorted by family. The system lists all the scratch volumes.

- Volumes with the MATCHONLYSERIALNO attribute, sorted by title. These volumes are recognized by serial number alone. The display format is as follows:

```
Volume name  KIND = <tape, petape>
             CREATED: <creation date>
             EXPIRES: <expiration date>
             BASE or EMPTY
             CREATED ON SYSTEM: <system serial number>

             [volume serial #] reel #, ...
```

- Expired volumes sorted by title. The following is a sample layout:

```
TEST         FAMILY OWNER: USER (usercode)
(family      CREATION DATE: SUNDAY MAY 18, 1987 (87138)
name)        EXPIRATION DATE: SUNDAY MAY 18, 1987 (87138)
             CREATION SITE: 9
             FAMILY TYPE: LIBRARY MAINTENANCE
             UPDATE TIMESTAMP: MONDAY MAY 19, 1987 (87139)
                               AT 14:Ø2:17
             SECURITY: PUBLIC - USAGE: IN
             GUARDFILE: S1/S2 ON XYZPACK

             (PETAPE) [ØØØ123] #ØØØ1 TEMPORARY, RESTRICTED
             (TAPE) [ØØØ456] #ØØØ2 TEMPORARY, NOT-RESTRICTED
```

- Valid volumes sorted by title. The reporting format is identical to the format for expired volumes sorted by title.
- Volume directory statistics. These statistics summarize the total of various volume entries as follows:

```
nnnn=VOLUME ENTRIES
nnnn=SCRATCH ENTRIES
nnnn=MATCHONLYSERIALNO ENTRIES
nnnn=MISSING ENTRIES
nnnn=BASE ENTRIES
```

Running the LISTVOLUMELIB Utility

If your installation has a volume library, the LISTVOLUMELIB utility reports the status of all the volumes in the library. To run the program, simply enter a WFL *RUN SYSTEM/LISTVOLUMELIB* statement. The report generated for the volume library contains the following sections:

- The date that the LISTVOLUMELIB utility was run and the disk unit number that contains the volume library structure.
- Scratch volumes, sorted by serial number within type.
- Expired volumes, sorted by serial number within type (DISK, TAPE, PACK, PETAPE). The LISTVOLUMELIB utility computes the expiration date of each volume by using the SAVEFACTOR and CREATIONDATE attributes of each volume.

- Valid volumes that contain files, sorted by serial number within type. These volumes have not yet expired and are assumed to have valid file data.
- Expired volumes, sorted by title. The following information is printed for each volume:

```
Volume name= KIND = <disk, tape, pack, petape>

             CREATED: <creation date>
             EXPIRES: <expiration date>
             BASE or EMPTY
             CREATED ON SYSTEM: <system serial number>

             [volume serial number] reel number, ...
```

- Valid volumes, sorted by title. The reporting format is as follows:

```
Volume name= KIND = <disk, tape, pack, petape>

             CREATED: <creation date>
             EXPIRES: <expiration date>
             BASE or EMPTY
             CREATED ON SYSTEM: <system serial number>

             [volume serial number] reel number, ...
```

- Volume library statistics. This part of the report shows disk space consumed by the volume library and used by the disk. The format is as follows:

```
nnnn=ALLOCATED SEGMENTS
nnnn=AVAILABLE SEGMENTS
nnnn=VOLUME BLOCKS
nnnn=VOLUME ENTRIES
nnnn=SCRATCH ENTRIES
nnnn=CONTINUATION BLOCKS
nnnn=BASE ENTRIES
```

Section 4
Archiving Disk Files

The archive subsystem is designed to assist you in protecting the disk files of your installation, and in using your disk and tape resources more efficiently. This subsystem enables you to copy files to backup tapes, to transfer archived files between backup tapes, to restore archived files to disk, and to maintain a record of the names, locations, and attributes of archived files in archive directories. The archive subsystem is available to anyone using an A Series system, provided he or she has the required security access privileges.

The discussions in this section describe features and commands of the archive subsystem and how you can use them. After reading this section you will be able to do the following:

- Identify the components of the archive subsystem.
- Identify the basic functions performed by each of the archive subsystem commands, and use the archive subsystem commands.
- Find information on backup files in archive directories.
- Understand the primary function of the archive support library, and how you can modify it to suit your installation's needs.

The features and capabilities provided by the archive subsystem are compared with those of the catalog subsystem in Section 6 of this guide.

Components of the Archive Subsystem

The archive subsystem consists of three main components. These components include commands that perform library maintenance operations, archive directories for each online disk family, and a support library that determines whether or not files can be copied by an archiving operation. Each of these components is described, generally, as follows:

- You can use archive subsystem commands to copy disk files to backup tapes, to restore backup files to disk, and to merge backup files from many tapes to a single backup tape or tape set.
- Archive directories are disk directories that record the names, locations, and attributes of disk files that have been copied through WFL archive operations to tape. The archive record for any particular disk file keeps track of the four most recent archive backups of the file. The archive subsystem maintains a separate archive directory for each online disk directory. These directories reside on the DL catalog family. System commands you can use to manipulate archive directories are described under "Reviewing or Changing File Information in the Archive Directory" later in this section.

- The support library is used by the archive subsystem to control which files are copied during an archive operation. That is, the archive support library can reject files the system presents to it for archive operations, based on various selection criteria. Consequently, the support library is sometimes called the *selector library.* The standard symbolic file for the selector library is the *SYMBOL/ARCHIVESUPPORT file. The standard code file for this library is the *SYSTEM/ARCHIVESUPPORT file. Selector libraries are discussed under "Modifying the Archive Support Library" later in this section.

AUTORESTORE Task Attribute

Another feature of the archive subsystem is called the AUTORESTORE task attribute. This feature enables the archive subsystem to automatically reload a copy of an archived file from a backup tape to disk if a program encounters a "NO FILE" condition during an archive file search. However, the archive subsystem can reload the file if the following conditions are satisfied:

- The job requesting the missing file and the file itself have the same usercode.
- A backup record for the missing file exists in the archive directory for the disk family.
- The AUTORESTORE task attribute is set to TRUE.

You can control the AUTORESTORE task attribute with the AUTORESTORE system option. Following are the valid values for the AUTORESTORE system option:

Value	Function
YES	Sets the default setting of the AUTORESTORE task attribute to TRUE.
DONTCARE	Sets the default setting for the AUTORESTORE task attribute to FALSE.
NEVER	Prevents the ARCHIVE/AUTORESTORE processes from starting, no matter what the value of the AUTORESTORE task attribute is.

If a "NO FILE" condition occurs and your installation or task is not using the AUTORESTORE feature, you see messages identifying the tape on which the missing file resides. You can respond by loading the appropriate tape and issuing a WFL *COPY* statement. This process is described in greater detail in the *A Series System Operations Guide.*

Note: *If you have removed a file from disk, but you have not purged the backup records that refer to it from the archive directory, the AUTORESTORE process can restore that file to disk, provided a copy of it still exists on the archive backup medium. Therefore, if you want to remove an archived file permanently, you must also purge its records from the archive directory with the* ARCHIVE PURGE *command.*

Files and File Searches

The archive subsystem uses resident and nonresident disk files in its operations.

A disk file is recognized as nonresident in the following context:

- The flat directory for its disk family does not include a file header for the file.
- The archive directory contains at least one backup record for the file.

A disk file is recognized as resident if it resides on disk; this is so even if the archive directory does not include a backup record for the file.

Note: *Operations initiated by the WFL statements* REMOVE, CHANGE, COPY, *and* ADD *do not affect nonresident, archive backup copies of files in any way.*

Whenever the archive subsystem creates a backup copy of a file or transfers an archived file from one backup tape to another, it creates or updates a record of the transaction in the archive directory. This action enables the archive subsystem and you to locate nonresident backup copies of files. Therefore, the presence or absence of records in the archive directory affects the file search processes the archive subsystem performs.

When a program attempts to open an existing disk file, when a WFL job executes either a RUN or a PROCESS statement, or when a program attempts library linkage, the disk subsystem begins a search operation to locate the file that the operation requests. This process is known as a *multiple directory search* and occurs under any of the following circumstances:

- The task that is searching for a file runs under a usercode, but it requests a file without explicitly specifying its usercode in parentheses or the asterisk (*) usercode. In this case, the system searches for files in the directory that is associated with the usercode of the requesting task. If the specified file is not found, a second search takes place in the asterisk (*) directory.
- The task is running with a family substitution specification that includes an OTHERWISE clause. This causes the first search to occur on the primary substitute family name. If the search is unsuccessful, a second search occurs on the alternate substitute family name.
- Both conditions listed above apply. In this case, the search for a disk file is performed in the following order:
  - The primary family under the usercode directory of the requesting task is searched.
  - The primary family under the asterisk (*) directory is searched.
  - The alternate family under the usercode directory of the requesting task is searched.
  - The alternate family under the asterisk (*) directory is searched.

  This process is known as a *four-way search*. If at any point in the search the file is found, the process stops and the archive subsystem returns the found file.

If the disk subsystem discovers an archive backup record for a file at anytime during a search, the search process terminates, and a "NO FILE" condition results. If the AUTORESTORE feature is enabled for the task, the disk subsystem discovers an archive reference to a file, and the system is not trying to locate a library code file, the system responds by reloading a copy of the file to the disk from the backup tape.

Using Archive Subsystem and WFL Statements

Collectively, the archive subsystem statements perform the following actions:

- Create backup copies of resident disk files.
- Free disk space for other uses.
- Merge archive backup files onto a single backup tape or tape set.
- Restore backup files to disk.
- Purge records from the archive directory of a disk family.

You can enter the archive subsystem statements at an ODT, or you can include them as statements in WFL jobs. In most instances, you probably issue the archive statements through WFL jobs. Therefore, these statements are referred to as WFL statements throughout the rest of this section.

The names of the archive subsystem statements and the functions they perform are summarized in Table 4-1. In many installations, use of these statements on files under usercodes other than your own is limited to those whose security access is *privileged.* However, a WFL job that is started at an ODT without a usercode can run these statements as well.

Table 4–1. Archive Subsystem Statements

Statement Name	Description of Function
ARCHIVE FULL, ARCHIVE DIFFERENTIAL, and ARCHIVE INCREMENTAL	Copy resident disk files to library maintenance tapes.
ARCHIVE MERGE	Merges files from one or more archive backup tapes to a single tape or tape set.
ARCHIVE PURGE	Removes the backup records for specific files or directories from the archive directory of the specified disk family. This statement does not affect resident disk files in any way.
ARCHIVE RESTORE and ARCHIVE RESTOREADD	Restore files that have been backed up through the archive subsystem from library maintenance tapes.
ARCHIVE ROLLOUT	Selects and copies disk files to backup tape, and then removes the original disk files from disk. This statement is intended to free disk space for other uses.

Except for the *ARCHIVE RESTORE* and *ARCHIVE RESTOREADD* statements, all of the statements shown in Table 4-1 cause the archive subsystem to modify records in the archive directory of the specified disk family. More information on archive records is provided in this section under "Reviewing or Changing File Information in the Archive Directory." Also, the actual layout of the archive records is presented in Appendix A of this guide.

The following discussions provide explanations and examples of each of the WFL statements. Railroad syntax diagrams of these statements are not provided in this guide. If you need complete syntax information, with full explanations of the various statement options and attributes, refer to the *A Series Work Flow Language (WFL) Programming Reference Manual.* To find examples of general operations you can perform through the archive subsystem, refer to the *A Series System Operations Guide.*

ARCHIVE Backup Statements

These statements copy resident disk files to a backup tape. You can use these statements to back up some or all of the disk files on a family; the number of files the archive subsystem selects for backup depends on which of the three statements you use. In any case, you can use the backup tape files created by this statement as input to the *ARCHIVE MERGE*, *ARCHIVE RESTORE*, and *ARCHIVE RESTOREADD* statements, and to the *COPY* and *ADD* WFL statements.

You can use these statements as you would use ordinary library maintenance commands to copy files from disk to tape. That is, first you specify a list of file and directory names that you want to be copied and then you specify the name of the disk family and, optionally, the output tape name. When the system has accepted the file and directory names you specify, the system locates the files and decides which of them to copy. This decision is based on the form of the ARCHIVE backup statement you have used.

The functions performed by each of the three forms are described as follows:

- The *ARCHIVE FULL* statement copies all resident disk files in the named family to one or more backup tapes.
- The *ARCHIVE DIFFERENTIAL* statement copies to one or more backup tapes the resident disk files that have been changed or added to the family since the last *ARCHIVE FULL* statement was executed.
- The *ARCHIVE INCREMENTAL* statement copies to one or more backup tapes only those resident disk files that have been changed or added since the last archive backup procedure was performed. In contrast to the *ARCHIVE DIFFERENTIAL* statement, this statement does not copy files solely on the basis of the last *full* archive operation.

The selection for an archive backup uses three steps:

1. You specify the list of files, directory names, or both that are to be copied.
2. The system does not select resident files for backup if you specify the DIFFERENTIAL or INCREMENTAL option and the appropriate backup files already exist.

3. The archive support library determines which of the candidate files can and cannot be archived to tape.

The standard support library resides in the SYSTEM/ARCHIVESUPPORT file. For each file it accepts for backup, it returns an "OK TO COPY" value. If your installation has compiled its own selector support library, you can use library equation to direct the file selection process through that library. Custom-tailored libraries of this kind are described in greater detail under "Modifying the Archive Support Library" later in this section.

As each file is copied to tape, the archive subsystem creates or updates a record in the archive directory of the associated disk family. The main function of this directory is to record the names and locations of files you have archived on tape. If your installation is using the catalog subsystem and you copy a cataloged file from a volumed disk to an archive tape, the archive process creates two backup records: one for the archive directory, and one for the catalog.

If you do not specify a disk family name in an archive backup statement, the archive subsystem backs up each online disk family. The files for each family are copied to a separate backup tape. If you do not specify the name of the backup tape to which you are archiving the files, the archive subsystem assigns one for you. The name consists of the first 12 characters of the disk family name, and the current year and day (in the form YYDDD).

Examples of ARCHIVE Backup Statements

The examples that follow show archive backup statements that use the various syntax options. In all cases, the archive subsystem assumes that the disk files it copies are on disk pack media. A syntax error results if you identify the source medium as KIND=TAPE, or assign the SECURITYTYPE, SECURITYUSE, or FAMILYOWNER attributes in the FROM clause of an archive statement.

Example 1

This example creates archive backup copies of all resident files under the KATZ usercode on the disk family DATABANK. Because the *ARCHIVE DIFFERENTIAL* statement is specified, only files that have been updated or added since the last *ARCHIVE FULL* statement was executed are copied.

```
ARCHIVE DIFFERENTIAL (KATZ)= FROM DATABANK;
```

Because this example does not specify a backup tape name, the archive subsystem automatically provides a name. For example, if this statement is run on day 300 of the year 1990, the tape name is DATABANK90300 (that is, the family name—or up to 12 characters of it—and the current year and day).

Example 2

This example shows a statement that starts a full archive backup. In this case, the family name is DATABANK and the backup tape name is DATABACKTAPE. File

security attributes are applied to the backup tape by the InfoGuard subsystem, so that any user, privileged or not, can restore the files from the tape.

```
ARCHIVE FULL *= FROM DATABANK TO DATABACKTAPE (KIND=TAPE,
      SECURITYTYPE=PUBLIC, SECURITYUSE=IN);
```

Example 3

This example shows a statement that starts an incremental archive backup. Files are selected for backup only if they have been updated or added since the last archive backup operation was performed; whether or not the backup operation was differential or full is irrelevant to the file selection process. Because the family name is not provided, the archive subsystem performs an incremental backup on each online disk family in turn. The archive subsystem creates a separate backup tape for each disk family. The library equation directs the archive subsystem to use the selector support library SUPPORTLIB to approve the files that are chosen by the disk subsystem.

```
ARCHIVE INCREMENTAL;
  LIBRARY SELECTOR (LIBACCESS=BYTITLE, TITLE=SUPPORTLIB ON PACK);
```

Differences between Differential and Incremental Backups

The decision to perform incremental or differential backup procedures between full backup operations depends on the policies and requirements of your installation. While the *ARCHIVE INCREMENTAL* and the *ARCHIVE DIFFERENTIAL* statements both provide protection of data stored in disk files, there are significant differences between them. The following discussion is presented to clarify these differences.

Suppose, for example, that you perform a full backup of all files on the DATABANK disk family each Monday by including the following statement in a WFL job:

```
ARCHIVE FULL *= FROM DATABANK TO MONDAY;
```

This operation copies all disk files from the DATABANK family to the MONDAY backup tape. Each Tuesday through Friday, you perform daily incremental backups with the following statement:

```
ARCHIVE INCREMENTAL *= FROM DATABANK TO DAILY;
```

These daily operations copy to the DAILY backup tape only those files that have been changed or added since the last archive backup was performed. Therefore, Tuesday's tape includes only the files that have been changed or added since Monday (when the full backup was performed); Wednesday's tape includes only the files that have been changed since Tuesday. By Friday, your library of backup tapes consists of one full backup of the DATABANK family and up to four incremental backup tapes if some files were changed or added each day.

To restore all files archived through Thursday, therefore, you must load Monday's tape, and each incremental tape from Tuesday through Thursday (a total of four tapes).

Suppose now that instead of using incremental backups Tuesday through Friday, you perform differential backups with the following statement:

```
ARCHIVE DIFFERENTIAL *= FROM DATABANK TO DAILY;
```

These daily operations copy to the DAILY backup tape all those files that have been changed or added since the last *ARCHIVE FULL* statement was executed (on Monday). Therefore, Wednesday's tape includes backup copies of all files copied on Tuesday's tape in addition to those files added or changed on Wednesday. By Friday, your library of backup tapes consists of one full backup of the DATABANK family and one differential backup tape; this is because the *ARCHIVE DIFFERENTIAL* statement causes the system to backup files cumulatively since the last full archive backup operation. An exception to this situation occurs if a file is changed or if a new file is added on a certain day of the week and subsequently removed from the disk before the backup is performed on Friday. To restore such a file, you need the backup tape created the day before the file was removed.

To restore all files archived to their status as of Thursday, therefore, you need to load only Monday's full backup tape and Thursday's differential backup tape.

ARCHIVE MERGE Statement

This statement transfers backup copies of files that reside on several backup tapes to a single tape. If there is insufficient space to merge all requested backup tape files to one tape, a reel switch condition results and the remaining files are copied to the next reel. This produces a single *tape set* of merged files. The primary purpose of this operation is to enable more efficient use of archive tape volumes.

Whether you merge files to a single backup tape or to a tape set, this statement is likely to free otherwise occupied tape volumes for other uses.

The *ARCHIVE MERGE* statement affects backup tape files only; resident disk files are not affected by it. This is true even if there are files on the disk family that have been added or updated since the last backup operation, or if a backup file you are merging has been restored to disk and updated. The *ARCHIVE MERGE* statement accepts as input any file that has been archived to a backup tape by an archive backup statement, the *ARCHIVE MERGE* statement, or the *ARCHIVE ROLLOUT* statement.

Note: *Use of the ARCHIVE MERGE statement usually requires privileged access. However, a WFL job can also run this statement if it is started from an ODT without a usercode. In any case, this statement affects all tape files that are backed up or rolled out to a backup tape through the archive subsystem.*

As a merge operation executes, it requests as input the backup tape files created by preceding archive processes. During the operation, the archive subsystem prompts you to load the input tapes on the tape drives twice: once while the system creates the output tape directory, and again when it copies the selected files.

If you are unable to load a tape during the first phase of a merge operation, the system displays the "NO FILE" message on the ODT. You can respond to this message by

entering the OF (Optional File) system command. If you do so, the merge process does not copy files from the *missing* tapes to the merge tape. Instead, it continues to refer to the older versions of these missing files in the archive directory. A complete description of the OF system command is provided in the *A Series System Commands Operations Reference Manual.*

Each time a file is copied to a tape, the corresponding archive record in the archive directory is updated to indicate the name and serial number of the new tape on which the backup copy resides. The archive directory maintains records of up to four backup copies for each file that is archived for a disk family. Although the system usually copies only the most recent generations of archived files from the most recently written tapes, it can copy earlier versions of files from older tapes. However, this situation occurs only under the following circumstances:

- The archive subsystem finds that the specified backup tape has been purged or overwritten, or the tape has been marked as destroyed. The subsystem determines this status by looking into the volume directory (if your installation has set the SECOPT TAPECHECK option to AUTOMATIC) or into the volume library in the catalog (if your installation runs its operations without the volume directory, but with the OP + CATALOGING option).
- Your installation uses a custom version of the archive support library, and it has selected an older version of a backup tape for the merge operation.

If you want to create a single backup tape that includes all resident and nonresident files, issue an *ARCHIVE INCREMENTAL* *= statement, followed by an *ARCHIVE MERGE* statement. This sequence of statements ensures that the output tape from the merge process contains complete backup copies of resident files that have been changed or added since the last archive backup statement was issued; it then merges the new and existing backup copies onto a single backup tape or tape set.

Examples of ARCHIVE MERGE Statements

The examples that follow show various constructions of the *ARCHIVE MERGE* statement.

Example 1

This example shows the basic required elements of the *ARCHIVE MERGE* statement. These elements include the keywords ARCHIVE MERGE, a FROM clause, the name of the disk family, and a TO clause. The FROM and TO clauses specify the names of the tape volumes involved in the operation. Other options are available as well; they are explained fully in the *A Series Work Flow Language (WFL) Programming Reference Manual.*

```
ARCHIVE MERGE FROM DATABANK TO DATATAPE;
```

Example 2

This example requests that all backup copies of files for the DATABANK family be merged. The *& DSONERROR* option directs the archive subsystem to discontinue the

merge task if an error occurs while it is processing files. Because library equation is used, file selection occurs through the ARCSUBLIB selector library.

```
ARCHIVE MERGE & DSONERROR FROM DATABANK (SERIALNO="DBANK")
        TO ACCTBACK;
        LIBRARY SELECTOR (LIBACCESS=BYTITLE,
                          TITLE=ARCSUBLIB ON PACK);
```

Procedures resulting from archive options, such as *& DSONERROR* are described under "Using Task Variables and Archive Options" later in this section.

ARCHIVE PURGE Statement

This statement purges archive backup records from the archive directory, without affecting resident files in any way. The *ARCHIVE PURGE* statement does not affect backup information from catalog subsystem operations either. Purging archive backup records removes only the references to backup files. To remove an actual resident disk file, and all archive records associated with it, issue a WFL *REMOVE* statement followed by a WFL *ARCHIVE PURGE* statement. Both statements must identify the name of target file and the family name.

Note: *If you remove a file and do not purge its corresponding archive backup records, other archive subsystem operations continue to search for that file. The file continues to be available to the system, although indirectly; that is, ARCHIVE RESTORE operations and AUTORESTORE processes can still copy the file to disk.*

Examples of ARCHIVE PURGE Statements

The examples that follow show various constructions of the *ARCHIVE PURGE* statement.

Example 1

This example purges the archive backup records for all files on the DATABANK disk family.

```
ARCHIVE PURGE *= FROM DATABANK;
```

Example 2

This example purges all archive backup records associated with the usercodes CISCO and BJONES.

```
ARCHIVE PURGE (CISCO)=, (BJONES)=
     FROM DATABANK (KIND=PACK);
```

ARCHIVE RESTORE and ARCHIVE RESTOREADD Statements

These statements reload backup copies of disk files from tape to disk. Only backup files that were copied through one of the backup statements of the archive subsystem, which include the archive backup statements and the *ARCHIVE ROLLOUT* statement, can be restored to disk through the *ARCHIVE RESTORE* and *ARCHIVE RESTOREADD* statements.

Note: *In most installations, reloading files other than your own requires privileged access. However, it is possible to reload someone else's files if the job running the RESTORE or RESTOREADD process runs without a usercode and starts from an ODT.*

In general, the *ARCHIVE RESTORE* and *ARCHIVE RESTOREADD* statements enable you to recover disk files that have been damaged, lost, or inadvertently removed from disk.

The *ARCHIVE RESTORE* statement reloads the specified backup copies of files to disk even if generations of those files already exist on the destination disk. In contrast, the *ARCHIVE RESTOREADD* statement reloads only the specified files for which generations do not already reside on the destination disk. In either case, the archive subsystem selects files by comparing the files you request with records in the archive directory. As the subsystem selects files for the RESTORE or RESTOREADD processes, it builds a library maintenance copy request for all selected files. The request includes the correct tape names and serial numbers of the most recently created backup tapes.

The RESTORE and RESTOREADD processes do not affect the archive directory in any way. Consequently, you can remove a restored file and restore it to disk again, even if an intervening archive backup has not been performed on that file.

For *ARCHIVE RESTORE* requests that use the standard archive support library (*SYSTEM/ARCHIVESUPPORT), an older, archived generation of a file can overwrite a newer resident copy of that file. However, the archive subsystem usually restores files from the most recently written archive tapes. The archive subsystem can restore files from older backup tapes under the following circumstances:

- The archive subsystem finds that the contents of a specified backup tape have been purged or overwritten, or the backup tape has been marked as destroyed. The subsystem determines this by looking into the volume directory (if your installation has set the SECOPT TAPECHECK option to AUTOMATIC) or into the volume library in the catalog (if your installation runs its operations without the volume directory, but with the OP + CATALOGING option).
- Your installation uses a custom version of the archive support library, and that library selects an older version of a backup tape file for the RESTORE or RESTOREADD process.

Examples of ARCHIVE RESTORE and ARCHIVE RESTOREADD Statements

The examples that follow illustrate various uses for the *ARCHIVE RESTORE* and *ARCHIVE RESTOREADD* statements.

Example 1

This example restores the file named STATS/DATA under the usercode BOBS to the DATABANK disk family. Because the *& COMPARE* archive option is specified, the file is copied to disk and then compared with its backup version to ensure the integrity of the RESTORE process.

```
ARCHIVE RESTOREADD & COMPARE (BOBS)STATS/DATA
     TO DATABANK
```

For this operation, the archive subsystem retrieves the archive record for the file (BOBS)STATS/DATA ON DATABANK. The subsystem then determines the name and serial number of the most recent backup tape that contains a copy of the file, and it calls library maintenance to copy the file from the backup tape to the disk family DATABANK.

Example 2

This example restores all backup copies of files to the disk family ACCTFILS. For this operation, the archive subsystem usually requests that you load various archive backup tapes as the COPY process proceeds.

```
ARCHIVE RESTORE *= TO ACCTFILS;
```

ARCHIVE ROLLOUT Statement

This statement moves selected disk files to archive backup tape media. The primary function of this statement is to free disk space for other uses, particularly when resources are limited. You can use this statement in one of three ways:

- By specifying the amount of disk space—in sectors—to be made available
- By specifying that all files belonging to specified users be rolled out
- By specifying a percentage of disk space to which each user is to be reduced

You specify this percentage through the DRC option of the *ARCHIVE ROLLOUT* statement. The DRC option is available to you even if your installation does not actively run the DRC subsystem. This option affects only those users for whom DRC limits are assigned in the USERDATAFILE.

Whether you roll out files to tape by specifying a number of sectors or by using the DRC option, you are not required to name the backup tape to which files are transferred. If you do not specify a tape name, a name is assigned for you. It consists of the first 12 characters of the disk family name from which files are being transferred, followed by the current year and day (in the form YYDDD).

As each selected file is rolled out to tape, the archive subsystem records the transfer in the archive directory. You can use rolled-out files as input to the *ARCHIVE RESTORE*, *ARCHIVE RESTOREADD*, and *ARCHIVE MERGE* processes and to ordinary library maintenance processes.

Note: *If the archive directory includes references to archived copies of the files your rollout process is requesting, the rollout process removes the resident versions of the files without recopying those files to tape. Therefore, it is possible to complete a rollout process without receiving a system request for an output tape.*

Before the archive subsystem actually performs a rollout operation, it evaluates files for possible selection. This process is based on the usercode of the job that is running the *ARCHIVE ROLLOUT* statement, as follows:

- If you do not list specific usercodes in your *ARCHIVE ROLLOUT* statement, only the files under the usercode of the job that is running the rollout process are evaluated for possible selection.
- If you start the job that runs the *ARCHIVE ROLLOUT* statement at an ODT without a usercode, only the files under the asterisk (*) directory are evaluated for selection.
- If your usercode permits privileged access and you specify ALL USERS in the *ARCHIVE ROLLOUT* statement, all files are evaluated for possible selection.

To select files for the rollout operation, the rollout procedure reviews the specifications and options used in the *ARCHIVE ROLLOUT* statement and calls the selector library. This library determines which of the files belonging to the specified usercodes are rolled out. The file selection criteria used by the standard selector library (SYSTEM/ARCHIVESUPPORT) when the ALL FILES option is not specified include the following:

- The size of the file
- The value of the LASTACCESSDATE attribute
- The value of the FILEKIND attribute
- The value of the SAVEFACTOR attribute
- Whether or not the file is currently in use with the EXCLUSIVE file attribute set to TRUE

Given these criteria, the archive selector library tends to select files with the following characteristics for rollout operations:

- Larger files
- Files that have not been used recently
- Files with smaller SAVEFACTOR attribute values

Smaller files, recently accessed files, and files with larger SAVEFACTOR attribute values are less likely to be rolled out to a backup tape. Files that are opened exclusively when the rollout is requested are not rolled out unless the ALL FILES option is specified.

In any case, even if the ALL FILES option is specified, the standard selector library never selects guard files or critical system files for rollout operations. When

the DRC option is used, files whose names begin with an asterisk (for example, *SYSTEM/ALGOL) are not selected for rollout. These files are not subject to the space limitations set in the DRC subsystem.

Specifying the SECTORS Option or Using the DRC Option

You can use the *ARCHIVE ROLLOUT* statement to make otherwise in-use sectors available in either of two ways. You can use the SECTORS option of this statement, or use can use the DRC option. This discussion describes the differences between these two approaches to freeing disk space.

Notes:

- *In general, use of the ARCHIVE ROLLOUT statement on files other than your own requires privileged access to files.*
- *ARCHIVE ROLLOUT statements that include SECTORS specifications do not require DRC limits to have been set in the USERDATAFILE.*
- *If you use the DRC subsystem option in this statement, the rollout operation affects only the files under usercodes for which DRC limits are assigned in the USERDATAFILE. A more detailed discussion of the DRC subsystem is provided in Section 9.*

When you use the *ARCHIVE ROLLOUT* statement with the SECTORS option to free a specified number of in-use sectors on a disk family, the archive subsystem copies files to tape or removes them from disk to satisfy your request. If you have specified more sectors than are already in use under your usercode on the disk family, the archive subsystem selects all of your files for the rollout operation. In this case, the operation is executed as if you had specified ALL FILES in the statement.

Examples of ARCHIVE ROLLOUT Statements

The examples that follow illustrate various uses for the *ARCHIVE ROLLOUT* statement.

Example 1

This example shows an *ARCHIVE ROLLOUT* statement that frees a total of 3,000 sectors of disk space by selecting files on disk pack PACK1 under the usercodes OCAROLAN, LECLERQ, or both.

```
ARCHIVE ROLLOUT 3ØØØ SECTORS SELECT (OCAROLAN),
        (LECLERQ) FROM PACK1 TO TAPE1;
```

Because the statement specifies two usercodes, running it requires a privileged usercode. A user without privileged access can specify only his or her own usercode in the usercode list of any *ARCHIVE ROLLOUT* statement. Notice that this syntax does not include an equal (=) sign to specify files.

Example 2

This example uses the ALL USERS option in the SELECT clause. If this statement is run under a privileged usercode, the archive subsystem either removes or copies files to tape until 10,000 sectors of in-use disk space are made available on PACK1. For instance, if there are 5,000 sectors available on PACK1 before the rollout, 15,000 sectors are available after the rollout process is complete. The archive subsystem selects files for this operation from all users; it does *not* free 10,000 sectors under each user on the family. All files, including those beginning with an asterisk (*), are possible candidates for rollout. If the usercode running this statement does not have privileged access, an error message results and the operation terminates.

```
ARCHIVE ROLLOUT 10000 SECTORS SELECT ALL USERS FROM PACK1 TO TAPE;
```

Example 3

This example uses the ALL FILES option. The only files exempt from rollout when you use this option and the standard selector library are guard files and critical system files.

```
ARCHIVE ROLLOUT ALL FILES (OLDUSER) FROM PACK1 TO TAPE;
```

Use of the ALL FILES option is recommended when a usercode at your installation is no longer valid and you want to make backup copies of the files under that usercode before you remove them from the pack.

Example 4

The DRC option of the *ARCHIVE ROLLOUT* statement examines the DRC subsystem user disk space limits for a disk family. These limits are set through either the FAMILYLIMIT or the OTHERFAMILYLIMIT attribute of the SYSTEM/USERDATAFILE.

A DRC rollout operation removes or transfers files associated with the usercodes that you specify until the in-use disk space occupied by users is reduced by the requested percentage. The rollout procedure uses the following formula to determine the number of sectors it should roll out:

```
((FAMILYLIMIT value) * (100 - percentage))/100
```

For example, suppose that you need to make available 25 percent of the in-use disk sectors occupied by files under your usercode. To do so, you can use the following statement:

```
ARCHIVE ROLLOUT DRC 25 FROM PACK1 TO TAPE;
```

Suppose, too, that the amount of disk space your files can occupy is 10,000 sectors (as established by the FAMILYLIMIT attribute), and your files are currently using 9,000 of those sectors. In this case, the rollout procedure applies the following formula to reduce your in-use disk usage:

```
((10,000)(100 - 25)) / 100
```

The resulting value is 75 percent of 10,000 sectors, or 7,500 sectors; but because your allocated files are already 1,000 sectors below your limit, the system rolls out only another 1,500 sectors.

If you issued the same statement, but you had fewer than 7,500 sectors in use, the system would not select any of your files for rollout.

In general, the following rules apply to system use of the FAMILYLIMIT and OTHERFAMILYLIMIT attributes in the USERDATAFILE during a DRC rollout operation:

- If the FAMILYLIMIT value has not been set for a specified usercode, the rollout operation uses the OTHERFAMILYLIMIT value for that usercode.
- If neither the FAMILYLIMIT nor the OTHERFAMILYLIMIT value has been set for a specified usercode, or if there is not an entry for a usercode in the SYSTEM/USERDATAFILE, disk files for that usercode are not rolled out.
- If the FAMILYLIMIT value is negative, the rollout operation for the specified usercode does not take place. If a FAMILYLIMIT value has not been set and the OTHERFAMILYLIMIT value is negative, the rollout operation for the specified usercode does not take place. A negative value for either attribute allows a user unlimited disk space for his or her permanent files.

If you need more information on how to set disk space limits through the FAMILYLIMIT or OTHERFAMILYLIMIT attribute, refer to the discussion on using SYSTEM/MAKEUSER in the *A Series Security Administration Guide.*

The following example provides another illustration of the DRC option in a rollout operation:

```
ARCHIVE ROLLOUT DRC 3Ø OF ALL USERS FROM PACK1 TO TAPE;
```

If this statement is run under a privileged usercode, files are selected for rollout from all users on the PACK1 family. For each user, the amount of space a user can use is reduced to 70 percent of the limits set by the FAMILYLIMIT or the OTHERFAMILYLIMIT attribute for that usercode.

If the above statement is run under a usercode for which privileged access is not allowed, files are not selected for rollout and the following message is displayed:

```
ARCHIVE ROLLOUT, NO FILES SELECTED FOR COPYING ON PACK1
```

Percentage values are always expressed as integer values. Usually, these values range from 0 to 100.

- If you enter 0 as the percentage value, files are rolled out until each user's disk space usage is less than or equal to the limit set in the FAMILYLIMIT attribute.
- If you enter 100 or a number greater than 100, the system rolls out all files associated with the specified usercode, except for guard files and critical system files.

- If you enter any integer value between 0 and 100, in-use disk space is reduced by the rollout operation, based on the values associated with the FAMILYLIMIT or OTHERFAMILYLIMIT attribute.

Using Task Variables and Archive Options

You can monitor the status of most archive subsystem processes, including the archive backup processes and the MERGE, RESTORE, RESTOREADD, and ROLLOUT operations, by using task variables with your WFL archive statements. If you use these variables, and an error occurs during the archive operation, the TASKVALUE attribute of the task variables shows a value other than zero. If you need specific information on task variables and attributes, refer to the *A Series Task Management Programming Guide* and the *A Series Task Attributes Programming Reference Manual.*

When an archive subsystem operation has completed backing up, merging, or rolling out files, the resulting library maintenance tape is rewound and unloaded automatically. If errors have occurred during the operation, however, the output tape might not include all the files you intended to backup, merge, or rollout.

You can use the *& DSONERROR*, *& VERIFY*, and *& COMPARE* options with any archive statement except the *ARCHIVE PURGE* statement. Each option is described in greater detail in the *A Series Work Flow Language (WFL) Programming Reference Manual.*

If you include the *& DSONERROR* option in a statement, and an error occurs during processing, the archive subsystem discontinues the operation it is performing. It then purges any tape it is creating and the archive records it has written during the operation from the archive directory of the disk family.

If you include the *& VERIFY* option in a statement, and an error occurs during the verification phase of the operation, the system displays a message on the ODT asking whether or not it should purge the output archive tape. You can respond to this message by issuing the FR (Final Reel) system command. This response causes the archive subsystem to

- Stop backing up, merging, or restoring files.
- Purge the output tape it is creating.
- Remove all archive backup records it has created during the operation from the archive directory of the disk family.

Examples for Using Task Variables and Archive Options

The examples that follow show two archive subsystem statements in which these options are used.

Example 1

This statement archives all files on the DATABANK disk family that have been added or updated since the last full archive backup operation was performed. The *& VERIFY*

option causes the system to write checksums on the tape for each file copied. The system then rewinds the tape, reads the files, and tests the checksum values.

```
ARCHIVE DIFFERENTIAL & VERIFY *= FROM DATABANK
    TO BACKTAPE1;
```

Example 2

This example merges archived files from the DATABANK disk family that are associated with the usercode BJONES onto a single backup tape or backup tape set. The *& DSONERROR* option causes the system to discontinue the operation if an error occurs during processing. If an error that is not recoverable occurs, the output tape is purged and any newly created archive backup records in the related archive directory are removed.

```
ARCHIVE MERGE & DSONERROR (BJONES)= FOR DATABANK;
```

Reviewing or Changing File Information in the Archive Directory

You can review some of the file information an archive directory maintains in any one of the following ways:

- Issue the PD (Print Directory) system command to list backup information for files in the archive directory. This command lists disk file and archive information on the ODT.
- Invoke the FILEDATA system utility and use one of the following requests to obtain a report:
  - An ARCHIVEINFO request to produce a report of resident and nonresident files and backup file information from the archive directory for the files.
  - A BACKUP request to produce a report of backup tapes in use for files of a specified family.
  - The ARCHIVE modifier in a CODEFILEINFO, FILENAMES, CATALOGINFO, ATTRIBUTES, or STRUCTUREMAP request to cause reports to include displays of archive backup information for files.

  The FILEDATA system utility is described in the *A Series System Software Utilities Operations Reference Manual.*

The GETSTATUS and SETSTATUS calls enable you to perform additional functions as well. For example, with these calls you can do the following:

- You can retrieve archive information programmatically for specific files or for all files under a named directory.
- You can program GETSTATUS calls to copy to a disk file all archive records in the archive directory for a disk family, or all archive records under a specified usercode for a disk family. You can then read the copied records and print detailed or special reports of the information they contain. Reports created in this way offer more detail than do reports created through the FILEDATA system utility.
- You can use GETSTATUS calls to create backup copies of archive directory records.
- You can program a SETSTATUS call to remove an existing archive record from the archive directory of a disk family. This kind of call can purge damaged, obsolete, or unused archive backup records from the archive directory of a disk family. This call corresponds functionally to the ARCHIVE PURGE statement.
- You can program a SETSTATUS call to add a new file record to an archive directory for a given disk family. This call can also restore missing or damaged records to an archive directory, or logically move a file from one family to another.

If you need specific information on the available GETSTATUS and SETSTATUS calls, refer to the *A Series GETSTATUS/SETSTATUS Programming Reference Manual.*

Appendix A of this guide provides the layout of the archive directory records.

Modifying the Archive Support Library

The archive support library to which you have mapped function names with the SL (Support Library) system command has final control over whether a file can or cannot be copied, merged, restored, or rolled out by the archive subsystem.

In its unmodified form, the standard archive support library that is issued with each A Series mark release can select any files to which the user has access for any archive operation. The only exception to this rule occurs when *ARCHIVE PURGE* statements are executed; the archive support library is not used for these operations.

The standard file selection procedure meets the needs of most conventional installations. If, however, the needs of your installation are different, it is possible to modify the standard archive support library, or to define a new one. For example, you can define a custom archive support library that prevents ROLLOUT operations from selecting files that are in use, or that prevents archive backup procedures or other ARCHIVE operations from selecting database files.

The *SYSTEM/ARCHIVESUPPORT file is the standard code file for the archive support library; the *SYMBOL/ARCHIVESUPPORT file is the symbolic file. If you choose to modify the standard archive support library, you can make those changes in the symbolic file. You must then recompile the file with the DCALGOL compiler.

If you want an ARCHIVE operation to use a library other than the library to which you have mapped functions with the SL system command, you must include a library equation with your archive subsystem statement. The library equation enables you to specify other libraries by title. The following example shows the format of the library equation as you might use it:

```
ARCHIVE ROLLOUT 1000 SECTORS (X) FROM DISK;
  LIBRARY SELECTOR (LIBACCESS=BYTITLE,TITLE=MY/ARCLIB ON PACK);
```

Notice that the library equation specifies the internal name *SELECTOR*, and uses the LIBACCESS and TITLE attribute specifications. Together these attributes define access to the library as *BYTITLE*, and specify the name of the library code file as *MY/ARCLIB* on the disk family. For more information on libraries in general, or on library equations and attributes, refer to the *A Series Task Management Programming Guide*.

About Archive File Selection

When you issue any of the archive subsystem statements, either through a WFL job or at an ODT, the system begins the operation by locating the files specified in the statement. During this process, the system locates the disk family and the disk file headers or archive records for all specified files, directories, and usercodes. If the system cannot locate these elements, it issues warning messages; it issues error messages if the statement specifies files that are exempt from archive subsystem operations, such as BADDISK and SYSTEMDIRECTORY files.

Because operations performed through the *ARCHIVE FULL*, *ARCHIVE DIFFERENTIAL*, *ARCHIVE INCREMENTAL*, and *ARCHIVE ROLLOUT* statements manipulate resident disk files, these processes require disk file headers to be present. In contrast, MERGE and RESTORE operations require archive backup records to be present.

The system calls the appropriate selector library for each file that it finds. This library can be the standard version in the *SYSTEM/ARCHIVESUPPORT file or a custom version. The selector library determines which of the files presented to it by the system can be used in the archive subsystem operation. The selector procedure selects the candidate files on the basis of the kind of archive operation that is executing, the name of the file, the disk file header information (if the file is resident), and any available information from the archive directory. The decision to select a candidate file can be delayed until all the files have been compared with the list of files in the selector library; this process can be useful during ROLLOUT operations in which attributes of the candidate files are compared to each other.

When the selector procedure has determined the files to be copied or added, the system constructs a library maintenance request to copy or add the chosen files. Based on the archive statement that is issued and the various parameter values defined in the selector procedure of the support library, files can be archived, merged, restored, or rolled out.

Note: *Keep in mind that the status of a file can change between the time it is selected and the time it is actually used in an archive operation. That is, the resident version of a file can be removed, updated, or overwritten between the time the system submits it to the selector procedure of the support library and the time the archive subsystem actually copies it during an archive operation.*

Selector Procedure and Parameter Values

The information presented here describes the parameters used by the selector procedure of the standard archive support library. If you are not already familiar with the layout of the words and fields associated with the parameters of the selector procedure, refer to the SYMBOL/ARCHIVESUPPORT file on your release tape. This file is a DCALGOL symbol file. The DCALGOL code file is the SYSTEM/ARCHIVESUPPORT file.

Any selector procedure used by the archive subsystem must be declared as type REAL, with the name ARCHIVESELECTOR. This declaration must also list the parameters CODES, SFN, DFHINFO, ARCREC, and MEM. An example of this declaration follows:

```
REAL PROCEDURE ARCHIVESELECTOR (CODES, SFN, DFHINFO, ARCREC, MEM);
                          VALUE           SFN;
ARRAY    CODES [Ø];   % DESCRIBES KIND OF CALL
POINTER        SFN;   % FILENAME IN STANDARDFORM
ARRAY  DFHINFO [Ø];   % DISK FILE HEADER ATTRIBUTES
ARRAY   ARCREC [Ø];   % ARCHIVE RECORD
ARRAY      MEM [Ø];   % ONE MEGAWORD SEGMENTED ARRAY
```

The ARCHIVESELECTOR procedure returns a single result of type REAL to the archive subsystem. Any changes the selector procedure makes in the contents of the parameters are ignored by the archive subsystem. For example, if the selector procedure changes a file name that the pointer parameter SFN refers to, the actual archive process does not change course and copy the file by its new name. Similarly, if the selector procedure changes some of the dates in the array parameter ARCREC, the actual archive directory record in which the original dates are stored is not affected.

The text that follows describes each of the five parameters shown in preceding example of the procedure heading.

CODES Parameter

The CODES parameter is an array. It defines, generally, the characteristics of a call. Specific elements in this array define

- Whether the procedure call is for a full, differential, or incremental backup, or for an archive rollout or other archive process. The kind of procedure is indicated in word 0, field [5:6] of the procedure. These code values are listed in Appendix A with the definitions under ARGTYPE.
- The usercode of the calling process, which is defined in substandard form in words 5 through 7 of the procedure. If the process is running without a usercode, these words consist of only zeros.
- The name of the disk family, which is defined in substandard form in words 9 through 11 of the procedure.

Word 4 of the CODES array contains the file number that is assigned to a file for ordering within the archive process. The value shown in word 4 is 1 for the first file, 2 for the second file, and so forth. If during the first call for a file the selector procedure

returns the real result 2 (which means *try again*), the procedure passes again over all files for which a 2 value is returned. During the second pass, bit [11:1] of word 0 is set to 1 and the same file number that was used in the first pass is placed in word 4 of the CODES array.

Note: During the second pass, the required information for some or all of the files that were to be reprocessed (those files for which a 2 value was returned) might be missing. This can occur when required disk file headers or archive records have been removed or purged. Under these circumstances, the subsystem does not call the selector procedure for those files during the second pass.

The CODES parameter also indicates whether the DFHINFO (disk file header information) parameter contains valid information and whether the ARCREC parameter contains a copy of the archive record for the chosen file. If the DFHINFO parameter contains information, bit [47:1] of word 0 of the CODES array contains the value 1; if the ARCREC parameter contains information, bit [45:1] of word 0 of the CODES array contains the value 1.

When archive record information exists, bits [35:2] of the CODES array indicate which archive tape is best to use. If the archive directory shows only one backup entry, bits [35:2] of word 0 receive either 0, 1, 2 or 3. If the archive directory shows two backup entries, bits [35:2] refer to the most recently written tape. However, it is possible for bits [35:2] to refer to an older backup tape. This can occur if there is a volume directory or a volume library for the tape that identifies the most recently written tape as purged, overwritten, or destroyed.

SFN Parameter

The SFN parameter points to the file name of the candidate file, in standard form.

DFHINFO Parameter

The DFHINFO parameter is an array that includes information on all file attributes for a file, as this information is recorded in the archive directory. If bit [47:1] of word 0 of the CODES array contains the value 1, the DFHINFO parameter contains information extracted from the disk file header for the resident disk file. This information describes the resident version of the file, including various timestamps, the file kind, and the size of the file.

ARCREC Parameter

If bit [45:1] of word 0 of the CODES parameter contains the value 1, the ARCREC parameter contains a copy of the archive record for the candidate file.

The layout of the archive directory records is provided in Appendix A.

MEM Parameter

The MEM parameter is a segmented array that is declared to be 1,048,575 words long. The archive subsystem passes this array to the selector procedure during archive operations. Its function is to retain information for each file between calls for other files. The archive subsystem does not store, change, or erase anything in the MEM array.

Because the standard support library (SYSTEM/ARCHIVESUPPORT) is defined as a shared-by-all library, the library cannot have any global memory of its own. The archive processing procedure in the MCP compensates by supplying the MEM parameter (a large storage array) for each independent archive process. Use of the MEM array enables the support library to *remember* file information for each archive process separately.

If your installation has modified the standard selector procedure to use the MEM array during differential, full, and incremental archive backup operations, you should be careful when specifying file names for selection. If you do not include a file or directory list and family name in the WFL statement, the operation executes sequentially, selecting files from each online disk family. For a multiple-family archive process, you can use the file number (word 4 of the CODES array) or the family name (words 9 through 11 of the CODES array) to determine which family is being processed and when processing began. At these transitions, you might want to erase some of the information you have stored in the MEM array that applies to the preceding family.

Selector Procedure and Returned Results

When the selector procedure determines whether a candidate file is to be copied in an archive operation, it returns a type REAL value, as described under "Selector Procedures and Parameter Values" earlier in this section. The REAL values indicate whether a file is to be copied, the tape on which an archived file resides, and if a second call is to be made after the first calls for all files have been made. Each possible REAL value is described as follows:

REAL Value	Description
1 & bkno [39:2]	A value of 1 indicates that the candidate file is to be copied. If the archive process is a rollout operation, a value of 1 indicates that the file can be either removed or copied and removed, as required. A *bkno* value in bits [39:2] is used by the system when the operation is a RESTORE, RESTOREADD, or MERGE request. This result identifies a backup tape number: if bits [39:2] show a value of 0, the backup tape number to be used is 0; if bits [39:2] show a value of 1, the backup tape number to be used is 1, and so on. The archive process ignores bits [39:2] if the archive directory shows only one backup tape for the file, and for all archive requests other than *ARCHIVE RESTORE*, *ARCHIVE RESTOREADD*, and *ARCHIVE MERGE*.

continued

continued

REAL Value	Description
2	Indicates that the archive selector procedure is postponing the decision to choose a file during its first pass over the candidate files. The archive subsystem performs a second call for all files yielding a value of 2 when it has completed its first call for all specified files. If during the second call the selector procedure again returns the value of 2 for a file, the file is rejected for the archive operation. That is, the file is not included in the executing archive operation. Therefore, a value of 2 during the second call for a file is functionally equivalent to a value of –1.
–1	Indicates that the selector procedure has rejected the file for inclusion in the executing archive operation. That is, the file is not to be copied or removed by the current operation.
All other values	Indicate that the selector procedure has rejected the file for inclusion in the current archive operation.

Standard Algorithms in the Selector Procedure

The following explanation describes the basic algorithms for file selection as they are defined in the SYMBOL/ARCHIVESUPPORT file.

For any archive process other than an archive rollout process, a value of 1 indicates to the archive process that the candidate file is to be copied. If the executing process is an archive RESTORE, RESTOREADD, or MERGE operation, and the selector procedure returns a value of 1 for a file, bits [39:2] are set to recommend a backup tape from which the file can be copied. Bits [39:2] are set according to the value of bits [35:2] of word 0 of the CODES parameter array.

When you issue an archive rollout request, the selector procedure passes twice over all candidate files. During the first pass, the procedure ranks each file according to the size of the file and the values associated with the LASTACCESSDATE and SAVEFACTOR attributes. If the operation does not specify the DRC option, the row sizes of the files are also considered. In general, larger files are assigned lower ranks.

Another important factor is the expiration date of a file. The selector procedure determines this date by adding the SAVEFACTOR value to the LASTACCESSDATE value. Files with later expiration dates are assigned higher ranks, while files with earlier expiration dates are assigned lower ranks. Files ranked lower are more likely to be rolled out by the archive subsystem.

The relative rank and size of each file are stored in the MEM parameter, as indexed by the FILE NUMBER parameter. The file number is supplied by the CODES array.

Between the first and the second passes, the selector procedure sorts the ranked files. The procedure then subtracts the sizes of the ordered files from the rollout goal, which appears in the CODES array, until the goal is reached. The *rollout goal* is the number of sectors or the percentage of disk space you specified in the archive statement.

During the second pass, the selector procedure chooses the rank-ordered files that fall below the specified rollout goal. Chosen files are either moved to tape or deleted from disk without first being copied. Files that exceed the specified criterion are left on disk.

In any case, the rollout algorithm does not select certain kinds of files, and can (in special cases) deviate from the selection process described above. The following list describes these files and special cases:

- The rollout algorithm does not select system files. This includes MCP code files designated with a CM (Change MCP) system command and archive directories.
- The rollout algorithm cannot select guard files. These files control access to other files. If you were to roll out a guard file, you would loose access to the disk files and tape volumes the guard file is protecting.
- The algorithm does not select files that are currently opened with exclusive access unless the ALL FILES option is specified.
- If you issue an *ARCHIVE ROLLOUT* statement or a WFL statement that specifies the ALL FILES option, the selector procedure selects all candidate files (except for the files listed above) without sorting them or issuing a second call.
- If your rollout statement includes the DRC option, the selector procedure ranks and sorts files for each usercode separately. In this case, each usercode is assumed to have a different rollout goal.
- If your rollout statement includes the DRC option, the selector procedure selects any file whose size exceeds the following formula:

```
((FAMILYLIMIT value) * (100 - percentage))/100
```

Section 5
Cataloging

All computer installations implement procedures to protect information they have stored in disk files. For additional protection, many installations keep extra copies of files in secure locations away from the site. This practice guards against loss of data from fire or other occurrences. Usually these backup files are kept on magnetic tape.

Although backup procedures such as these are necessary, keeping track of the resulting backup copies and their various versions can pose difficulties. The CATALOGING option of the Unisys A Series systems can simplify the file tracking process. This section describes the catalog subsystem and explains the following topics:

- Understanding how a cataloging system works
- Setting up a cataloging system
- Using a cataloging system
- Using tape security
- Rebuilding catalogs
- Creating and using backup copies of the catalog
- Replacing a damaged disk on a cataloging system

Understanding How a Cataloging System Works

The CATALOGING option provides an automated method of locating where backup copies of disk files and tape files are stored. The system handles cataloged disk files and cataloged tape files similarly, but this section focuses on cataloged disk files. This guide uses the term *resident* to refer to the primary copy (as opposed to a backup copy) of a file that is stored on disk, regardless of whether or not the disk is online. A file is *nonresident* if it is stored only on a backup tape or if it is a backup copy of a file and is stored on another disk family. Only one version of a file can be resident at one time. In this guide, the terms *resident* and *nonresident* do not pertain to the file attribute RESIDENT.

This discussion of how a cataloging system works is divided into two parts: "Catalog Components" and "File Generations."

Catalog Components

The disk file access structure discussion under "Disk File Access" in Section 1 noted that cataloging systems store the access structure in the file SYSTEM/CATALOG/<family index number>. It also noted that the SYSTEM/CATALOG file stores other information that pertains to cataloging.

The SYSTEM/CATALOG file contains two other structures:

- The volume library
- The catalog

The bulk of the SYSTEM/CATALOG file is made up of the catalog, which stores information about the location of resident backup copies of cataloged files. If a disk file is cataloged, the catalog keeps track of backup copies. Cataloged disk files can reside only on a disk volume that has been entered into the volume library. Cataloging can keep track of the backup copies of disk files only if the copies are stored on disks or tapes that have been entered into the volume library.

If a tape file is cataloged, the catalog keeps track of its location. Cataloged tape files can reside only on a tape volume that has been entered into the volume library. Refer to Section 3 for a description of the volume library.

File Generations

You might want to keep one or two backup copies of a file. You might also want to retain older versions of a file for historical or developmental reasons. Cataloging can be used to automatically keep track of these copies and versions.

The different copies of a file are referred to as the generations. The generation is determined by two file attributes, CYCLE and VERSION, and also the timestamp of the file. The system maintains the timestamp of each file automatically.

The CYCLE and VERSION file attributes are integer values that programs can use to distinguish each generation of a file. The higher the value of CYCLE, and the higher the value of VERSION within a particular CYCLE, the better the genealogy of the generation is said to be, relative to other generations of the file. If you do not assign values to CYCLE and VERSION, the default value is 1 for CYCLE and 0 for VERSION. CYCLE and VERSION can be used when you create a new file or access an existing file.

The timestamp is a system attribute for each file that the system maintains to show the last time the file was altered, including changes to its data, name, security status, and certain file attributes. If a file has never been altered, the timestamp shows the time and date the file was created.

Characteristics of a New File Generation

When a new generation of a file is created and made permanent, the system removes the previous resident generation of the file. If there are backup copies of the previous resident generation, an entry for that generation remains in the catalog. If there are no backup copies for the previous resident generation, the system deletes its catalog entry.

When a new disk file is created, the system cannot check the contents of the file to see if it really is a new generation of an existing cataloged file or if it is a completely different file. The system checks only to see that both the file name and the family name are the same. This concept is important to remember if CYCLE and VERSION are not being used, and you copy a disk file to backup media and then remove the resident disk file. If

you then create a different, unrelated file on the same family with the same name, the system automatically treats the new file as the newest generation of the other file that had the same file name.

Keeping Track of File Generations

Cataloging allows you to keep track of up to seven different generations of a file and up to two backup copies of each generation. The CATALOGLEVELSET define that is compiled into the MCP designates the catalog level. The catalog level determines how many generations your installation can have. The catalog level is assigned the value 3 when the MCP is shipped to your installation, which means that the catalog can keep track of four generations (including the resident generation). You can recompile the MCP with CATALOGLEVELSET equal to a value in the range 1 through 7. You can use the WM (What MCP) system command to find out what the catalog level is at your installation.

If the maximum number of generations has been reached and a new copy is added, the generation with the worst genealogy is deleted from the catalog. The term *worst* can be defined as the oldest timestamp with the lowest version number of the lowest cycle number. The system ARCHIVING function allows you to save information about older generations after they are deleted from the catalog.

If the USECATALOG file attribute is TRUE when you want to access an existing disk file (a file for which the NEWFILE file attribute is equal to FALSE), the system locates the file through the catalog. By default, the system accesses the generation with the best genealogy unless you specify a particular CYCLE and VERSION or use the GENERATION file attribute. GENERATION allows you to access a particular generation by specifying an integer value that designates that generation. The value of GENERATION can range from 0 for the generation with the best genealogy to the value of CATALOGLEVELSET minus 1 for the generation with the worst genealogy. The default value of GENERATION is 0. GENERATION is ignored if USECATALOG is FALSE when the file is accessed.

When a GENERATION Value Is Not Designated

CYCLE and VERSION function in the following manner on old files (files for which the NEWFILE file attribute is equal to FALSE) if USECATALOG is TRUE and if GENERATION has not been designated or has been assigned the value 0.

Example 1

If you designate CYCLE and VERSION when you want to access the file, the system locates the generation that has that exact CYCLE and VERSION.

If you do not assign a value to VERSION, the system locates the generation with that CYCLE and the VERSION equal to 0.

The following situations can occur when the system tries to locate the proper generation:

- If there is no entry in the catalog for a file with that file name, the system displays a "NO FILE" message on the ODT.
- If there is more than one generation of the file with that CYCLE and VERSION, the system chooses the generation with the most recent timestamp. If that generation is resident, the system accesses the file. If that generation is not resident, the system displays a "NO FILE" message on the ODT. This message also displays the serial number of the backup media on which that generation is stored.

Example 2

If you do not designate CYCLE and VERSION when you want to access a file, and you either do not specify GENERATION or assign GENERATION a value of 0, the system locates the generation that has the highest CYCLE, the highest VERSION within that CYCLE, and the most recent timestamp within that CYCLE and VERSION. The following situations can occur when the system tries to locate the proper generation:

- If there is no entry in the catalog for a file with that file name, the system displays a "NO FILE" message on the ODT.
- If there is an entry in the catalog for the file and the generation with the best genealogy is resident, the system accesses the file.
- If there is an entry in the catalog for the file but the generation with the best genealogy is not resident, the system displays a "NO FILE" message on the ODT along with a list of where backup copies of the generation are stored.

When You Designate a GENERATION Value

CYCLE and VERSION function in the following manner on old files (files for which the NEWFILE file attribute is equal to FALSE) if USECATALOG is TRUE, and you assign a value greater than 0 to the GENERATION file attribute.

Example 1

If you designate CYCLE and VERSION when you want to access the file, the system first locates the generations that have that exact CYCLE and VERSION.

If you do not assign a value to VERSION, the system locates the generations with that CYCLE and the VERSION equal to 0. The system then compares the timestamps of the generations within the selected CYCLE and VERSION and chooses the generation based on the value of GENERATION.

For example, if you assign GENERATION the value of 0, the system selects the most recent timestamp within the appropriate CYCLE and VERSION. If you assign GENERATION the value of 1, the system selects the next most recent timestamp within the appropriate CYCLE and VERSION, and so on.

The following situations can occur when the system tries to locate the proper generation:

- If there is no entry in the catalog for a file with that file name, the system displays a "NO FILE" message on the ODT.
- If the system locates generations with the correct CYCLE and VERSION, but the generation designated by the GENERATION file attribute is missing from the catalog, the system displays an "UNMATCHED GENEALOGY" message on the ODT. The designated generation is missing if GENERATION is assigned a value greater than or equal to the number of generations within that CYCLE and VERSION. For example, if there are three generations within that CYCLE and VERSION (generation 0, generation 1, and generation 2) and you assign GENERATION the value of 3, the system cannot find the correct generation.
- If the system locates the correct generation based on CYCLE, VERSION, and GENERATION, but the designated generation is not resident, the system displays a "NO FILE" message on the ODT. This message contains the serial number of the backup media on which that generation is stored.

Example 2

If you do not designate CYCLE and VERSION when you want to access a file but do assign USECATALOG the value TRUE and assign a value greater than 0 to GENERATION, the system ignores the values of CYCLE and VERSION and uses only the timestamps to rank the generations. The most recent timestamp becomes generation 0, the next most recent timestamp becomes generation 1, and so on. The system then chooses the generation specified by the value of the GENERATION file attribute.

The following situations can occur when the system tries to locate the proper generation:

- If there is not an entry in the catalog for a file with that file name, the system displays a "NO FILE" message on the ODT.
- If the designated generation is not resident, the system displays a "NO FILE" message on the ODT. This message contains the serial number of the backup media on which that generation is stored.
- If there are not enough generations of the file in the catalog (for instance, if there are only two generations but GENERATION is assigned the value 3), the system displays an "UNMATCHED GENEALOGY" message on the ODT.
- If the designated generation is resident, the system accesses the file.

Examples of File Generation Selection

The following examples illustrate how the system uses CYCLE, VERSION, GENERATION, and the file timestamp to access the appropriate generation of a file named DEPOSITS. The resident generation of DEPOSITS is stored on a disk family named SAVINGS. Each example is based on the same table that lists the available generations of DEPOSITS. In these examples, the catalog level is 6, so that the system can keep track of six generations.

Example 1

Cycle	Version	Timestamp	Resident	Backup Tape
6	0	03/28/85 18:06:05	No	230642
4	1	01/10/85 13:20:48	No	421640
3	2	01/17/85 16:45:00	No	726000
3	2	01/07/85 10:47:53	Yes	839216
3	2	01/07/85 10:00:00	No	587231
2	0	12/27/84 13:30:32	No	032105

Assume that you assign the following file attribute values:

Attribute	Value
CYCLE	3
VERSION	2
USECATALOG	TRUE
GENERATION	Not specified

The system finds three generations with the correct CYCLE and VERSION, so it chooses the one with the most recent timestamp, 01/17/85. That generation is not resident, so the system displays the following message on the ODT:

```
NO FILE DEPOSITS ON SAVINGS (PK), FIND ON (MT) [726000]
```

You can then copy the file in from tape 726000.

Example 2

Cycle	Version	Timestamp	Resident	Backup Tape
6	0	03/28/85 18:06:05	No	230642
4	1	01/10/85 13:20:48	No	421640
3	2	01/17/85 16:45:00	No	726000
3	2	01/07/85 10:47:53	Yes	839216
3	2	01/07/85 10:00:00	No	587231
2	0	12/27/84 13:30:32	No	032105

Assume that you assign the following file attribute values:

Attribute	Value
CYCLE	Not specified
VERSION	Not specified
USECATALOG	TRUE
GENERATION	Not specified

The system locates the generation with the best genealogy (the highest CYCLE and the highest VERSION within that CYCLE): a CYCLE of 6 and a VERSION of 0. That generation is not resident, so the system displays the following message on the ODT:

```
NO FILE DEPOSITS ON SAVINGS (PK), FIND ON (MT) [230642]
```

You can then copy the file in from tape 230642.

Example 3

Cycle	Version	Timestamp	Resident	Backup Tape
6	0	03/28/85 18:06:05	No	230642
4	1	01/10/85 13:20:48	No	421640
3	2	01/17/85 16:45:00	No	726000
3	2	01/07/85 10:47:53	Yes	839216
3	2	01/07/85 10:00:00	No	587231
2	0	12/27/84 13:30:32	No	032105

Assume that you assign the following file attribute values:

Attribute	Value
CYCLE	Not specified
VERSION	Not specified
USECATALOG	FALSE
GENERATION	Not specified

USECATALOG is FALSE, so the system does not examine the catalog. The system automatically accesses the resident generation, which has a CYCLE of 3, a VERSION of 2, and a timestamp of 01/07/85 10:47:53.

Example 4

Cycle	Version	Timestamp	Resident	Backup Tape
6	0	03/28/85 18:06:05	No	230642
4	1	01/10/85 13:20:48	No	421640
3	2	01/17/85 16:45:00	No	726000
3	2	01/07/85 10:47:53	Yes	839216
3	2	01/07/85 10:00:00	No	587231
2	0	12/27/84 13:30:32	No	032105

Assume that you assign the following file attribute values:

Attribute	Value
CYCLE	3
VERSION	2
USECATALOG	TRUE
GENERATION	1

The system finds three generations with the correct CYCLE and VERSION and uses the timestamps to rank them. The most recent timestamp (01/17/85) is ranked as generation 0. The next most recent timestamp (01/07/85 10:47:53) is ranked as generation 1. The earliest timestamp (01/07/85 10:00:00) is ranked as generation 2. The generation specified by the GENERATION file attribute, 1, is resident and the system accesses it.

Example 5

Cycle	Version	Timestamp	Resident	Backup Tape
6	0	03/28/85 18:06:05	No	230642
4	1	01/10/85 13:20:48	No	421640
3	2	01/17/85 16:45:00	No	726000
3	2	01/07/85 10:47:53	Yes	839216
3	2	01/07/85 10:00:00	No	587231
2	0	12/27/84 13:30:32	No	032105

Assume that you assign the following file attribute values:

Attribute	Value
CYCLE	3
VERSION	2
USECATALOG	TRUE
GENERATION	3

The system finds three generations with the correct CYCLE and VERSION and uses the timestamps to rank them. The most recent timestamp (01/17/85) is ranked as generation 0. The next most recent timestamp (01/07/85 10:47:53) is ranked as generation 1. The earliest timestamp (01/07/85 10:00:00) is ranked as generation 2. The generation specified by the GENERATION file attribute is 3, however, and generation 3 does not exist. As a result, the system displays the following message on the ODT:

```
UNMATCHED GENEALOGY
```

Example 6

Cycle	Version	Timestamp	Resident	Backup Tape
6	0	03/28/85 18:06:05	No	230642
4	1	01/10/85 13:20:48	No	421640
3	2	01/17/85 16:45:00	No	726000
3	2	01/07/85 10:47:53	Yes	839216
3	2	01/07/85 10:00:00	No	587231
2	0	12/27/84 13:30:32	No	032105

Assume that you assign the following file attribute values:

Attribute	Value
CYCLE	Not specified
VERSION	Not specified
USECATALOG	TRUE
GENERATION	4

The system ignores the CYCLE and VERSION values of the generations and uses only the timestamps to determine the genealogy. The system then ranks the generations in the following order:

Generation	Cycle	Version	Timestamp
0	6	0	03/28/85 18:06:05
1	3	2	01/17/85 16:45:00
2	4	1	01/10/85 13:20:48
3	3	2	01/07/85 10:47:53
4	3	2	01/07/85 10:00:00
5	2	0	12/27/84 13:30:32

The generation specified by the GENERATION file attribute, 4, is not resident, so the system displays the following message on the ODT:

```
NO FILE DEPOSITS ON SAVINGS (PK), FIND ON (MT) [587231]
```

You can then copy the file in from tape 587231.

Impact of Cataloging on System Performance

Using cataloging to automate the locating of backup copies of files does involve a tradeoff because system performance can be reduced. Each time you open or close a disk file on a noncataloging system, two to three disk I/O operations must occur. One to two additional I/O operations are required to open or close a disk file on a cataloging system. Also, rebuilds take substantially longer on a cataloging system than on a noncataloging system because both a catalog rebuild and a family rebuild must be done.

Another tradeoff involved in cataloging is that the file SYSTEM/CATALOG is usually large. At least one sector and sometimes several sectors are needed to store the catalog information for each disk file on the system. Unisys recommends that the SYSTEM/CATALOG file be stored on a family with files that are not accessed frequently. Frequent I/O operations to the family containing SYSTEM/CATALOG slow the handling of the catalog and thus the performance of the entire system. Use the DL (Disk Location) system command to designate the family on which you want to store SYSTEM/CATALOG.

Setting Up a Cataloging System

To set up a cataloging system for the first time, perform the following steps. Do not use this procedure if you are already running on a cataloging system, because the current catalog becomes inaccessible as a result. The MCP must be compiled with CATALOGINGLEVEL > 0. By default, the MCP is compiled with CATALOGINGLEVEL = 3.

1. Use the DL (Disk Location) system command to designate a disk family as the catalog family.
2. Use the OP + CATALOGING version of the OP (Options) system command to enable the system option CATALOGING.
3. If you want files to be cataloged by default, use the OP + USECATDEFAULT version of the OP (Options) system command to assign TRUE as the default value of the USECATALOG file attribute.
4. Halt/load the system. The system then displays the following message on the ODT:

   ```
   OK TO CREATE NEW CATALOG
   ```

5. Enter the reply *OK*. The system then creates the new SYSTEM/CATALOG file on the catalog family.
6. Use the WFL statement VOLUME ADD to enter into the volume library each disk and tape on which you want to store cataloged files. It is suggested that all disks and tapes used on your system be added to the volume library.
7. Use the WFL statement CATALOG ADD to enter existing files into the catalog.

Operating a Cataloging System

This subsection explains the following procedures that you can use to operate a cataloging system:

- Entering files into the catalog
- Making backup copies of cataloged files
- Accessing cataloged files
- Removing catalog entries
- Purging catalog backup tapes

Entering Files into the Catalog

After cataloging is set up on your system, you can enter a file into the catalog by performing one of the following actions. You must store the file on a volumed disk.

- Assign the value TRUE to the USECATALOG file attribute before creating the file. The file is cataloged automatically when the file is made permanent.
- Operate the system with the system option USECATDEFAULT enabled. This makes the default value of the USECATALOG file attribute TRUE and all permanent files are entered into the catalog when they are created unless their USECATALOG file attribute is assigned the value FALSE.
- Use the WFL statement CATALOG ADD to mark a permanent file as *cataloged*.
- Use the WFL statement COPY & CATALOG or ADD & CATALOG to copy the file to disk from backup media and enter the file into the catalog. The copied version is marked in the catalog as the resident version of the file with the source version listed in the catalog as a backup copy.
- Use the SM (Send to MCS or Database) system command to turn on the CANDE option CATDEFAULT. Enter the following when the CANDE MCS is running:

  ```
  <mix #> SM OP + CATDEFAULT
  ```

  When CANDE workfiles are saved, the CANDE option CATDEFAULT governs whether or not those files become cataloged files.

Making Backup Copies of Cataloged Files

After a file is cataloged, you can use the WFL statement COPY & BACKUP to make backup copies of the file. COPY & BACKUP performs two functions: it copies the file onto backup media, and stores in the catalog the volume name and serial number of the media the backup was copied to. Both the source and destination volumes in the COPY & BACKUP statement must be volumed media.

You can save up to two backup copies of each generation of a file. If you use COPY & BACKUP to store a third copy of a generation on another volume, the catalog keeps

track of only the two most recent copies. The system does not remove the earliest copy, but you cannot access it through the catalog.

When you copy a cataloged disk file with the WFL statements *ARCHIVE FULL*, *ARCHIVE INCREMENTAL*, *ARCHIVE DIFFERENTIAL*, or *ARCHIVE ROLLOUT*, the system automatically places a backup reference in the catalog to the library maintenance backup tape that is generated by the archive process.

Accessing Cataloged Files

The catalog can keep track of up to seven generations of each cataloged file, depending on the catalog level. You can choose which generation of the file the system accesses. The system uses the catalog to access the file if the value of the USECATALOG file attribute is TRUE or if the USECATDEFAULT system option is enabled so that the default value of USECATALOG is TRUE. USECATALOG takes precedence over USECATDEFAULT. If USECATALOG is assigned the value TRUE, but the default value of USECATALOG is FALSE because USECATDEFAULT is not enabled, then USECATALOG still has the value TRUE. If USECATALOG is assigned the value FALSE, but the default value of USECATALOG is TRUE because USECATDEFAULT is enabled, then the value of USECATALOG is still FALSE.

If USECATALOG is TRUE, the system accesses the generation with the best genealogy unless you request another generation by using the CYCLE, VERSION, or GENERATION file attributes. Again, the generation with the best genealogy is the one with the highest CYCLE, the highest VERSION within that CYCLE, and the most recent timestamp.

USECATALOG must be TRUE in order for the system to automatically access the generation with the best genealogy. If USECATALOG is FALSE, the system accesses the resident version of the file by default. If the generation with the best genealogy has been removed and another generation has been copied to disk, and thus is the resident version, the system might be using an outdated version of the file.

The WFL statements REMOVE, CHANGE, ADD, SECURITY, and CATALOG ADD always act on the resident generation of the file, regardless of whether or not it has the best genealogy and regardless of whether or not USECATDEFAULT is enabled.

When you use the WFL statement CATALOG DELETE, you can specify a particular generation by assigning CYCLE, VERSION, or GENERATION. If you do not specify a particular generation, the CATALOG DELETE operation acts on the generation with the best genealogy. The WFL statement CATALOG PURGE acts on all the generations of the file. Refer to "Removing Catalog Entries" in this section for more information about the CATALOG DELETE and CATALOG PURGE statements.

The WFL statement RUN acts only on the generation with the best genealogy if USECATDEFAULT is enabled. If USECATDEFAULT is not enabled, RUN acts on the resident generation.

If you use the RUN statement and the generation with the best genealogy is not resident, the system displays a "NO FILE" message on the ODT. This message also

displays the serial numbers of the media on which backup copies of the file are stored. An example of this message is as follows:

```
NO FILE DEPOSITS ON SAVINGS (PK), FIND ON (MT) [347681]
```

DEPOSITS is the file name, SAVINGS is the family name, (PK) indicates that SAVINGS is a disk pack, (MT) indicates that the backup copy is on a magnetic tape, and 347681 is the serial number of the tape containing the backup copy.

The system command CS (Change Supervisor) acts only on the generation with the best genealogy if USECATDEFAULT is enabled.

The system commands CM (Change MCP), CP (Control Program), MC (Make Compiler, PP (Privileged Program), SI (System Intrinsics), and SL (Support Library) always act on the resident generation, regardless of whether or not it has the best genealogy and regardless of whether or not USECATDEFAULT is enabled.

To examine backup file information such as the available generations, use the PD (Print Directory) system command. The display generated by the PD command refers to the generations as entries. These entries range from 1 for the generation with the best genealogy to the value of CATALOGLEVELSET for the generation with the worst genealogy. Note that this numbering differs by 1 from that used in the GENERATION file attribute. If you use the PD command to examine the resident version and it is not the generation with the best genealogy, the system omits some of the PD display. You can also use the FILEDATA utility to display information about the attributes of a file and the backup copies of that file.

After a file has been entered into the catalog and has been backed up, it remains in the catalog even if the resident version is removed. You might want to remove the resident version because you need to make disk space available, because the resident version is damaged, or because you want to create a new version of the file.

If you try to open the removed file, the system displays a "NO FILE" message on the ODT that indicates the serial numbers of the media on which backup copies of the file are stored. You can then locate a backup file and copy it back in with the WFL *COPY* statement.

Removing Catalog Entries

There are two ways to remove information about a file from the catalog. The WFL statement CATALOG DELETE lets you remove references to a particular generation, including the one that is resident. The WFL statement CATALOG PURGE removes all the backup information for a file. The CATALOG DELETE and CATALOG PURGE statements delete only catalog entries; the resident and backup files are still available but cannot be accessed through the catalog. If the copy specified in the CATALOG DELETE statement is the resident version, the copy still remains the resident version after its catalog entry has been deleted.

Purging Catalog Backup Tapes

After you have used the cataloging system for awhile, you might discover that you have backup copies on tape that are no longer needed. You can purge these backup tapes, rename them, and use them for other purposes, and the system automatically changes the name of the tape in the volume library to SCRATCH. However, the catalog entries for the files that were backed up on the tape still indicate that backup copies are stored on that tape and refer to its old name.

Rebuilding Catalogs

When a cataloging system needs to perform a family rebuild to recover from directory errors, it performs a family rebuild and then a catalog rebuild. A catalog rebuild is a lengthy process.

During a family rebuild on a cataloging system, the system reads the flat directory to determine which files are resident and inserts the names of the resident files into the catalog and the FAST.

During a catalog rebuild, the system reads the catalog to extract all the backup information for each cataloged file and enters the names of files that have backup copies into the file access structure table (FAST).

For more information about the FAST and family rebuilds, refer to "Family Rebuilds" in Section 1.

Creating and Using Backup Copies of the Catalog

You might want to make backup copies of the catalog itself from time to time to provide backups in case any catalog data on disk becomes corrupted or the catalog family fails. How often you should copy the SYSTEM/CATALOG file depends on the needs of your installation. When you use a backup catalog, the information is not available that has been added to the catalog since that backup copy was made. If the catalog entries are updated frequently and recovering the cataloging information is critical on your system, you will want to back up the catalog more often.

The system can also have duplicate online copies of the catalog. Refer to "Duplicating Catalog Directories" in Section 8 for more information about this subject.

You can copy the active catalog (SYSTEM/CATALOG/nnnn or SYSTEM/ACCESS/nnnn) to a backup tape by using the following procedure:

1. Issue the system command COPYCAT to create an inactive copy of the catalog. The correct syntax is as follows:

```
COPYCAT <dummyname> ON <family name> <family index number>
```

The variables <dummyname> and <family name> are designated so that the system can build a disk file that does not conflict with the active catalog. The variable <family index number> is the family index of the disk volume to receive the inactive copy.

2. Use library maintenance to copy the inactive catalog file to a backup tape. At cataloging installations, the COPY command has the following syntax:

```
COPY & COMPARE <dummyname>
         AS SYSTEM/CATALOG/<family index number>
           FROM <family name> (PACK) TO <tape name>
```

At noncataloging installations, the COPY command has the following syntax:

```
COPY & COMPARE <dummyname>
         AS SYSTEM/ACCESS/<family index number>
           FROM <family name> (PACK) TO <tape name>
```

3. After the copy to tape has completed, you can remove the dummyname copy of the catalog from the disk.

When you restore the catalog from a backup copy, some catalog information is not up-to-date. The names and serial numbers of backup media are not available if the backup media have been made since the last time the catalog was backed up. As a result, the backup copies of files that were made after the catalog was copied to tape cannot be accessed through the catalog. Also, the names and other information are not available for volumed tapes that have been changed since the catalog was backed up.

When you need to replace the current catalog, you can do one of the following operations:

- If the catalog family is usable, you can replace the current catalog so that the backup catalog is on the same family and is thus easy for you to locate. However, replacing the current catalog requires extra steps, because the current catalog is marked as a nonremovable system file.
- If you cannot restore the catalog family, you must designate a new catalog family and copy the backup catalog to that family.

Replacing the Current Catalog

To replace the current catalog, perform the following steps:

1. Use the OP – CATALOGING version of the OP (Options) system command to disable the CATALOGING system option.
2. Halt/load the system.
3. Use the WFL *REMOVE* statement to delete the catalog. The REMOVE statement syntax is as follows:

```
REMOVE SYSTEM/CATALOG/= FROM <family name>
```

The variable <family name> is the name of the catalog family.

4. Use the WFL *COPY* statement to copy the backup catalog from tape. The COPY statement syntax is as follows:

```
COPY SYSTEM/CATALOG/<family index number> FROM <tape name>
TO <family name> (PACK)
```

   The variable <family index number> is the family index number of the disk that the new catalog is to be stored on and the variable <family name> is the name of the catalog family.

5. Use the OP+CATALOGING version of the OP (Options) system command to enable the CATALOGING system option.
6. Halt/load the system. The system then performs a catalog rebuild, followed by a family rebuild for all the families on the system. These rebuilds can be lengthy processes and depend on the number of files on disks at the installation.

Designating a New Catalog Family

To designate a new catalog family and store the backup catalog file there, perform the following steps:

1. Use the WFL *COPY* statement to copy the backup catalog from tape to a family that is not the current catalog family. The syntax is as follows:

```
COPY SYSTEM/CATALOG/<family index> FROM <tape name> TO <family name>
(<family index>)
```

   The variable <family index> is the three digit family index number of the disk that the catalog is to be stored on and the variable <family name> is the name of the family that is to become the new catalog family.

2. Use the DL (Disk Location) system command to designate this family as the catalog family.
3. Halt/load the system. The system now uses the new catalog on the restored family.

Replacing a Damaged Volumed Disk

A base pack for a volumed family might become damaged so that none of the files on the family can be accessed. To preserve the backup information for the files that were on the original family, you must substitute another base pack for the damaged one or repair and reuse the damaged disk.

To substitute a new base pack for the damaged one so that the new family inherits the backup information, perform the following steps:

1. Use the RC (Reconfigure Disk) system command to reconfigure the new disk with the same family name and serial number as the damaged disk. When you use the original name and serial number, the new family is connected automatically to the old system archive directory for the family. When you issue the RC command, the system displays the following message on the ODT:

   ```
   PK<unit number> OK TO RE-ENTER INTO VOLUME LIBRARY
   ```

2. Enter the reply *OK*. The system then performs a catalog rebuild. When the catalog rebuild is finished, the disk is ready to have files stored on it. Whenever a program tries to access a file that was stored on the original disk, the system displays a "NO FILE" message on the ODT. This message lists the volume name and serial number of the media where backup copies are stored.

If the damaged disk has been repaired, you must perform the following steps to re-create the family so that it inherits the backup information:

1. Use the RC (Reconfigure Disk) system command to reconfigure the disk with exactly the same family name and serial number as that of the original disk. The system then displays the following message on the ODT:

   ```
   PK<unit number> VOLUMED DISK BEING CHANGED BY RC
   ```

2. Bring the catalog family and any other desired disks online. The system then displays the message on the ODT:

   ```
   PK<unit number> OK TO RE-ENTER INTO VOLUME LIBRARY
   ```

3. Enter the reply *OK*. The system then performs a catalog rebuild. When the catalog rebuild is finished, the disk is ready to have files stored on it. Whenever a program tries to access a file that was stored on the original disk, the system displays a "NO FILES" message on the ODT. This message lists the volume name and serial number of the media where backup copies are stored.
4. To restore the old disk files to the disk, copy the desired files from backup tapes.

Section 6
Comparing the Archive and Catalog Subsystems

This section provides a general comparison of the differences and similarities that exist between the archive and the catalog subsystems. If you need specific information on the overall usage of these systems, refer to Sections 4 and 5.

Availability and Compatibility Issues

As an A Series user, you always have access to the archive subsystem. Use of this subsystem does not require special system option settings. Nevertheless, some of the archive subsystem statements require a usercode with privileged access. In contrast, the catalog subsystem is optionally available on A Series systems. To use this subsystem, your installation must enable the CATALOGING system option by issuing an OP (OPtions) system command that includes a + *CATALOGING* specification. The OP system command is described in the *A Series System Commands Operations Reference Manual.*

Although the archive subsystem and the catalog subsystem operate independently of each other, some operations occur in parallel if your installation both catalogs and archives files. For example, when you execute an archive backup or rollout operation on a cataloged disk file, the system records the backup entry in the catalog as if you had executed a WFL *COPY & BACKUP* statement through the catalog subsystem. In a similar manner, if you issue an archive statement that either opens or refers to an existing disk file and the USERCATALOG file attribute is set to TRUE, the catalog subsystem controls the generation of the file the archive subsystem uses. File generation is controlled in this manner even if an archive record for a usable version of the file exists in the archive directory. More information on file references and file OPEN requests is provided under "Effects of Nonresident Files on OPEN Requests and File References" later in this section.

File Management

The archive subsystem enables you to create backup copies of resident disk files, and to manage the resulting nonresident backup copies. The catalog subsystem performs similar functions in that it also enables you to back up and manage disk files. In addition, the catalog subsystem enables you to manage tape files. The only management of nonresident files you can perform through the archive subsystem occurs through the *ARCHIVE MERGE* WFL statement, which merges archived files from two or more backup tapes to a single tape or tape set.

Only one generation of a file can reside on disk at any time. The archive subsystem can track up to four backup copies of one generation of a disk file. The cataloging

subsystem keeps track of up to two backup copies of several generations of a disk or tape file as determined by the file attributes CYCLE and VERSION; the total number of generations your catalog subsystem can maintain is determined by the CATALOGLEVELSET option (an MCP compile-time option). The mechanics of how generations are assigned to files and used in the catalog subsystem are described in greater detail in Section 5 of this guide.

If you use the archive subsystem, it is important to notice that the resident and nonresident generations of archive files are not always the same. For example, different generations can result when you change a disk file without either rolling out or backing up the changed file to tape. If you then reload the unchanged, nonresident generation of the file through an archive restore process, the system replaces the changed disk file with the unchanged, nonresident generation.

Effects of Nonresident Files on OPEN Requests and File References

The archive and the catalog subsystems recognize a disk file as *nonresident* when both of the following conditions are met:

- The disk file header of the file is not stored in the flat directory of its disk family.
- The archive directory or the catalog directory contains references to backup copies of the file.

Archive and catalog backup records are significant because they can affect the way in which the respective subsystems search for a requested file. For example, when disk family substitution occurs, a search for a disk file can be repeated with and without the usercode of the initiating process, and on the primary and alternate disk families. In these cases, the subsystem can terminate a file search if it does not find a resident generation of a file, or if it finds a record that refers to a nonresident generation.

Searches for disk files can also be terminated when the system is processing ordinary file OPEN requests and code-file RUN requests, PROCESS requests, or library invocations.

In general, the archive subsystem terminates its disk-file searches whenever it locates archive backup records that refer to the requested files. In the catalog subsystem, shortened searches result if the USECATALOG file attribute is set to TRUE during a file OPEN request, or if the *OP + USECATALOGDEFAULT* system option is enabled during a code-file invocation and the catalog subsystem finds a backup record for the requested file in the catalog directory.

Note: *The presence of archive or catalog backup records has no effect on the WFL requests CHANGE and REMOVE, or the library maintenance requests COPY and ADD. These requests operate only on resident generations of files.*

Responses to the NO FILE System Message

When a disk file search results in a "NO FILE" condition, but a backup copy of the requested file exists, the archive or the catalog subsystem displays information describing the location of the backup file. The subsystems retrieve this information from their respective backup directories.

If an archive record for the requested file exists, and your installation has enabled the AUTORESTORE system option, the SYSTEM/AUTORESTORE process automatically reloads the missing file to disk if the initiating task has the same usercode as the file.

Displays of Backup Information and Reports

Through the PD (Print Directory) system command or the FILEDATA system utility, you can display and generate reports of archive or catalog file information. These system commands are described more fully in the *A Series System Commands Operations Reference Manual.* The GETSTATUS and SETSTATUS system intrinsics that the PD command and the FILEDATA utility use are described fully in the *A Series GETSTATUS/SETSTATUS Programming Reference Manual.*

About Volumed Tapes and Disks

The catalog subsystem supports a *volume library*, which is the part of the SYSTEM/CATALOG directory that tracks all volumed tapes and disks used by the system. Tapes and disks for which descriptive entries have been added to the volume library though the WFL *VOLUME ADD* statement are said to be *volumed.* All cataloged files, including backup copies of cataloged files, can reside only on volumed tapes and disks.

The archive subsystem does not provide a volume library facility, and does not require volumed disks and tapes for its operations. However, if your installation uses the catalog volume library or the volume directory, the archive subsystem uses the volume information in its operations. Specifically, the archive subsystem refers to the volume library or directory to determine whether a backup tape has been overwritten or purged. If this subsystem detects an altered tape, it updates the affected backup records in the archive directory.

Detailed information on volume libraries and directories is provided in Section 3 of this guide.

Section 7
Planning and Installation

Performance of your system depends on the efficient operation of certain critical system files. The best system performance is achieved if these files are allocated to disk families so that the accesses of the files do not interfere with each other. This section suggests methods that your installation can use to make efficient use of the disk subsystem. Disk file allocation is a complex subject, and this section does not attempt to make flat recommendations about where system files should be placed. The needs of each system are different, and this section is designed to suggest approaches you might want to consider when tuning and balancing the operation of your system.

This section discusses the following topics:

- The characteristics and requirements of the various system files
- Allocation of system files to improve system performance and to ease recovery from certain errors
- Starting up the system
- The characteristics and use of disk file headers

System File Requirements

The MCP, disk access structure, flat directories, archive directories, system support libraries, system utility programs, and other system software components are stored as disk files. When this guide refers to system files, it means these system software components.

The following information discusses the characteristics of selected system files.

MCP Code File

The current MCP code file (the one the system is running on) is accessed frequently and all accesses are time-critical. You can use the WM (What MCP) system command to determine which MCP code file the system is running on. Most accesses of the MCP code file are caused by presence-bit interrupts, which in this case notify the MCP that an MCP code file sector is needed that is not in main memory. The system then reads the code file sector into main memory from disk. The MCP cannot predict which code file sectors are going to be needed to be read into main memory, so it cannot read them into a *look-ahead buffer* ahead of time. The unit on which the MCP code file is stored is called the halt/load unit or the boot unit. Corruption of the MCP code file might require a halt/load. You can restore the file by recopying the original file and then using the CM (Change MCP) system command to designate the new code file as the halt/load MCP. The MCP code file is not updated during system operation.

System Library and Intrinsic Code Files

These files include system libraries that are linked to function names established through the SL (Support Library) system command and system intrinsics established with the SI (System Intrinsics) system command. The characteristics of the files are similar to those of the MCP code file. Their failure usually does not require a halt/load but can cause jobs using them to be discontinued.

Other System Code Files

These files, such as system compilers, have characteristics similar to those of the MCP code file. Their failure usually does not require a halt/load, but can cause jobs using them to be discontinued. It is suggested that these files be grouped onto a single family so that they can be conveniently referenced with the family substitution operation. Refer to "Family Substitution" in this section for more information on this subject.

Overlay Files

Data is stored in a temporary overlay file when it is overlaid in main memory. These files do not have headers in the flat directory. These files are accessed frequently, especially if main memory is full or if the overlay goal (the MCP memory management factor OLAYGOAL) is assigned a value greater than 0. All overlay file I/O operations are time-critical, and contention for the family where these files are stored can seriously affect system performance. Failure of the family where overlay files are stored can cause jobs to be discontinued and can require a system halt/load. It is not necessary or possible to save or restore the contents of overlay files.

JOBDESC and Job Files

The JOBDESC file contains information about job files. Job files are code files that are compiled by the WFL compiler. Job files do not have headers in the flat directory. These files are accessed regularly, and all JOBDESC and job file I/O operations are time-critical. Failure of these files can require a halt/load. You cannot recover the contents of these files, but you can re-create them by rerunning jobs. If persistent I/O errors occur for the JOBDESC file or the job file, you might have to perform one of the following two procedures:

- Enter the primitive system command *??RJ* (Remove JOBDESC File) and then halt/load the system to build a new JOBDESC file.
- Use the DL (Disk Location) system command to create a new JOBDESC file on another family.

Failure of the JOBDESC file results in the loss of the input job queues, information about the queues, and job-log output for jobs that are waiting to be printed. The JOBDESC file also contains certain system operational characteristics such as ADM (Automatic Display Mode) settings and TERM (Terminal) settings. This information is lost if the JOBDESC file fails or is moved to another family.

Printer and Punch Backup Files

These files are accessed frequently, but I/O operations are buffered and thus are not time-critical. However, if these files are on the same disk with other system files, the I/O operations of printer and punch backup files can interfere with the time-critical I/O operations of other system files. The I/O operations of JOBDESC and job files are particularly time-critical. Failure of the printer or punch backup files leads to the loss of some output, and some jobs might be discontinued. You cannot recover the data in these files, but you can rerun the jobs again to produce output.

SYSTEM/SUMLOG File

The SYSTEM/SUMLOG file contains the log of many system activities and is accessed regularly. Its failure usually does not require a halt/load. You cannot recover the data in the log, which can pose a problem if your installation uses the log to provide information for billing purposes. If you use the log for billing, you should make backup copies as a precaution. If there are persistent log file errors, use the TL (Transfer Log) system command to save the log file and create a new one.

Disk Access Structure File

The disk access structure file is named SYSTEM/ACCESS/<family index number> on noncataloging systems and SYSTEM/CATALOG/<family index number> on cataloging systems.

Disk Space Requirements

The SYSTEM/ACCESS directory requires approximately 2,400 disk sectors plus 0.1 disk sector for every permanent disk file at the installation. For example, if the installation has three disk families and each has 2,000 files, the SYSTEM/ACCESS directory uses about 3,000 disk sectors. At cataloging installations, the SYSTEM/CATALOG directory requires approximately 3,600 disk sectors plus 1.1 disk sectors for every permanent disk file, for every cataloged disk file that has been backed up, and for every cataloged tape at the installation. For example, if the installation has three disk families and each has 2,000 files, the SYSTEM/CATALOG directory uses about 10,200 sectors. In addition, the SYSTEM/CATALOG directory needs several words for every disk and tape volume that has been added with the WFL *VOLUME ADD* statement. Moreover, if the installation runs with SECOPT TAPECHECK=AUTOMATIC, the space requirements for the SYSTEM/ACCESS or SYSTEM/CATALOG directory are expanded by approximately 1.1 disk sectors for each tape volume that has been added with the WFL *VOLUME ADD* statement.

Restoration

The disk access structure file is referenced regularly. Its failure can cause jobs to fail, can cause family and catalog rebuilds, or can require a halt/load. It is not necessary to restore the disk access structure file on noncataloging systems that are not using the tape security subsystem. If you re-create a disk access structure with a family rebuild,

no data is lost. If you are using the tape security subsystem (system command SECOPT TAPECHECK=AUTOMATIC), restoration is necessary on noncataloging systems. You can write a program to make backup copies of the volume directory by using GETSTATUS and SETSTATUS calls. For more information, refer to the *A Series GETSTATUS/SETSTATUS Programming Reference Manual*. On a cataloging system, restoration of the disk access structure file is difficult and usually involves the loss of some catalog backup information. Refer to "Duplicating Catalog Directories" in Section 8 for more information on the subject.

Archive Directories

Archive directory files names have the following form:

```
SYSTEM/ARCHIVE/<family name>/<family index>
```

The variable <family name> indicates the family on which the archive directory resides. The system stores and references archive directories on the same family that it stores the directory SYSTEM/ACCESS or SYSTEM/CATALOG.

The variable <family index> indicates the member of the catalog family on which the archive directory resides. Archive directories are accessed by archive backup and restore statements such as *ARCHIVE FULL* and *ARCHIVE RESTORE* and by the system PD (Print Directory) command. An archive directory needs about two disk sectors for each backup file to be tracked. For example, if a disk family contains 5,000 disk files, and a WFL *ARCHIVE FULL* statement is executed for that family, the archive directory on the catalog family needs approximately 10,000 disk sectors. Archive directory failures can cause the archive subsystem to lose track of the backup tape locations of files previously processed by WFL *ARCHIVE* statements. Refer to "Duplicating Archive Directories" in Section 8 for more information on this subject.

SYSTEM/USERDATAFILE File

This file contains usercodes and passwords for system users and is small. Most of the I/O operations for this file are not time-critical. A copy of this file should be kept for backup, although you should take security precautions with the copy because of the sensitive nature of the information it contains.

Sort Files

These are temporary files used by the sort intrinsic. The I/O operations of sort files do not affect the performance of the MCP, but the I/O operations do affect the performance of programs that use the sort intrinsic. Also, if the sort files are on the same family as other system files, sorting can interfere with the time-critical I/O operations of these system files. Sorting uses large temporary files, and it is not necessary to save backup copies.

Disk File Allocation

Your installation can control system performance and the ease of recovery from catastrophic errors to a great extent by your choice of which families you place various system files on. The MCP lets you select the location of the following code files (and thus to control what the halt/load family is):

- The intrinsics file
- The system libraries
- The system files controlled by the DL (Disk Location) system command

Your installation can mix some or all system files with files used by application programs or can segregate system files from files used by application programs. These decisions involve tradeoffs, which are discussed in this section.

It is important to remember that disk file allocation is a complex task, and the needs of one system can differ from the needs of another. Suggestions included in this section are not flat recommendations that apply to every system; instead, these suggestions provide approaches you might want to consider when tuning and balancing the operation of your system.

For maximum system performance it would be ideal to store every system file on a different disk and never put any other files on those disks. But this is not practical for two reasons: some of the system files are very small and a large amount of disk space would be wasted, and the system would be vulnerable to the failure of any one of several different disks or disk drives. When you are deciding where to locate disk files, consider the following criteria:

- System performance (speed)
- Amount of storage space needed
- Ease of recovery from damaged files or media
- The probability that the failure of a single disk will cause a service interruption (such as DISK DRIVE NOT READY errors that require a halt/load)

The failure of a system file or the family it resides on usually does not result in the permanent loss of any important data except for USERDATAFILE, the log, and the catalog at an installation that uses cataloging and/or the tape security subsystem. Creating a new file or recopying the original file usually solves the problem caused by the failure of system files other than USERDATAFILE, the log, and the catalog on cataloging systems. Sometimes you must move a system file to another family until the original family is restored. Use the DL system command to move a system file from one family to another.

Most system files do not change in ways that require saving frequent backup copies. Unisys recommends that they be segregated from application data files that do need frequent safety backups. In this way, total failure of a family that contains system files does not entail the lengthy recovery operations that might be required for application data files. Similarly, total failure of a family dedicated to application data files does not harm the operations of the MCP. Thus, recovery actions to restore the application files

can take place while the MCP is fully operational. In other words, it is probably simpler to segregate static system files (and static application files) from dynamic application files by putting them on separate families.

System performance usually improves if you segregate the halt/load family and the overlay files from all other files. MCP code file presence-bit handling and overlay handling sharply affect the performance of the system. The worst possible performance is produced if these time-critical files share families with application programs that perform many I/O operations.

Unisys suggests that the halt/load family be a single disk family with no application program files stored on it. The halt/load family should not be a multidisk family unless you want to have duplicate MCP code files. Refer to "Duplicating MCP Code Files" in Section 8, "Safety Mechanisms," for more information on this subject. System performance usually improves if the halt/load family is reserved for the files that have to be there: the MCP code file and the SYSTEM/TRAINTABLES file, which controls printer operation. Systems that use network support processors (NSPs) must have the file FIRMWARE/NSP stored on the halt/load family, and systems that use line support processors (LSPs) must have the file FIRMWARE/LSP stored on the halt/load family. On A 1, A 2, A 3, A 4, A 5, and A 6 systems, Unisys also suggests that the BOOTCODE file be stored on the halt/load family.

If your installation has a limited number of disk families, you can group as many system files as desired, taking into account the impact on system performance of grouping system files together. For example, you could store the disk access structure, the USERDATAFILE, and the overlay files on the halt/load family. Application files such as seldom-changed code files also could be grouped with the system files.

If your installation has a limited number of disk families, you may have to group all the system files described above on the halt/load family and name that family DISK. If your installation has several disk families, however, you can improve performance by distributing the system files to different families.

Family Substitution

You might find it convenient to group the disk resources of your installation so that each department or other organizational unit using the system has access to its own disk family where its private files are stored. These organizational units, which this section refers to as departments, can still share the disk family on which the system compilers and other system files are stored.

Family substitution provides the mechanism so that files can be conveniently divided up among disk families. Family substitution allows you to designate which private family a particular program, job, or usercode should access for private files and which system family it should access for system files.

You can assign a default family specification in the following ways:

- With the MAKEUSER utility for usercodes
- With task attributes within programs

- In jobs with a WFL statement or the MQ (Make or Modify Queue) system command

  Unisys A Series system software and many application programs are written so that the system searches by default for a designated file on a family named DISK. Placing all files on DISK would be inefficient, so family substitution can be used to redirect files to other families. The family specification can designate both a substitute family name and an alternate family name. The syntax of the family specification statement is as follows:

  ```
  FAMILY <target family name> = <substitute family name>
  OTHERWISE <alternate family name>
  ```

  To use family substitution to distribute files to families other than DISK, use the family specification statement with DISK as the <target family name>, the private family of a department as the <substitute family name>, and the general family that contains system code files as the <alternate family name>.

  For example, the accounting department might use the following family specification statement:

  ```
  FAMILY DISK = ACCOUNTPACK OTHERWISE SYSTEMPACK
  ```

  In this example, the private files of the accounting department are on ACCOUNTPACK and the system files are on SYSTEMPACK.

  When a program needs to access a file that is specified as being on the family DISK, the family specification causes the system to search on the substitute family, ACCOUNTPACK, and then the alternate family, SYSTEMPACK, if the file is not located on ACCOUNTPACK. If the system needs to access an application file for the accounting department, it uses the family specification statement to locate the file on ACCOUNTPACK. If the system needs to access a system file, such as the COBOL74 compiler, to do processing for the accounting department, the system performs the following steps:

  1. Instead of checking on DISK, the system checks ACCOUNTPACK but cannot find the COBOL74 compiler.
  2. The system then checks SYSTEMPACK, which is shared by all the departments, and locates the COBOL74 compiler.

     All new files that are specified as being directed to the family named DISK are stored on ACCOUNTPACK instead.

  On a system with several disk families, Unisys suggests that the system compilers, code files, and general support programs be stored on an alternate family that is shared by all the departments. On a small system that does not have a large number of disk families, all system files could be grouped on the halt/load family. You could call the family DISK. The family specification statement in this case would be the following:

  ```
  FAMILY DISK = <substitute family name> OTHERWISE DISK
  ```

  In this example, each department could have its own <substitute family name>. If a program designates that the system is to check for a file on a family named DISK, the system checks for the file on the department family first, and then searches for the file on DISK if the file is not located on the department family.

Using family substitution does not prevent a program or user from accessing or allocating a file on a particular family other than the substitute family or the alternate family. If a family name is used that is not the target family in the family specification statement, the system places the file on the designated family. For example, assume that the substitute family name is ACCOUNTPACK and you enter the following WFL syntax:

```
RUN AUDIT/UPDATE ON AUDITPACK;FILE YEARTODATE
(TITLE = FIRST/QUARTER ON AUDITPACK)
```

The system locates both AUDIT/UPDATE and FIRST/QUARTER directly on AUDITPACK; ACCOUNTPACK is not accessed. However, if the program AUDIT/UPDATE needs to locate files other than YEARTODATE, and these files are designated in the program as being stored on DISK, family substitution causes the system to search for the files on ACCOUNTPACK.

For more information on the family specification statement, refer to the *A Series Work Flow Language (WFL) Programming Reference Manual* and the discussion of usercodes in the information on MAKEUSER in the *A Series Security Administration Guide.*

Example of Disk File Allocation

The following example shows how your installation might want to allocate system files on a new system that has several disk families. This example uses four disk families for system files: HLPACK (the halt/load family), SYSTEMPACK, JOBPACK, and PACK:

Pack Name	Contents
HLPACK	SYSTEM/TRAINTABLES, SYSTEM/SUMLOG,SYSTEM/USERDATAFILE, MCP code file
SYSTEMPACK	Code files, system libraries, system intrinsics, IPSUPPORT library
JOBPACK	JOBDESC file, job files, SYSTEM/ACCESS or SYSTEM/CATALOG, SYSTEM/ARCHIVE directories, overlay files
PACK	Printer and punch backup files, work files for sort tasks

The following is an example of the procedure you can use to allocate the system files to the disk families designated previously:

1. Use the WFL COPY statement to copy SYSTEM/TRAINTABLES to HLPACK.
2. Use the COPY statement to copy the system libraries, SYSTEM/INTRINSICS, and code files such as SYSTEM/COBOL and SYSTEM/ALGOL to SYSTEMPACK.
3. Use the SL (Support Library) system command so that the system references the following libraries on SYSTEMPACK:

```
SL BASICSUPPORT = SYSTEM/BASICSUPPORT ON SYSTEMPACK

SL BNAV2ENVIRONMENT = SYSTEM/BNAV2/ENVIRONMENT ON SYSTEMPACK

SL BNAV2MANAGERS = SYSTEM/BNAV2/MANAGERS ON SYSTEMPACK
```

```
SL BNAV2TRANSLATION = SYSTEM/BNAV2/TRANSLATION ON SYSTEMPACK

SL COMSSUPPORT = SYSTEM/COMS ON SYSTEMPACK

SL CPXSUPPORT = SYSTEM/CP2000/CP/SUPPORT ON SYSTEMPACK

SL DATACOMSUPPORT = SYSTEM/DATACOMSUPPORT ON SYSTEMPACK

SL DRCSUPPORT = <drcsupport code file> ON SYSTEMPACK

SL DSS = SYSTEM/DSS ON SYSTEMPACK

SL FORTRANSUPPORT = SYSTEM/FORTRANSUPPORT ON SYSTEMPACK

SL GENERALSUPPORT = SYSTEM/GENERALSUPPORT ON SYSTEMPACK

SL HELPSUPPORT = SYSTEM/HELP ON SYSTEMPACK

SL IPSSWITCHBOARD = OBJECT/IPSSWITCHBOARD ON SYSTEMPACK

SL KEYEDIOSUPPORT = SYSTEM/KEYEDIOSUPPORT ON SYSTEMPACK

SL MCPSUPPORT = >>CURRENT MCP<<

SL NCSDBSUPPORT = SYSTEM/NCSDB/LIBRARY ON SYSTEMPACK

SL NETWORKSERVICES = SYSTEM/BNAV1/NETWORKSERVICES ON SYSTEMPACK

SL NPSUPPORT = THRASHER/NPSUPPORT ON SYSTEMPACK

SL PASCALSUPPORT = SYSTEM/PASCALSUPPORT ON SYSTEMPACK

SL PLISUPPORT = SYSTEM/PLISUPPORT ON SYSTEMPACK

SL PRINTSUPPORT = SYSTEM/PRINTSUPPORT ON SYSTEMPACK

SL RPGSUPPORT = SYSTEM/RPGSUPPORT ON SYSTEMPACK

SL SCREENDESIGN = SYSTEM/SCREENDESIGN/MANAGER ON SYSTEMPACK

SL SCREENFORMATS = SYSTEM/SCREENDESIGN/FORMATS ON SYSTEMPACK

SL SCREENSUPPORT = SYSTEM/SCREENDESIGN/INTERFACE ON SYSTEMPACK

SL SDASUPPORT = 37/SYSTEM/SDASUPPORT ON SYSTEMPACK

SL TADSSUPPORT = SYSTEM/TADSSUPPORT ON SYSTEMPACK

SL XREFSUPPORT = SYSTEM/XREFSUPPORT ON SYSTEMPACK
```

4. Use the SI (System Intrinsics) system command so that the system references the intrinsics on SYSTEMPACK:

```
SI SYSTEM/INTRINSICS ON SYSTEMPACK
```

5. Use the DL (Disk Location) system command to place the following system files on the appropriate disk families. When the system creates each file, it does so on the appropriate family.
   - DL BACKUP ON PACK
   - DL CATALOG ON JOBPACK
   - DL IPFILES ON SYSTEMPACK
   - DL JOBS ON JOBPACK
   - DL LOG ON HLPACK
   - DL OVERLAY ON JOBPACK
   - DL SORT ON PACK
   - DL USERDATA ON HLPACK
6. Use the SB (Substitute Backup) system command to specify the substitute backup family for the printer and punch backup files:

```
SB DISK = DLBACKUP

SB PACK = DLBACKUP
```

   If a program designates that the backup files are to be stored on a disk family, the system uses the SB specification to place the backup files on the family designated by the DL command.
7. Use the COPY statement to copy the MCP code file to HLPACK and then use the CM (Change MCP) system command to designate that code file as the halt/load MCP.
8. Use the MAKEUSER utility to establish a default family specification for each usercode so that the system is directed to first look for files on a departmental family and then on SYSTEMPACK. In this example, the departmental family is ACCOUNTPACK.

```
FAMILY DISK = ACCOUNTPACK OTHERWISE SYSTEMPACK
```

System Startup

On nearly all Unisys A Series systems, you use SYSTEM/LOADER the first time that you initialize the system (A 1, A 2, A 3, A 4, A 5, and A 6 systems are delivered to the customer preinitialized). The functions of the LOADER include setting up the halt/load family and copying the MCP to the halt/load family.

After the system is running, the LOADER will probably not be needed again. Instead, you can establish your system configuration with the WFL *COPY* statement and system commands such as DL (Disk Location), SB (Substitute Backup), SI (System Intrinsics), and SL (Support Library). After the system is running satisfactorily, you should create

alternate halt/load families. Refer to "Making Alternate or Standby Halt/Load Families" in Section 8, "Safety Mechanisms," for more information on this subject. It is probably safer and more productive if you use the MCP to recover from catastrophic errors than to depend on the LOADER. You should be more familiar with running the MCP than the LOADER because the MCP provides a wider range of capabilities. The LOADER should be used only as a last resort, such as if all alternate halt/load families fail.

Disk File Headers

This subsection describes disk file headers and briefly provides a history of their evolution through the Mark 3.8 release. The subsection also discusses the concept of the family header version, and provides instructions for converting the version to one that can be used with the current release.

Understanding Disk File Header Versions

Disk file headers contain the attributes necessary to describe the disk pack files and include such information as the file title, the actual location of each area of the file, the maximum and minimum record sizes, the block size, and the title of the security guard file (if any). This information is used mainly by the MCP directory management and logical I/O routines. Disk file headers are relatively permanent data structures that reside primarily in disk pack directory files located on the base units of each disk pack family. Disk file headers are also found on library maintenance tapes.

Prior to Mark 3.6, all headers in directories had a version of either 3 or 4. The version 4 header, introduced in Mark 3.3, makes use of some words and fields that were not used in the version 3 header. The version 4 header has a rigid structure that can create migration problems when attributes are added to it. The Mark 3.6 release introduced a totally new disk file header format that is much more flexible and allows new attributes to be added without creating migration problems. The version number of the new flexible header is 6. In general, the MCP now does all of its header processing by using the version 6 header format.

To minimize problems in migrating from version 3 and 4 headers to version 6 headers, an intermediate disk file header format was also introduced on the Mark 3.6 release. The version number of the intermediate format header is 5. The version 5 header layout is basically the same as the version 4 header, except that the row address words can be in one or two formats: either the format used by version 4 headers or a new format with an expanded disk address field. When the MCP creates version 5 headers, it creates them with version 4 header format row address words whenever possible. Such headers are fully compatible with Mark 3.5 MCPs and earlier MCPs.

Starting with the Mark 3.8 release, the MCP uses only version 6 headers for disk pack directories and library maintenance tapes. Library maintenance tapes created by the Mark 3.8 MCP and later releases of the MCP have version 6 headers. Library maintenance tapes created with earlier header versions are still usable, but these versions must be converted to version 6.

Understanding Family Header Versions

The concept of a family header version was introduced in the Mark 3.6 release to ease migration from version 4 to version 6 headers. The family header version of a disk or pack family controls the format in which headers are written to the directory of the family. In the Mark 3.8 MCP and later releases of the MCP, only version 6 family headers are allowed.

The family header version is established when a family comes online and is based on the header of the SYSTEMDIRECTORY file of the current base pack of the family. For example, if the header version of the SYSTEMDIRECTORY is 6, the family header version is designated as 6. Treatment of family header versions other than version 6 is described in "Converting Family Header Versions."

Converting Family Header Versions

The Mark 3.8 release and later releases use only version 6 family headers. At halt/load time when the system initializes the disk subsystem, the action that you must take when a family with a header version less than 6 comes online depends on whether the family is the halt/load unit. In each case, the system issues one or more messages that describe the situation.

Halt/Load Unit

The halt/load unit must have a family header version of 6. If it does not, the halt/load cannot proceed. The system issues the following message:

```
HALTLOAD UNIT <family number> <family name> HAS THE WRONG FAMILY
HEADER VERSION, CHANGE HALTLOAD UNIT AND HALTLOAD AGAIN
```

You must either designate a new halt/load unit or use the SYSTEM/LOADER program to load a Mark 3.6 MCP or a later release of the MCP.

Catalog Unit

The catalog unit must have a family version of 6. If it does not, the system pauses the halt/load and issues the following message:

```
CATALOG UNIT <unit number> <family name> HAS THE WRONG FAMILY
HEADER VERSION
```

You must either designate a new catalog unit or halt/load the system to a Mark 3.6 MCP or a later release of the MCP.

Other Units

If any other family does not have a family header version of 6, the system issues the following messages:

```
OLD FORMAT HEADERS ON FAMILY <unit number> <family name>;
IF FORMAT NOT CHANGED, FAMILY WILL NOT BE USABLE

OK TO CHANGE FAMILY HEADER VERSION TO 6: PK <unit number>
<family name>
```

The system then waits for you to respond. System initialization continues while waiting for a response to the second message. You must reply *OK* for the family to be usable. If you answer *DS* (discontinue), the Mark 3.8 MCP and later releases of the MCP cannot use this family.

Additional information about each of these messages can be found in the *A Series System Messages Support Reference Manual.*

Section 8
Safety Mechanisms

The MCP code file, the flat directory of each family, the archive directory for each family, and the catalog on cataloging systems are vital for proper system operation. Although the failure of one of these critical system files is rare, it is still important to ensure that a problem with one of these files does not cause a service interruption. The disk subsystem software enables you to duplicate these four system files so that the system can still operate after the failure of a disk on which a critical system file is stored. This section describes the duplication process for each type of file, and compares the approaches to duplicating these files. This section also explains how to make alternate halt/load families.

The most basic safety mechanism is making backup copies of user disk files on tape or other disk families. The needs of each installation are different, depending on how often the files are changed, how critical the information on each file is, and whether the information can be more easily restored through an audit or other means. This section focuses on safety mechanisms for system files and does not provide details about safety mechanisms for user files. You can use the FILEDATA utility to produce a list of when each user file was last changed or updated. You can use the FILECOPY utility to automate the creation of WFL jobs that make backup copies of user files. Refer to the *A Series System Software Utilities Operations Reference Manual* for more information about FILECOPY and FILEDATA. For information about making backup copies of database files, refer to the *A Series DMSII Utilities Operations Guide.* You can also refer to the *A Series File Attributes Programming Reference Manual* for information about the DUPLICATED and COPIES file attributes.

Duplicating Flat Directories

As the number of disks in a family grows, a large amount of data becomes vulnerable to a single failure of the base pack. If the entire base pack is unusable, none of the data on any of the disks in the family can be accessed. If a particular record in the flat directory is corrupted, the file referenced by that record is not available. You can use the DD (Directory Duplicate) system command to avoid these problems. The DD command makes an exact duplicate of the flat directory on another member of the family.

The duplicate flat directory has the following title:

```
SYSTEMDIRECTORY/<family index number>
```

The variable <family index number> is the family index number of the disk where you want to store the duplicate flat directory for the family. Up to three disks in each family can have a copy of the flat directory (the base pack and two continuation packs). When the system halt/loads or the family is readied, any of the family members containing flat directories can be used as the base pack. The system arbitrarily chooses one as the base pack and treats the others as continuation packs with duplicate directories.

All the family members that contain a copy of the flat directory must be online when you access that family, so that all changes in the directory can be made simultaneously to all copies. If all the disks containing directories are not mounted, then the missing ones become outdated and their flat directories are not accepted by the system as up-to-date duplicates.

If only one of the disks with a duplicate directory is mounted, the system requests the other ones by displaying the following message on the ODT:

```
REQ PK <serial number> ON <family name>
```

You should then mount all the disks in the family that contain a flat directory. If for some reason you cannot mount one of these missing disks, you should enter the OF (Optional File) system reply after the message is displayed on the ODT. The reply OF allows the family to be marked online without the missing disk. After the family is online, you should use the DD– version of the DD system command to delete the flat directory of the missing member.

If a disk with an outdated directory is subsequently mounted, you must be careful to avoid designating the outdated disk as having the most up-to-date directory. If the outdated disk is mounted when the up-to-date base pack is online, the system recognizes the discrepancy and asks what you want to do with the mismatching directory by displaying the following message on the ODT:

```
DIRECTORY NOT CURRENT
```

You can respond to this message in one of two ways:

- Remove the outdated directory with the RM (Remove) system reply, which is equivalent in this situation to the DD– command.
- Replace the outdated directory with the up-to-date directory by replying *OK*. This replacement is equivalent to deleting the outdated directory with the DD– command and then creating a new version with the DD command.

Serious problems can occur if you do mount a disk with an outdated directory, and an up-to-date base pack is not online. The system detects that the up-to-date disk is missing and requests that you mount it. If you then enter *OF* as a response, the family is marked online without that disk. The system cannot determine that the directory on the outdated base pack is out-of-date because the system does not have access to the up-to-date disk to compare timestamps. The system then marks the outdated base pack as even more up-to-date than the offline, up-to-date disk. The file header pointers to areas on the family are invalid on the outdated flat directory, and changes to the outdated flat directory make the more current directory also invalid. This problem can never be corrected, and the system cannot detect the conflict.

If the original base pack is deleted from a family on a cataloging system, the original base pack serial number is still used by the volume library to designate the family. This serial number should be specified when a *VOLUME ADD* or *VOLUME DELETE* WFL statement is used for the family.

Duplicating Archive Directories

You can use the *ARCDUP <family name>* system command to create online duplicates of the archive directories for selected families. The system places all duplicate directories on the catalog family where the base archive directory is located rather than on the family for which the archive directory is used. Therefore, if you want to duplicate archive directories, you must make the catalog family a multipack family.

Another way to protect archive information is to make offline backup copies of the archive directories by using the ARCCOPY (Archive Copy) system command, the WFL *COPY* statement, or both. On a regular basis, you should make backup copies of your archive directories for each disk family at your installation. The frequency at which you perform these backup procedures depend on how often you execute WFL archive statements, because those statements cause new information to be written into the archive directories.

To make backup copies of the archive directories for all disk families on your system, you can issue a series of ARCCOPY system commands that identify the disk family names, followed by a WFL *COPY* statement that moves the backup copies to a tape for safe storage.

The following example shows three ARCCOPY commands that copy the archive directories for the families called DISK, PACK, and DATABANK from the catalog family to the PACK family.

```
ARCCOPY DISK ARCBACKUP/DISK ON PACK
ARCCOPY PACK ARCBACKUP/PACK ON PACK
ARCCOPY DATABANK ARCBACKUP/DATABANK ON PACK
```

When these commands have been processed to completion, you might issue the following WFL statements:

```
COPY & COMPARE ARCBACKUP/= FROM PACK TO ARCBACKUP;
REMOVE ARCBACKUP/=;
```

The COPY statement shown in this example copies the backup files created by the previous ARCCOPY commands to tape and compares each copied file, bit-for-bit, to the original file. The REMOVE statement deletes the backup copies of these files from the PACK family. As an alternative to this procedure, you can copy your archive directories directly to tape. However, you can use this method only if other archive procedures or functions are not executing when you issue the COPY statement.

The following example copies all archive directories from the volume named DISK to the ARCBACKUP volume:

```
COPY & COMPARE SYSTEM/ARCHIVE/= AS ARCBACKUP/=
        FROM DISK TO ARCBACKUP;
```

Restoring Backup Archive Directories

If an archive directory is lost or damaged, you can replace it with its backup copy. The following procedure describes this process:

1. Use the COPY statement to replace the lost or damaged archive directory with its backup copy. To ensure that the entire backup directory is copied onto a single member of the catalog family, use the FAMILYINDEX clause of the COPY statement.

   For example, to restore the backup copy of the archive directory for the DATABANK disk family, issue the following COPY statement. For this example, it is assumed that the DISK family was defined as the catalog family.

   ```
   COPY & COMPARE ARCBACKUP/DATABANK FROM
          ARCBACKUP TO DISK (FAMILYINDEX=1);
   ```

2. Use the ARCREPLACE system command to remove the defective directory and activate the backup directory, as follows:

   ```
   ARCREPLACE DATABANK ARCBACKUP/DATABANK;
   ```

Duplicating Catalog Directories

The AD (Access Duplicate) system command duplicates the system catalog or access structure to protect catalog and volume information. A duplicate of the SYSTEM/CATALOG file is useful on cataloging systems. A duplicate of the SYSTEM/ACCESS directory is useful on systems that run with SECOPT TAPECHECK=AUTOMATIC. On noncataloging systems that do not run with SECOPT TAPECHECK=AUTOMATIC, the SYSTEM/ACCESS directory can be rebuilt more easily than it can be duplicated. You can have up to three copies of SYSTEM/CATALOG or SYSTEM/ACCESS (the original and two duplicates).

Another way to protect catalog information is to make a backup copy of the SYSTEM/CATALOG or SYSTEM/ACCESS file on tape from time to time. The latest catalog copy can be copied back to disk after a catastrophic failure of the catalog file or the catalog family. Refer to "Creating and Using Backup Copies of the Catalog" in Section 5 for more information on this subject.

If you are using the system command SECOPT TAPECHECK=AUTOMATIC on a noncataloging system, you can use GETSTATUS and SETSTATUS calls to save and restore the volume directory. You can use GETSTATUS calls to copy all data records in the volume directory and SETSTATUS calls to delete serial numbers from the volume directory and add records to it. For more information on GETSTATUS and SETSTATUS calls, refer to the *A Series GETSTATUS/SETSTATUS Programming Reference Manual*.

For maximum safety, you might want to have both duplicate online catalogs and backup copies of the catalog. Keeping backup copies of the catalog has one advantage over the use of duplicate catalogs. Changes are made to all online copies of the catalog, and in

rare instances corrupted data can be placed in all the copies, such as in the event of an operator error. However, catalog backup tapes have three disadvantages:

- Changes made to the catalog since the catalog was copied to tape are lost when the catalog file is copied back to disk from tape.
- The system performs lengthy catalog and family rebuilds when the backup copy replaces the old catalog.
- An interruption in system operation is required to copy the backup catalog back to disk.

Duplicating MCP Code Files

If a halt/load family has a duplicate flat directory, it can also have a duplicate MCP code file. There are two reasons to have a duplicate MCP code file. The first is to improve system performance; the second is to provide an alternate halt/load unit.

Having two MCP code files on the halt/load family can improve system performance because the code files can share the I/O operation load. However, performance might not improve significantly if the flat directories of the halt/load family are updated frequently. The closing, creation, and removal of disk files on the halt/load family require additional I/O operations to update more than one directory.

A duplicate MCP code file also provides an alternate halt/load unit to use when the normal halt/load unit fails. Duplicating the MCP code file to provide an alternate halt/load unit in the current halt/load family is different from making an alternate halt/load family. The following differences exist between alternate halt/load families and alternate halt/load units, and your installation can use none, one, or both safety mechanisms:

- If errors occur on one copy of a duplicate MCP code file, the system can still use the other copy and does not require a halt/load to recover from the error. A halt/load is required to switch to an alternate halt/load family.
- Alternate halt/load units require duplicate flat directories and a multidisk family, which can slow system performance if the directories are updated frequently or if many files are stored on the family because it has more than one member. Alternate halt/load families do not require duplicate directories and can be single-member families.
- In extremely rare cases, family failures can be so severe that a duplicate MCP code file or flat directory does not help anyway.
- On systems with removable disks, alternate halt/load families can be stored offline; alternate halt/load units cannot be stored offline.

  Refer to "Making Alternate and Standby Halt/Load Families" in this section for more information about this subject.

The CM (Change MCP) system command is used to duplicate the MCP code file. The duplication process results in a halt/load. The CM command that creates duplicate MCP code files on the halt/load family is as follows:

```
CM <file name> (<family index number list>)
```

The variable <file name> is the name of the MCP code file to be duplicated and the variable <family index number list> is the family index numbers, separated by commas, of the disks to receive the duplicates. Up to three disks can contain copies of the MCP code file (the original and two duplicates), as long as each disk contains a copy of the flat directory.

The following example shows the form of the name that the MCP creates for a duplicate MCP code file:

```
<file name>/FMLYINX<family index number>
```

The variable <file name> is the name of the original MCP code file and the variable <family index number> is the three-digit family index number of the disk that contains the duplicate.

You can reduce the time needed by the MCP to do the CM operation by first copying the MCP code file (giving it the desired name) to the halt/load family members that are to become alternate halt/load units. Unlike the DD (Directory Duplicate) and AD (Access Duplicate) system commands, the CM command skips the copy operation if it locates matching files with the proper names and timestamps on the proper family members.

The following series of statements shows how to use the WFL *COPY* statement to save time in the CM operation. In this example, the members of the family HLDISK with the family index numbers of 1 and 2 receive copies of the MCP code file. MCPBACKUP is the name of the tape containing the backup MCP code file, and SYSTEM/MCP is the name of the backup MCP code file.

```
COPY SYSTEM/MCP AS SYSTEM/MCP/FMLYINXØØ1 FROM MCPBACKUP
TO HLDISK(FAMILYINDEX=1,PACK)

COPY SYSTEM/MCP AS SYSTEM/MCP/FMLYINXØØ2 FROM MCPBACKUP
TO HLDISK(FAMILYINDEX=2,PACK)
```

After the copy operation completes, issue the CM command as follows:

```
CM SYSTEM/MCP/FMLYINXØØ1(1,2)
```

The system waits for null mix with all libraries terminated. To force the CM to proceed, issue a primitive CM command as follows:

```
??CM SYSTEM/MCP/FMLYINXØØ1
```

Monitoring Directory Duplication

If you have duplicate flat directories, catalog files, or MCP code files, you should monitor the duplication operation at least once each day or after each halt/load to ensure that all the required duplicates are still active. It is possible for an operator error or system error to cause the system to stop using a duplicate MCP code file, catalog, or flat directory. You can use the PD (Print Directory) system command to examine the

duplication and then restart the operation if one of the duplicate files has become invalid. By issuing the appropriate PD command, you can determine which files are in use. If a flat directory, catalog, or MCP code file is not shown in the PD display as "IN USE", then it is not a valid duplicate. The PD command syntax can have several forms:

- Use the following command syntax for the catalog on cataloging systems:

  ```
  PD SYSTEM/CATALOG/= ON <family name>
  ```

  The variable <family name> is the name of the family where the catalog is stored.

- Use the following command syntax for the access structure on noncataloging systems:

  ```
  PD SYSTEM/ACCESS/= ON <family name>
  ```

  The variable <family name> is the name of the family where the access structure is stored.

- Use the following command structure for flat directories:

  ```
  PD SYSTEMDIRECTORY/= ON <family name>
  ```

- Use the following command syntax for the running MCP code file:

  ```
  PD <file name>/= ON <family name>
  ```

  The variable <file name> is the name of the MCP code file, and the variable <family name> is the name of the halt/load family.

- Use the following command syntax for archive directories:

  ```
  PD SYSTEM/ARCHIVE/= ON <family name>
  ```

  The variable <family name> is the name of the family where the catalog or access is stored.

Comparing Duplication Commands

This subsection discusses the functional and syntactical differences and similarities of flat directories, archive directories, catalogs, and MCP code files. These differences and similarities can affect the duplication of these files.

The AD (Access Duplicate), ARCDUPLICATE (Archive Duplicate), DD (Directory Duplicate), or CM (Change MCP) system command can be used to make up to two exact copies of a given file, for a total of three copies. All four commands place the duplicates on different members in the same family; the file name assigned by the system indicates the family index number of the disk the file is located on. However, the form of the name for the AD, ARCDUPLICATE, and DD commands differs from that of the CM command. Each copy of the file is always fully contained on one member of the family.

The CM command ensures that the copies match the original by checking the MCP code file timestamp during the CM operation. Duplicate directories created with the AD, ARCDUPLICATE, or DD command have special internal timestamps that are checked and changed after every halt/load or every time a disk is readied. Because of an operator

or system error, it is possible for an error in the timestamps to occur after a directory is duplicated, so that the timestamps do not match. If that occurs, the system might cancel the AD, ARCDUPLICATE, or DD operation. The system can still use the original copy of the directory, but you must reenter the AD, ARCDUPLICATE, or DD command to restore the duplication process. If the DD operation is canceled for a halt/load family member, any duplicate MCP code file for that family member is also deleted.

Duplicate directories created by the DD command and duplicate catalogs created by the AD command lead to extra system overhead, because more than one file must be updated every time a file header is added, deleted, or changed. Duplicate archives created by ARCDUPLICATE commands lead to only a small increment in overhead during the execution of WFL archive statements. The duplicate MCP code file created by the CM command does not lead to any direct overhead, except that a duplicate MCP code file can be used only on a family that has a duplicate flat directory.

The AD, ARCDUPLICATE , and DD commands have almost the same syntax. You must specify the family index number of the disk that is to receive the copy. For the ARCDUPLICATE system command, you must specify not only the family name for which the archive directory is to be duplicated, but also a family index on the catalog family where the duplicate is to be placed. The syntax of the CM command requires that you list the family members that are to receive a copy. If you do not supply this list, the CM command uses the last CM command list supplied for the family. If this is the first time the CM command is used for that family and no list is supplied, the command defaults to family index 001.

The AD–, ARCDUPLICATE–, and DD– commands enable you to delete a duplicate copy of a directory. The CM command does not provide a direct method to delete a duplicate MCP code file. (The CM– command cancels an impending change in the MCP.) To delete a duplicate MCP code file, you must enter the CM command with a family index list that omits the family index number of the disk where the unwanted duplicate is stored.

You can use the DD and CM commands for any multidisk family. You can use the ARCDUPLICATE command when the catalog family is a multipack family, even if the family for which the archive is to be duplicated is not a multipack family. The AD command applies only to the current catalog, so the syntax does not require a family name or file name.

You can use the AD and ARCDUPLICATE commands on a multidisk family whether or not the DD command has been used on that family. Even if the directory of the family has been duplicated with the DD command, the AD and ARCDUPLICATE commands are not restricted to placing the duplicate catalog on a family member that contains a duplicate flat directory. You can use the CM command to place a duplicate MCP code file on a family member only if that member contains a duplicate flat directory. However, every member that has a duplicate flat directory need not receive a copy of the MCP code file. For example, there might be two copies of the flat directory but only one copy of the MCP code file.

Making Alternate or Standby Halt/Load Families

You can use an alternate halt/load family or a standby halt/load family if the normal halt/load family fails. Making an alternate halt/load family is different from making an

alternate halt/load unit on the current halt/load family. The discussion "Duplicating MCP Code Files" in this section describes the differences between alternate halt/load families and alternate halt/load units, and your installation can use none, one, or both safety mechanisms.

There are two ways you can effectively use the command CM <file name> ON <family name> to make an alternate halt/load family. Each is based on the kind of disk media being used and whether or not your installation uses the MIRRORING subsystem. The methods are discussed in the information that follows.

Using Online Disks

1. Copy the MCP code file to several families that are usually online and then use one of the following command syntaxes to designate those families as alternate or standby halt/load families:

```
CM <file name> ON <family name>

CM <file name> ON <family name> + STANDBY
```

   Note: *If your installation runs with OP + MIRRORING enabled, you should always use the + STANDBY form of the command. Standby is the preferred method for large systems that use partitioning such as the A 15 and the A 17.*

2.

   If the halt/load family experiences a failure, change to one of the alternate halt/load families and run from it while you restore the original halt/load family.

Using Removable Disks Offline

1. Make one or more complete copies of the halt/load family on selected disks. Use the following command syntax to designate those disks as alternate halt/load families:

```
CM <file name> ON <family name>
```

2. Power off those disks and store them offline.
3. If the halt/load family experiences a failure, retrieve one of the offline families and continue operations with it, including the operation of making a replacement alternate halt/load family.

Note: *Do not halt/load using an alternate halt/load family created by this method on a system that uses OP+MIRRORING. The result is full stop.*

This technique can be made even more reliable by rotating these offline backup families as the online halt/load family. This rotation is a way of checking that each backup family is intact. Note that if SYSTEM/ACCESS or SYSTEM/CATALOG file is stored on the halt/load family, time-consuming rebuilds must occur each time the halt/load family is changed.

The command syntax CM <file name> ON <family name> uses the designated MCP code file on the designated family, but it copies a large number of system attributes from the running halt/load family. These attributes cover everything from the host name to the list of saved and reserved units. For this reason, you might want to change a few of these system options (such as CATALOGING or OKTIMEANDDATE) just before you enter the CM command. Then, when the alternate halt/load family is finally used as a halt/load unit the system will not be running with the wrong options.

After you have designated an MCP code file with the CM command, the system marks the code file as a nonremovable file. If you later try to copy in a new version of the file, the copy attempt stops and the following message appears on the ODT:

```
DUP FILE(SYSTEM FILE)
```

Enter *CM– ON <family name>* and remove the old MCP code file before starting the copy operation. After the CM– operation is completed, the disk is no longer a halt/load-capable family until the new version of the MCP code file is copied and the CM command has been used again to designate it as an MCP code file.

If you have removable disks, you may want the backup halt/load families to have the same name as the current halt/load family. It is then easier to switch from one halt/load family to another. However, if you are running on a cataloging system, never use the *VOLUME ADD* WFL statement to add a family that does not have a unique family name. To avoid conflicts between the name of the backup halt/load family and the active halt/load family, you can use the following procedure to make backup halt/load families:

1. Relabel the backup disk with the LB (Relabel Pack) system command to have a temporary alias.
2. Do all the copy and CM command operations with the alias.
3. Save the unit with the SV (Save) system command.
4. Relabel the disk with the LB command to have the proper name.
5. Use the POWER (Power Up/Down) system command to power down the disk drive unit.
6. Store the backup disk offline.

If these precautions are not taken, the system repeatedly issues the error message "DUP FAMILYNAME" during the copy and CM command operations.

Section 9
Disk Resource Control System

The disk resource control (DRC) system is an optional feature that provides the ability to control disk space usage on a per user basis. Its information resides in the SYSTEM/USERDATAFILE; disk resource controls can be specified through the SYSTEM/MAKEUSER utility or through the programmatic interface to USERDATA. The DRC system is not a security system, but normal security checking occurs. The DRC system does not support interchange disk packs.

The DRC system enables site managers to control the following allocations of disk space:

- The amount of space on a family at any point in time that a user can have for permanent files
- The amount of space on a family on an ongoing basis (disk integral) that a user can have for permanent files
- The total amount of temporary space on the system at any one time that the task of a user can have

DRC System Features

The following information tells about using the DRC (Disk Resource Control) system command, limiting the use of permanent files, limiting the use of temporary files, and handling user errors.

Using the DRC System Command

The DRC (Disk Resource Control) system command can be used to make the disk resource control system active or inactive. A halt/load is required after the system is turned on to actually bring up the DRC system. The command can also be used to inquire about the current state of the DRC system and to perform certain maintenance operations. Refer to the *A Series System Commands Operations Reference Manual* for more information about the DRC system command.

You can also use a selection on the MARC System Control screen to perform the DRC command.

Handling Permanent Files

The DRC system allows site management to designate disk resource controls for a user. The information concerning these controls resides in the user's entry in the SYSTEM/USERDATAFILE and is designated by using the SYSTEM/MAKEUSER utility. The DRC system does not need to be active for user specifications to be made. If

a user entry in the SYSTEM/USERDATAFILE does not contain disk resource controls, no disk resource controls are enforced for that user.

A user entry in the SYSTEM/USERDATAFILE can designate the disk resource controls for a list of families. Each family is an entry in the family list and can contain usage and limit information. The DRC system stores in each family entry of the family list the total amount of permanent file space currently owned by a user along with the disk integral (the maximum possible space limit on an ongoing basis). The DRC system updates values in the SYSTEM/USERDATAFILE at the following times:

- When a permanent file is expanded
- When a temporary file is changed to a permanent file
- When a permanent file is crunched
- When the ownership of a file is changed
- When a permanent file is removed
- When a file is shrunk or expanded with the EXCHANGE intrinsic
- When a file is changed from temporary to permanent with the CHECKPOINT intrinsic

Each listed family can designate a maximum space limit to be owned by the user at any one time as well as a maximum possible space limit on an ongoing basis. Either or both of these limits can be designated as unlimited. If either limit is designated as unlimited, usage information is gathered, but the limit is not applied.

Defaults can also be designated for the space limits for families not included in the list of families for the user. The DRC system uses these defaults to determine user limits when someone is using a family that is not designated in the list of families. If the defaults are 0, the user can allocate space only on the families designated in his family list. Either or both of the default limits can be designated as unlimited. If either limit is designated as unlimited, usage information is gathered, but the limit is not applied. The defaults can be designated without specifying a family list.

The DRC system makes entries in the family list whenever space is allocated for a permanent file for a user on a family that was not previously in the family list. The default limits of the user are assigned to the family entry. For information about making disk resource control specifications in the SYSTEM/USERDATAFILE, refer to the information on MAKEUSER in the *A Series Security Administration Guide*.

The system enforces disk resource controls when a user allocates space for a file if the DRC system is active. User actions that cause space allocation can include changing a temporary file into a permanent file and changing the ownership of a file. When the user who is allocating space is the owner of the file, the limits enforced are the limits designated in the SYSTEM/USERDATAFILE for that particular user. When a user attempts to allocate space for a file owned by someone else, the space limits applied are those of the owner. Files that do not reside under a usercode in the system directory (* files) are not limited by the DRC system.

The DRC system does not limit space allocation for the following files:

- DMSII database files
- Job code files created by WFL
- Recovery files created by MCSs (files of FILEKIND RECOVERYFILE)
- Overlay files and other such system files
- Backup files created under the default system directories

Handling Temporary Files

Site managers can designate an overall temporary file space limit for a user that is applied to all tasks initiated by that user. This information resides in the user's entry in the SYSTEM/USERDATAFILE.

Site managers can also designate a temporary file limit for a task by assigning the task attribute TEMPFILELIMIT for that task. The current in-use value can be interrogated by using the TEMPFILEMBYTES task attribute. For more information regarding these task attributes, refer to the *A Series Task Attributes Programming Reference Manual*. For information about making disk resource control specifications in the SYSTEM/USERDATAFILE, refer to the information on MAKEUSER in the *A Series Security Administration Guide*.

Disk resource controls for temporary files are enforced when the system attempts to allocate space for a temporary file. The DRC system does not allow a process to exceed the specified limit.

Handling User Errors

The DRC system detects errors when a user exceeds or attempts to exceed a family in-use limit or a temporary file limit, or when a user exceeds a family integral limit. After it detects the error, the DRC system informs the user about the error.

The system treats disk resource control errors as a form of I/O error. You can handle them programmatically as with other I/O errors. Programs that do not programmatically handle disk resource control errors are discontinued as they would be for any other I/O error.

Except for Message Control Systems (MCSs), the first disk resource control error encountered for each file causes an error message to be displayed for the program. The message gives the exact nature of the error.

After any disk resource control error, you can interrogate the STATE file attribute (IORESULT or IOERRORTYPE file attribute for direct I/O) to determine the exact nature of the error. Programmatically handling a disk resource control error does not allow the program to bypass the error; it only prevents the program from being discontinued.

When library maintenance encounters disk resource control limit errors, it handles them in a somewhat different but compatible manner. When a library maintenance task encounters a disk space limit error, the system displays an error message and skips

the file that encountered the error. An error also causes the task value to be given the value 1; this action allows the user to check if something went wrong during the library maintenance run. The limits on temporary files are not applied by library maintenance.

When BNA file transfer encounters a disk space limit error, BNA displays an error message and does not copy the file.

If the user receives a family in-use limit error, he can make space available space by removing files that belong to him from the family. If the user receives a family integral limit error, he cannot allocate space until the integral usage value for his family in the SYSTEM/USERDATAFILE has been changed to less than the integral limit.

Refer to the *A Series File Attributes Programming Reference Manual* for more information about the DRC limit errors that can be returned. Refer to the AVAILABLE file attribute for values that can be returned as a result of the CLOSE system command. Refer to the file attributes STATE, IORESULT and IOERRORTYPE for values that can be returned by read or write operations.

Warning the User about Family Substitution Changes

CANDE displays warning messages during the log-on process or at any time that a user changes his family substitution if the SYSTEM/USERDATAFILE entry of the user would prevent him from allocating space on his primary family.

DRC Operations

The following information explains how to operate the DRC system.

Activating the DRC System

Perform the following set of operations to activate the DRC system:

1. Enter the following system command:

   ```
   SL DRCSUPPORT = <drcsupport code file> ON <family name>
   ```

   On the release tapes, < drcsupport code file > is called SYSTEM/DRCSUPPORT.
2. Enter the following system command:

   ```
   DRC +
   ```

   The system displays the following message in response to the command:

   ```
   THE DRC SYSTEM WILL BE ACTIVE AFTER THE NEXT HALT/LOAD
   ```
3. Halt/load the system.

When the system comes back up, the DRCSUPPORT library is brought back up. The first time that the system is activated, the DRCSUPPORT library comes up on a "NO FILE" RSVP for the file SYSTEM/DRC/FAMILYAUDIT on the halt/load pack. Enter *OF* to create the file. The DRC system is now active.

Note: *The system waits for the DRC system to become active before system initialization is completed. Jobs are queued to prevent the manipulation of usercoded files before the DRC system is active so that correct disk space usage values can be recorded. While the system is waiting for the DRC system to initialize, it displays an RSVP message that gives you the option to continue system initialization with an inactive DRC system.*

WARNING

Jobs queued during DRC initialization can be started before DRC is active by using the FS (Force Schedule) system command. However, be aware that in doing so, you might cause unexpected results if the job handles usercoded files.

The system ignores any attempt to change or remove a usercoded file before DRC is initialized. The system can execute a WFL *COPY* statement before DRC is initialized, but DRC does not account for any disk space usage that was affected by the *COPY* statement. Incorrect DRC disk space usage can also occur if you use the primitive ??COPY system command to duplicate usercoded files during DRC system initialization.

To ensure that the existing disk space usage is correctly recorded by the DRC system, disk resource controls should be established for all usercodes defined in the SYSTEM/USERDATAFILE before the DRC system is activated. Refer to "DRC Restrictions" later in this section.

Deactivating the DRC System

To deactivate the DRC system, enter the following system command:

```
DRC -
```

The system responds with the following message:

```
THE DRC SYSTEM IS TERMINATING
```

The DRCSUPPORT library is brought down and leaves the mix. The DRC system is now inactive.

You can also use the DRC – command to cancel a pending DRC + where the DRC + command has been entered but a halt/load has not yet been performed.

Making Inquiries about the Status of the DRC System

To inquire about the status of the DRC system, enter the following system command:

```
DRC
```

The system responds with one of the following messages:

```
THE DRC SYSTEM IS INACTIVE

THE DRC SYSTEM WILL BE STARTED AFTER THE NEXT HALT/LOAD

THE DRC SYSTEM IS INITIALIZING

THE DRC SYSTEM IS ACTIVE

THE DRC SYSTEM IS TERMINATING
```

For more information about the DRC system command, refer to the *A Series System Commands Operations Reference Manual*.

Creating SYSTEM/USERDATAFILE Entries for Disk Resource Control

This information provides various examples of how you can use the SYSTEM/MAKEUSER utility to set up SYSTEM/USERDATAFILE entries with disk resource controls. Refer to the information on MAKEUSER in the *A Series Security Administration Guide* for more information about syntax and semantics for DRC SYSTEM/USERDATAFILE specifications.

Disk resource control limits and usage values are in units of megabytes (2 ** 20 bytes) for OTHERFAMILYLIMIT, FAMILYLIMIT, and FAMILYINUSE. DRC limits and usage values are in units of megabyte days for OTHERFAMILYINTLIM, FAMILYINTLIMIT, and FAMILYINTEGRAL. A limit designated with a negative value is treated as unlimited.

To ensure the correct recording of disk space usage by the DRC system, a user entry in the SYSTEM/USERDATAFILE should have disk resource controls established before any files are created under that usercode. Refer to "DRC Restrictions" later in this section.

Examples of SYSTEM/USERDATAFILE Entries

The following SYSTEM/MAKEUSER examples show how you can designate various limits for the SYSTEM/USERDATAFILE of a specific user.

Example 1

```
USER DRCUSER MINPW=1 MAXPW=1 PASSWORD=DRCPASSWORD;
```

This user is not limited by the DRC system, and in-use values are not accumulated.

Example 2

```
USER DRCUSER OTHERFAMILYLIMIT = -1 OTHERFAMILYINTLIM = -1;
```

This user is not limited by the DRC, but usage values are gathered for all families on which the user allocates space.

Example 3

```
USER DRCUSER FAMILYLIST AT FAMILYNAME = "MYPACK" (FAMILYLIMIT = -1
FAMILYINTLIMIT = -1);
```

This user can allocate space only on family MYPACK because OTHERFAMILYLIMIT and OTHERFAMILYINTLIM are not designated and are therefore 0. The amount of space that the user can allocate on MYPACK is not limited, but usage values are gathered for it.

Example 4

```
USER DRCUSER OTHERFAMILYLIMIT = -1 OTHERFAMILYINTLIM = -1
    FAMILYLIST AT FAMILYNAME = "MYDISK"
        (FAMILYLIMIT = 2.5  FAMILYINTLIMIT = -1),
    AT FAMILYNAME = "MYOTHERDISK"
         (FAMILYLIMIT = 10  FAMILYINTLIMIT = 900);
```

This user can allocate space on MYDISK up to 2.5 megabytes. On MYOTHERDISK he can allocate up to 10 megabytes or until 900 megabyte days. He is not limited on any other families, but usage values are gathered for them.

Example 5

```
USER DRCUSER OTHERFAMILYLIMIT = 10  OTHERFAMILYINTLIM = 1000
    FAMILYLIST AT FAMILYNAME = "MYDISK"
        (FAMILYLIMIT = 20  FAMILYINTLIMIT = 2000);
```

This user can allocate space on MYDISK up to 20 megabytes or until 2000 megabyte days. On any other family, he can allocate space up to 10 megabytes or until 1000 megabyte days.

Example 6

```
USER DRCUSER;
```

This is an inquiry about DRCUSER as it was set up in Example 5, with space having been allocated under usercode DRCUSER on families MYDISK and DISK. SYSTEM/MAKEUSER returns the following information:

```
USER = DRCUSER
     MAXPW = Ø
     OTHERFAMILYLIMIT = 1Ø
     OTHERFAMILYINTLIM = 1ØØØ
     FAMILYLIST            % GROUP: LENGTH=16 WORDS, 2 ENTRIES
          AT  FAMILYNAME = ( "MYDISK" )
           (  FAMILYINUSE = 4"262333333333" %       2.2ØØ
              FAMILYLIMIT = 2Ø
              FAMILYINTEGRAL = 4"2646666666" %        4.4ØØ
              FAMILYINTLIMIT = 2ØØØ
              FAMILYTIMESTAMP = 14:18:25 Ø8/14/88
                                                   )
         ,AT  FAMILYNAME = ( "DISK" )
           (  CREATEDBYSYSTEM
              FAMILYINUSE = 4"274D85ØØØØØØ" %       Ø.Ø76
              FAMILYLIMIT = 1Ø
              FAMILYINTEGRAL = 4"269374BC6A7F" %       Ø.152
              FAMILYINTLIMIT = 1ØØØ
              FAMILYTIMESTAMP = 14:18:25 Ø8/14/88
                                                   )
```

For the family MYDISK, the FAMILYINUSE value is 2.2 megabytes and the FAMILYINTEGRAL is 4.4 megabyte days. For the family DISK, the FAMILYINUSE value is 0.076 megabytes and the FAMILYINTEGRAL is 0.152 megabyte days. The family list entry for DISK has been created by the DRC system with the default limits because the disk was not specified in the family list.

Maintenance of Integral Limits

After an integral limit is exceeded, the user cannot allocate space on the family until the security administrator changes the SYSTEM/USERDATAFILE entry of the user. This process can be accomplished by changing the FAMILYINTEGRAL usage value so that it is again below the FAMILYINTLIMIT.

SYSTEM/USERDATAFILE Entry Overflow

The DRC system makes entries in the family list whenever space is allocated for a permanent file of a user on a family that was not previously in the family list. If the DRC system needs to add to the family list of a user and there is not room in the SYSTEM/USERDATAFILE entry (the maximum size of a SYSTEM/USERDATAFILE is 240 words), the DRC system tries to delete a family list entry and to log the fact that this has happened. The family list entry selected for deletion (if possible, a

CREATEDBYSYSTEM family) is the one with the least FAMILYINUSE amount. If two family lists have the same FAMILYINUSE amount, the one with the lowest FAMILYINTEGRAL is selected. If two family lists have the same FAMILYINTEGRAL, the one with the oldest FAMILYTIMESTAMP is selected. If it is not possible to find space in the SYSTEM/USERDATAFILE entry, the fact that this family list entry could not be put in the entry is logged. Refer to the SUMLOG information in the *A Series System Software Support Reference Manual* for more information about the records that were logged.

DRC Restrictions

While the DRC system is active, do not install a different SYSTEM/USERDATAFILE. For example, do not use the SYSTEM/MAKEUSER utility CREATE command. If you install a different SYSTEM/USERDATAFILE, it could adversely affect the DRC system.

While the DRC is active, you cannot remove or change a *USERCODE/= directory from any family. The *USERCODE/= directory is a pseudodirectory that contains all the files with usercodes; that is, it contains all files whose file names begin with a usercode instead of an asterisk (*). If you attempt this action, the system does not perform the remove or change, and it generates an error message.

If a usercode is newly set up with disk resource controls while the DRC system is active, the FAMILYINUSE count for that usercode does not include any space allocated for files created before the usercode was set up. To include such space in the usage values, the SYSTEM/DRC/FAMILYAUDIT file must be removed and a halt/load must then be performed to reactivate the DRC system.

To ensure that all disk space usage is recorded correctly by the DRC system, establish disk resource controls for each user entry in the SYSTEM/USERDATAFILE before you first activate the DRC system. If the space usage of a user does not require that limitations be enforced by the DRC system, designate the default limits as unlimited. This designation enables usage values to be gathered for that usercode. After the DRC system is active, each new user entry that is added to the SYSTEM/USERDATAFILE should have disk resource controls set up before any files can be created under that usercode.

Section 10
Mirrored Disk Feature

On systems that use data link processor (DLP) I/O, the Mirrored Disk feature allows from two to four disks to be maintained as a *mirrored set,* that is, as exact copies of each other.

This section provides an overview of the Mirrored Disk feature and covers the following topics:

- Benefits of the Mirrored Disk feature
- Requirements
- Site options
- Initiation
- I/O handling
- Operations

Benefits

The Mirrored Disk feature increases both system availability and data integrity. If one copy in a mirrored set is destroyed, goes offline, or experiences any irrecoverable errors, another online copy in the set will allow normal functioning to proceed. Disk mirroring is completely transparent to applications. In the event of an error on one mirror, the application proceeds normally, and the system notifies the operator about the error.

For mirrored disks, you should evaluate the backup and audit features used at your site. Because mirroring significantly decreases the possibility of data loss due to equipment or media malfunction, the number of available processing hours on a system is increased.

Mirroring disks can also improve I/O throughput on disk subsystems that experience a high ratio of reads versus writes. This improvement occurs because the MCP distributes the read operations equally to all the mirrors in a set. If a disk subsystem has a low ratio of reads to writes, the I/O throughput can be reduced, because writes must be issued to all mirrors in a set. The actual impact of mirrored disks varies according to the individual characteristics of the installation.

Mirrors of existing disks can be created without bringing disks offline or interrupting use by the system.

Requirements

No special hardware is required for disk mirroring. However, disk units must be available to allow for the redundancy.

The Mirrored Disk feature applies to all supported disk types except memory disks and can be used with both cataloging and noncataloging file systems.

Options

You designate which disks are to be mirrored, the number of disks in a set, and the units on which mirrored disks are to reside. The Mirrored Disk feature puts few restrictions on these choices, except that disks within a given mirrored set must be of the same model. (See "Configuration Recommendations" in this section for system configuration guidelines and for the location of mirrors within a set.)

After they are created, mirrored sets are maintained automatically across system interruption. Operator action is required only in certain exception conditions.

You can remove mirrors from any mirrored set or create new mirrors for any set while the system is in normal operation and while the disks are in use. You can also move mirrored sets between systems and within systems.

You can specify recovery options for any mirrored set with the MIRROR OPTION system command. This option determines the action to be taken after a halt/load if certain critical MCP information has been lost.

Initiation

When the Mirrored Disk feature is chosen for some or all of the disks in a system, turn on the MIRRORING option by entering the system command OP + MIRRORING. Then halt/load the system to create the internal structures needed for mirroring. Thereafter, you can create mirrors of existing disks by using the system command MIRROR CREATE. Other MIRROR commands will also be valid.

You can also use the OP + MIRRORING command when you move a disk subsystem to a new system, or when you cold start a system with preexisting mirrored sets. After the system executes an OP + MIRRORING command, the system brings the mirrored disks online following the next halt/load.

Creating Mirrored Disks

After mirroring is initiated, you can create mirrors of selected disks by using the system command MIRROR CREATE. This command copies the source disks and brings the destination disk online. The source disk can already be a part of an existing mirrored set. The source disks remain online during the copy process.

The destination disk must not have any open files on it and must not be a member of a mirrored set when you issue the MIRROR CREATE command. The system overwrites all files on the destination disk during the mirror creation process. Therefore, before you enter the MIRROR CREATE command, copy to another disk any files you want to save from this disk.

If errors occur while the process is copying data during mirror creation, the defective sectors are listed in a report file on the halt/load unit. The report file has the following name:

```
CREATE/<source name>/FAMILYINDEX <source family index>/SECTORSINERROR
```

Configuration Recommendations

To ensure full redundancy of disks, it is best to maintain separate I/O paths to mirrors within a set. Because disk mirroring is managed by the MCP, failure of not only disks, but of controllers and other hardware can be handled if the system is so configured. However, maintaining separate paths is not required by the mirrored disk feature. The decision is up to the site manager. Other configuration options, such as multiple paths to disks through different controllers and exchanges, can be just as beneficial.

I/O Handling

I/O operations are handled differently for mirrored disks than for nonmirrored disks. As described in the following sections, read operations are issued to only one disk within a mirrored set, and write operations are issued to all disks within a mirrored set.

In the following discussion, the terms *online*, *offline*, and *pending* have special meanings. Online refers to current members of a mirrored set that are available and synchronized. Offline refers to disks that are in a state of write lockout, are not ready, or are otherwise unavailable to the system because they are being audited. Pending disks are those that are returning from an offline to an online state, mirrored set members being created, and all members of a mirrored set that has members not yet seen by the system. All three states are mutually exclusive.

Read Operations

Reads are issued to each online disk in turn within a mirrored set. If an I/O error occurs on a read operation to a disk, the next disk in the mirrored set is selected and the read is issued to that disk.

NOT READY and WRITE LOCKOUT exceptions cause the system to place the disk in an offline state and to start an audit for the disk. Any other read errors release the disk from the mirrored set. If errors occur to all online disks within a set, the application program receives an error on the I/O; otherwise, the errors are transparent to the application.

Offline and pending members are not candidates for reads.

Write Operations

Write operations are issued to all online disks within a set. If an irrecoverable write parity error occurs, the system releases the disk from the set. For NOT READY and WRITE LOCKOUT errors, the system marks the disk offline and starts an audit for

the disk. In either case, the system notifies the operator, who should take appropriate action. An application program can receive an error on a write only if errors are encountered by all online disks within a set.

If an irrecoverable write error occurs on a mirror, the system releases the disk from the mirrored set and issues a message to the operator to indicate that the release has occurred. The system modifies the disk label so that you cannot use the disk without reconfiguring it with the RC (Reconfigure Disk) system command. In this situation, you must resolve the problem and decide whether or not to create a new mirror for the set.

If a disk goes offline, the system automatically starts an audit for the disk and issues a message to the operator that identifies the disk as being in a WRITE LOCKOUT or NOT READY state.

Whenever a write to a disk is directed to a mirrored set, it is recorded in an MCP table called the outstanding write list (OWL). When the writes to associated disks are complete, the system removes the entry from the OWL. After a halt/load, the OWL updates disks within a set so that they are identical.

If the OWL is destroyed, the disks might not be identical, because the writes in process to a set might not have completed to all members of the set. The system command MIRROR OPTION allows you to specify appropriate action in the event the OWL is lost.

Audits

An audit table of write operations is kept for offline mirrors. This audit table is a bit map of the disk in question. Each bit in the table corresponds to a fixed area on the disk. Therefore, the size of the audit table is fixed and is a function of the number of cylinders defined for the disk type. Whenever a write is issued to a mirrored set, corresponding bits are turned on in the audit table of each offline member.

The audit table is abandoned if the offline disk is released from the mirrored set. Packs being audited at the time of a halt/load are released from the mirrored set.

When an offline disk is returned to an online state, the system uses the audit table to update that disk.

Operational Information

Mirrored disks can be moved within a system just like nonmirrored disks. Mirrored sets can also be moved between systems and MCPs.

Moving Packs within a System

You can move members of a mirrored set within a system while it is down with no ill effects.

To move a pack within a system, either use the system command MOVE or power off the disk drive and move the pack to a new drive. In either case, the pack being moved is

audited just as any other offline copy. The rest of the mirrored set remains online and usable. After you have moved the pack to the new drive, the audit is applied to update the pack.

For fixed media, you can copy data to other units with the system command MIRROR CREATE. After the new copy is complete, you can release the source disk by using the MIRROR RELEASE command.

Moving Packs between Systems

It is possible to move mirrored sets between systems by physically removing them from one system and installing them in another. Before being moved, however, the mirrored set must be closed (prior to being freed). The system updates the disk labels to indicate that the disks are closed. After you move the mirrored set, ready it with the system command RY (Ready). When the disks are readied by another system, the new system reads the label to determine if the disk was closed before it was moved and to determine the number of valid copies in the mirrored set.

If sets that were not closed are moved to a new system, they are not brought online as a set unless they have a recovery type specified as DMS using the MIRROR RECOVERY command (see the *A Series System Commands Operations Reference Manual* for more information).

Mirrored sets with a recovery type of DISCARD are broken (mirroring is discontinued) when one disk is brought online as a nonmirrored unit. The other members of the set are released (the pack label is invalidated, the pack is not brought online, and a message is issued).

Mirrored halt/load units cannot be used to halt/load on another system. This action results in a dead stop.

Offline Packs Returning Online

When you bring a mirrored disk online, the system verifies that mirroring has been established for that system. If mirroring has not been established, the system displays the following message and terminates the process of bringing the disk online:

```
PK<unit number> [<serial number>](<family name>) ERR:
MIRRORED PACK ON NONMIRRORED SYSTEM
```

You can bring a mirrored disk online as a nonmirrored disk only by issuing an RC (Reconfigure Disk) or PG (Purge) system command for the closed unit.

When a mirrored disk is readied on a mirroring system, one of the following conditions is required:

- The disk must have been closed prior to going offline.
- The system must have a record of the disk in question.
- The disk must have a recovery type of DMS.

If the disk was closed, the disk is brought online when the other members are seen by the system, or when audits are begun (with a MIRROR AUDIT system command) for those members not yet seen by the system.

If the disk was not closed, but the disk is recorded in the system tables, the system does one of the following:

- If a halt/load occurred while the disk was offline or the disk encountered a parity error, the disk is released (that is, the disk label is invalidated and the disk is not brought online) and a message is issued.
- If the disk went offline and is currently being audited, the audit is applied to the disk. When the audit is complete, the disk is returned online.
- If the disk was online and is returning after a halt/load, it is brought online when the other members are seen by the system. Any writes that were outstanding at the time of the halt/load are applied.
- If the OWL was corrupted or lost during the halt/load, the set is brought online only if it has a recovery type of DMS; otherwise, only one disk of the set is brought online as a nonmirrored unit, and the other members are released.
- If a mirrored disk returning online has no history of being present on the current system (or the disk has been in use on another system) and the set was not closed (by means of the system command CLOSE), then the set is brought online only if its recovery option is DMS.
- Similarly, if the OWL is lost during a halt/load, only those sets with recovery option DMS are brought back online. When these situations arise for sets with recovery option DISCARD, only one disk is brought online as nonmirrored, and the other members are released.

  Following a halt/load (or when a mirrored set is moved to a new system), a set does not go online until all disks previously online (or previously part of the closed set) are visible to the system. If some of the members are absent, or the operator does not wish to wait for all the members to be ready before using the set, the absent members can be forced into an audit state and the set brought online with the MIRROR AUDIT command.

  Sets that are in the process of being brought online are called *partial mirrored* sets. If a task is initiated that causes a list of partial mirrored sets to be displayed, and if that task becomes a waiting entry, the list of partial mirrored sets is redisplayed whenever a change occurs in the state of a mirrored set or when the operator reactivates the waiting entry using the system command OK. Partial mirrored disks are identified with a lowercase m in the system commands PER and OL.

Recovery

When a system is halt/loaded with mirrored critical units (that is, units required for the system to run), offline copies of those units are automatically audited until they are seen by the system. This allows the system to start and you to take appropriate action if one of the members is missing or corrupted before, during, or after the halt/load. Noncritical units are handled as described in "Offline Packs Returning Online" in this section.

Note: You cannot move mirrored halt/load units between mirrored systems. The result is a dead stop.

During a halt/load, all needed structures are created or recovered. The mirror information table (MIT), which contains information concerning all mirrored sets on the system, is stored on the halt/load disk as an MCP structure and is restored from there after a halt/load. The OWL, which keeps track of writes in process to mirrored sets, is preserved in memory. If this structure is corrupted or lost during a halt/load, only mirrored sets with the DMS recovery option are brought online after the halt/load. Other mirrored sets are broken and only one member is brought online as a nonmirrored member.

The audit tables for offline disks are not preserved across halt/loads and the offline disks are released from the set when they are brought online. Packs having their audits applied and incomplete online creations are also released upon return to service.

Mirror Deallocation

You can deallocate mirrored copies from a mirrored set by using the MIRROR RELEASE system command. Different command forms allow you to release a designated disk, to release offline copies of a designated disk, and to release all copies of a designated disk. During normal system operations, you can release copies at any time.

Mirrors being audited, created, or having audits applied are automatically released from sets after a halt/load. Units receiving parity errors are also released if they are not the last online member of a mirrored set.

Whenever a disk is released from a mirrored set due to an operator command, a system error, or loss of the OWL, the label of the disk is invalidated, which requires you to enter an LB (Relabel Pack or Host Control Unit), a PG (Purge), or an RC (Reconfigure Disk) system command before you can use the disk again. If the label cannot be updated at the time, the label is invalidated when next seen by the system. This feature is necessary to prevent out-of-date members from replacing more recent copies of a unit.

Caution

If you ready out-of-date members on another mirroring system, they can be brought online. Because this could lead to data corruption, take special care to prevent this from occurring.

Transferring MCPs

When you create an alternate halt/load family with the system command *CM <file name> ON <family name>*, system mirror information is copied to the disk on which the alternate MCP is established. However, subsequent changes in system mirror

information is not automatically copied to the alternate halt/load family unless you establish the disk as a standby halt/load family by including a + *STANDBY* clause at the end of the CM command. If you do not use the + *STANDBY* clause, certain actions that involve mirrored sets will invalidate the accuracy of the mirror information table on that alternate halt/load family.

The actions that would invalidate the mirror information include the following:

- Using MIRROR system commands
- Bringing mirrored units online
- Releasing mirrored units due to exception conditions
- Auditing mirrored units due to exception conditions
- Closing a mirrored set

If any of these actions occur after you issue the CM command, then when a halt/load occurs from the alternate halt/load family, the system will come to a dead stop. To resolve this situation, you must halt/load back to the original halt/load unit.

A similar situation occurs if you attempt to transfer halt/load units between mirroring systems when an alternate halt/load family is moved to a different system that is using the OP + MIRRORING option. In such a case, you should reestablish the alternate halt/load family as a standby halt/load family before you attempt to halt/load. To reestablish the family as a standby halt/load family, issue the following command syntax for the disk on the new system:

```
CM + STANDBY ON <family name>
```

Otherwise, when you halt/load from that alternate halt/load family, the result will be a dead stop.

Precautions

The MIT on the halt/load family and the OWL in memory keep track of the status of the mirrored packs on a system. The MIT also contains records of packs that have been removed from a mirrored set, but whose labels could not be invalidated. These records are the only protection against such an outdated pack returning online. When you are changing halt/load units, if the new halt/load unit has an outdated MIT (that is, the timestamps on the MIT and the OWL do not match), this protection is removed. Data corruption can occur if you return such an outdated pack online. Therefore, keep the following recommendations in mind:

- Instead of maintaining a separate, backup halt/load unit, mirror the current halt/load unit. This ensures that the MIT will not be lost unless both the halt/load disk and its mirror are both lost.
- Establish one or more standby halt/load families.

- If you choose to maintain a separate backup halt/load family that is not an active standby halt/load family, update that unit with the CM (Change MCP) system command every time a pack is added or removed from the set of mirrored packs currently online. Update the backup, halt/load unit in the following cases:
  - When a pack is added to a mirrored set
  - When a pack is released from a mirrored set
  - When a pre-existing mirrored set is brought online
  - When a mirrored set is closed
  - After a halt/load
- If a dead stop occurs, halt/load back to the original halt/load unit. If this cannot be done (as when the original halt/load unit has failed), you must reinitialize the OWL in memory to match the MIT on the new halt/load unit. Note that you must take extreme care not to bring outdated packs online, or severe data corruption could result. You can remove the OWL from memory in one of the following ways:
  - For EMS systems, use UTILOADER to halt/load to the new halt/load unit
  - Use UTILOADER to warm start the system
  - Reinitialize memory by either powering the system off and on, or for host data unit (HDU) and resource management module (RMM) systems, remap memory at the maintenance station.

Section 11
Memory Disk Feature

Overview

Memory disk is the use of memory as if it were a disk unit. Memory disk provides file access with extremely high data-transfer rates and relatively little access time. You can declare one or two units of memory disk, each with up to 10 megawords of memory. You can use the units, with certain restrictions, in the same way that you use any other disk. The unit designations are retained across halt/loads. Files on the units are retained across most halt/loads; vulnerability of data is discussed later in this section. Memory disk is supported on all A Series systems.

Memory disk is established when you use the RECONFIGURE (Reconfigure System) system command to reconfigure to a group that contains a memory disk declaration. Instructions on how to make a memory disk declaration are provided in the *A Series System Configuration Guide*. The system allocates the requested amount of memory (or as much memory as is available) and fills memory disk with zeros. The location of the allocated memory is preserved on the halt/load unit so that the memory disk areas can be found after a halt/load. Each memory disk unit is reconfigured and brought online automatically if you supply the family name in the declaration. You must manually reconfigure unnamed memory disk units by using the RC (Reconfigure Disk) system command. You can use a memory disk family like any other family with the following exceptions: the memory disk cannot be the halt/load family, and it cannot contain a dumpdisk file.

Vulnerability of Data

Memory disk offers high-speed file access in a convenient and flexible manner, but you should be aware that memory is a volatile medium which is vulnerable to power failure and memory reconfiguration. Therefore, you should carefully choose the files that are to reside on memory disk. You will have to reconstruct those files after any system event that causes memory to be altered or corrupted.

Possible candidates that you might want to put on memory disk are read-only data files, code files, and temporary files. You should also consider using memory disk for the system command *DL SORT* and your data comm information files (assuming that you have a backup copy).

Creating a Memory Disk Unit

You can create a memory disk unit by modifying the configuration file for your system to add a new declaration in the MEMORY section of your system group. The declaration specifies a unit number and the amount of memory associated with the unit. You can optionally specify the family name or base unit of the family.

You can specify the amount of memory either as a number of words or as a number of pages. A page is 1,298 *kilowords,* which is 1,329,152 words because a kiloword is 1,024 words. In either case, the system uses an amount of memory that is the lower multiple of 512 kilowords—approximately half a page—nearest the amount that you specify.

For specific information on modifying the configuration file, refer to the *A Series System Configuration Guide.*

How Memory Disk Is Initialized

Memory disk is allocated above the system memory limit during memory establishment. A warning message is issued if a unit has insufficient usable memory. The amount and location of the memory are saved to allow the unit to be reestablished after a halt/load.

The memory disk units are established during peripheral initialization. Each unit is defined as though it were a physical head-per-track disk device with a maximum of 10 modules and with 2 switches per storage unit (SU). Each SU contains the equivalent of 34,952 disk sectors (approximately 1 megaword). Each switch contains the equivalent of 17,476 disk sectors (approximately 512 kilowords). The size of a memory disk is always an integer number of switches.

If you designated a family name for the memory disk unit in the configuration file, the system automatically reconfigures and readies the unit whenever necessary. This process happens on the next halt/load after you reconfigure your system or change the memory configuration of your system. If you did not specify a family name for the memory disk unit in the configuration file, you must manually reconfigure the memory disk unit whenever necessary by using the system command RC (Reconfigure Disk).

Memory Disk Halt/Load Recovery

All memory used by memory disk is reestablished from information saved on the halt/load disk. The memory areas marked for memory disk are not overwritten by the processors during memory test. If the current memory configuration does not match the saved configuration or if the memory disk areas have been overwritten, the system marks all memory disk areas as available, removes the saved information, and displays an error message. The memory disk configuration entry is then in its initial state. Memory is reallocated as described in the previous section, and the unit is reinitialized.

Memory Reconfiguration

Some memory reconfiguration commands cause memory disk to be reinitialized on the next halt/load. The system issues an appropriate request for response (RSVP) before the reconfiguration is performed. If you elect to proceed with the memory reconfiguration, you should be aware that the next scheduled or unscheduled halt/load will cause memory disk to be reinitialized.

I/O Handling

Statistics on memory disk I/O operations are handled in the same way as other I/O operations except that the processor time used by the data transfer procedure is charged to the task as processor time only. The bytes transferred are included in the MCP or user utilization counts as appropriate. The bytes are also counted into separate memory disk MCP and user subtotals. Because memory disk operations are synchronous (no I/O interrupt occurs), they do not contribute to either the IOFINISH or IOINTERRUPTS system counts.

Any irrecoverable error that occurs during a memory disk operation is reported in the logical result descriptor. The normal system action occurs for correctable memory parity errors, so they are not irrecoverable. All other errors are irrecoverable and are not retried. Memory parity errors and memory fail errors are reported as I/O parity errors but are also logged normally as memory errors. Missing pages are reported as not ready I/O results. The memory protect bit of the Input/Output Control Word (IOCW) is honored only if the MCP is compiled with the DIAGNOSTICS option. Memory protect errors are reported as descriptor errors.

Operational Restrictions

Memory disk units are pseudo head-per-track disk units. As such, they have the inherent operational characteristics of a head-per-track disk. The following restrictions apply to a memory disk unit:

- It cannot be the halt/load unit.
- It cannot contain a dumpdisk file.
- It cannot be designated in a FREE (Free Resource) system command.
- It cannot be a member of a disk family that contains nonmemory disk units.
- Peripheral test driver (PTD) tests cannot be run against it.
- It cannot be a mirrored unit.

Operational Considerations

When using memory disk, take the following considerations into account:

- A system dump does not dump the contents of memory disk.
- Reverting to a pre-Mark 3.6 MCP causes all memory disk pages to be marked as available and the units to disappear. Going forward to a Mark 3.6-or-later MCP again reinitializes memory disk. The changes to the configuration file for memory disk are transparent to pre-Mark 3.6 MCPs.

Explanation of Selected System Commands

The system commands that follow display information about memory. These commands are given in alphabetical order. For additional information about these system commands, refer to the *A Series System Commands Operations Reference Manual*.

CU (Core Usage)

This command displays only the amount of memory available for system use. The numbers do not include memory disk.

MM (Memory Module)

The following example shows the MM display on an A15 system using MCP/AS:

```
MM

 MSM Ø
   24M WORDS INUSE
 MEMORY USAGE:
   SYSTEM      12M WORDS
   MEMORY DISK 12M WORDS

 MSM STATUS
  SIM    MSUS ONLINE   MSUS SAVED     MSUS TO-BE-SAVED
   Ø     Ø-7           NONE           6-7
   1     2-7           Ø-1            2-3,6-7
   2     Ø-3,5-6       4,7            5-6
   3     Ø-1,3-4,6-7   5              6-7
```

OL (Display Label and Paths)

This command shows memory disk units as type MD:

```
OL DK3ØØ

   DK    3ØØ*MD  (READY: Ø-1) [ØØØ3ØØ] #1 MEMDISK (2)
   CREATED ON: Ø6/19/86 AT 18:48:31
   DLP       STATUS
   * NO PATHS *
```

PER (Peripheral Status)

This command shows memory disk units as type MD:

```
PER DK

    ----- DK STATUS -----
     3ØØ*MD (READY: Ø-1) [ØØØ3ØØ] #1 MEMDISK (2)
```

U (Utilization)

This command includes memory disk I/O subtotals in its display. An additional component of the I/O portion of the display identifies those I/O requests issued to memory disk as subtotals of user and MCP I/O requests.

Section 12
Recovery

The disk subsystem can encounter three types of problems:

- Hardware problems, such as a broken disk drive or a damaged or destroyed disk
- Operator error, such as the accidental reconfiguring of a disk or the accidental removal of files
- Directory software errors, such as directory I/O errors, directory data corruption, or MCP program errors

Although these problems are very rare, it is important that your installation be able to recognize and differentiate these problems so that they can be resolved in the shortest amount of time with little or no loss of data. This section explains the following error recovery procedures:

- How to isolate defective sectors on a disk
- What to do if base packs or continuation packs are damaged or destroyed
- When disks can be moved from one drive to another to eliminate errors
- When data can be moved from one disk to another to eliminate errors
- What to do if directory errors are encountered
- What to do if errors are encountered during family rebuilds
- How to fix a defective archive directory

Before the system can use the data or directories on a disk, you must complete the following procedure without encountering any errors:

1. Place the disk on a drive.
2. Turn on the drive.
3. Acquire (if necessary) and ready the disk drive on which the disk is mounted.

The system then has to read the disk label and, if the disk contains a flat directory, read the directory file header.

Processing of the disk stops if any of these steps fail. Usually, hardware I/O errors cause these failures, and these errors must be corrected before the data on the disk can be used. Sometimes you can correct an I/O error by closing the disk and then readying it again. If this process does not solve the problem, your only recourse is to move the disk to another disk drive.

After the disk label and flat directory header have been successfully read, the system provides various techniques for overcoming errors. You can isolate defective sectors, replace the disk, or move the data from the disk to another disk.

Isolating Defective Sectors

When an I/O error occurs while the system is using or trying to use a disk file, the entire disk is not necessarily damaged. A group of sectors on the disk might be defective, and it is possible to isolate these bad sectors so that the rest of the disk can still be used.

When the system encounters an I/O error, it automatically attempts the I/O operation again and places an entry in SYSTEM/SUMLOG that describes if the error was corrected in the retry or if the error was not corrected and the I/O operation was discontinued. These log entries and the I/O error messages that are displayed on the ODT can be used to determine if the problem is caused by a few defective sectors or if the entire disk is damaged.

If the log entry or the I/O error messages repeatedly designate a certain range of sectors, the damage is probably limited to those sectors. Also, if the I/O error affects only a few files, but the rest of the files on the disk are accessed properly, that is an indication that the problem was caused by a few defective sectors.

You can use the SCAN (Scan Disk or Pack Volume) system command to read a disk and record any defective sectors that were encountered during an attempt to read a file on the disk. The SCAN operation is a time-consuming process, and should be done at night or another time when system usage is minimal.

At the end of mirror creation, errors found on sectors not spanned by BADDISK areas are reported in a file on the halt/load family named *CREATE/<source name>/FAMILYINDEX <source family index>/SECTORSINERROR*. This report provides a quick alternative to SCAN in locating defective sectors discovered during the creation of a mirrored set.

After you have determined which sectors are defective, use the RES (Reserve) system command to transfer the data stored on the defective sectors to other sectors. Use the AS BADDISK clause of the RES command so that the system marks the defective sectors so they cannot be used again.

Refer to the *A Series System Commands Operations Reference Manual* for more information about the SCAN and RES commands.

Dealing with Damaged or Destroyed Disks

A disk can be considered damaged if it experiences irrecoverable I/O errors or if the directory is corrupted. A disk is considered destroyed if it is physically broken. There are different ways to resolve the problem of a damaged or destroyed disk, depending on whether the disk is a base pack or a continuation pack. This subsection discusses the techniques for replacing both types of disks.

Replacing a Base Pack

If the base pack is destroyed and the flat directory is not duplicated on another family member, you must use the RC (Reconfigure Disk) system command to reconfigure a new family and then copy all the required files back onto the family from the latest backup

tapes. You cannot reconfigure a new base pack in place of the old one, even if the same serial number is used, because the flat directory on the old base pack is lost.

The system marks all the continuation packs in a family with the date and time that the base pack was reconfigured. When you reconfigure a new base pack with the RC command, there is no link between the new base pack and the old continuation packs. If the system did not take this precaution, continuation packs from an old family would get confused with continuation packs for the new family. You can check the creation date and time of a disk family with the OL (Display Labels and Paths) system command. The OL command syntax is as follows:

```
OL PK <unit number>
```

The variable <unit number> is the unit number of the disk drive on which the disk is mounted.

If you are running on a cataloging system and need to reconfigure a new base pack, you must take special steps to avoid losing all the catalog information for the old family. The system asks if the new family should inherit all the backup information of the old family by displaying the following message on the ODT:

```
PK <unit number> OK TO RE-ENTER INTO VOLUME LIBRARY
```

Enter the reply *OK* to retain the catalog information. Refer to "Replacing a Damaged Volumed Disk" in Section 5 for more information about this subject.

Replacing a Continuation Pack

If a continuation pack is destroyed, you have more choices. The base pack with the flat directory is still intact, so files on the base pack and other continuation packs are still usable. When programs try to access files that are partially or totally stored on the missing continuation pack, the system responds with the following RSVP message:

```
REQ <family name> <serial number>
```

You then must discontinue those programs by using the DS (Discontinue) system command.

You can create a substitute continuation pack by using the RC (Reconfigure Disk) system command. The RC command syntax is as follows:

```
RC PK<unit number> BP = <serial number> FAMILYINDEX = <family
index number>
```

The variable <unit number> is the unit number of the disk drive the substitute disk is mounted on, the variable <serial number> is the serial number of the base pack, and the variable <family index number> is the family index number of the missing disk.

The system then replaces the old continuation pack with the new one and removes all the files in the entire family that had portions stored on the old continuation pack. To retain files that had portions stored on the missing continuation pack, add the KEEP

clause to the RC command. The system does not completely remove the files, but marks them as having some areas missing. Programs then can use some of the data in the files, but not data from the areas that were on the missing continuation pack. If a program tries to access data from the areas that were on the missing continuation pack, an I/O error occurs.

Moving Disks to Another Disk Drive

When a disk drive breaks down or appears to be receiving many I/O errors, it might be possible to correct the problem by moving the disk to another drive. This technique applies only to removable media such as Model 677 disks; you cannot move fixed media such as Model 207 disks.

Caution

Before moving a disk from one drive to another, especially a disk that has experienced errors, have a Unisys field engineer inspect the disk to make sure that the surfaces of the disk have not been marred or scratched. A scratched disk can in turn scratch the disk drive read/write heads. Scratched read/write heads can in turn scratch other disks mounted on them, and so on. Your installation might want to keep written records of the drives that disks have been mounted on, by time and date. When a scratched disk surface or a scratched disk read/write head is discovered, you will know which other disks and drives to inspect for damage.

If you decide to move a disk to avoid I/O errors, the first thing you must do is to locate an available disk drive. This unit must be acquired and not saved, not reserved, and not in-use by another disk. If the disk that you want to move is not in-use, all you have to do is power off the disk and move it to the other drive. If the disk that you want to move is in-use, you must enter the MOVE (Move Job/Pack) system command with the following syntax:

```
MOVE PK <source unit number> TO PK <destination unit number>
```

The variable <source unit number> is the unit number of the original disk drive and the variable <destination unit number> is the unit number of the destination disk drive. The system does some checking and then tells you to manually move the disk. The following is an example of the MOVE command:

```
MOVE PK 67 TO PK 143
```

On systems other than the A 12 and A 15, you cannot move some families (such as the family on which the JOBDESC file is stored) by using this technique. On A 12 and A 15 systems, you can move any disk by using the MOVE command.

Sometimes the I/O error problem occurs so many times that a halt/load is necessary before you can move the disk to an operable disk drive. However, you may be able to

avoid the halt/load by performing the following steps. This process is not recommended for A 12 and A 15 systems.

1. Halt the processors.
2. Manually move the disk.
3. Start the processors when the disk is ready.

This technique does not always work. If the system does not start running again, proceed with a halt/load.

Moving Data to Another Disk

Usually when a disk experiences problems, it is not totally unusable. If a particular disk in a family produces a large number of I/O problems, you can move the data to another disk by one of two techniques.

Using the RES Command

If the disk is a continuation pack of a multidisk family, you can reserve the disk with the RES (Reserve) system command with the following syntax:

```
RES PK <unit number> AS BADDISK
```

The system copies all the data from the disk to other members in the family. After the reserve operation is completed, you can perform maintenance operations on the disk. For instance, you can use the PG (Purge) system command. If the system encounters parity errors during the RES operation, it terminates the operation and displays a message on the ODT that designates which addresses encountered the error. You must then decide whether to remove the affected files or use the COPYERRORS clause of the RES command. COPYERRORS causes the RES operation to copy all the data on the disk, including data that has parity errors. The data that has parity errors is still invalid and cannot be used.

For more information, refer to "Using the RES, XD, and SQUASH Commands" in Section 2.

Using the REPLACE Command

Use the REPLACE (Replace Disk or Pack Volume) system command instead of the RES command in the following situations:

- If the disk is a base pack or if it contains a large amount of data. In this case, the RES operation can be a lengthy process. Also, if the disk contains a large amount of data, it can be difficult for the RES operation to find sufficient space for the files on the other members of the family.
- If you know that the disk contains portions of files that were assigned to the disk with the FAMILYINDEX file attribute. The RES operation does not permit files to be moved from a disk if the files were assigned to that family member with the FAMILYINDEX attribute.

To use the REPLACE command, obtain an available disk of the same model (677, for example) or a different model disk that contains a larger number of sectors, and enter the following command syntax:

```
REPLACE PK <source unit number> ONTO
PK <destination unit number>
```

The system then copies all the files and label information from the original disk to the new disk. The new disk receives the serial number and family name of the old disk. When the copying is complete, the system erases the label on the old disk so that it does not become confused with the new disk.

Caution

Do not use the REPLACE system command to make backup copies of disks because the command erases the label of the source disk.

Directory Error Recovery

Disk directories are very important. If something goes wrong with a directory, it is possible to lose some or all files referenced by that directory. The directories are designed so that errors in one part of the directory do not adversely affect other parts of the directory. The system constantly monitors the directories for discrepancies. When an error is detected, the system automatically attempts to eliminate the problem. The following system features help to ensure the integrity of the directories:

Standard Disk I/O Error Recovery

The standard I/O subsystem attempts to recover from all failed I/O operations, including directory I/O errors. Only I/O errors that are not corrected by these attempts can cause problems.

Directory Record Integrity Tests

The MCP tests all directory records before they are used. Each record must pass tests such as HDRMARKER, HDRLOCATION, and CHECKSUM. If a record fails one of

these tests, the directory subsystem invokes one or more of the following procedures: error reporting, recovery, or termination.

Automatic ERRORHANDLER Family Rebuilds

These rebuilds are invoked to build a new file access structure table (FAST). Family rebuilds solve two types of problems. First, certain types of bad records in the flat directory or catalog are bypassed during the rebuild so that these records are not referenced by the rebuilt FAST and do not cause problems. Second, certain corrupted FAST records are discarded during the rebuild. Refer to "The Family Rebuild Process" later in this section for more information on this subject.

Directory Duplication

Duplicate directories on multidisk families can be used to solve most directory I/O errors and problems caused by directory data corruption. Duplicate directories also prevent directory loss if a disk is physically destroyed. Duplicate directories require a modest amount of additional disk space and require more I/O operations because more than one copy of the directory must be updated.

Disk directories are stored as disk files. As with any disk file, they can experience problems. The flat directory and the catalog can have three different, but interrelated, types of problems: directory I/O errors, directory data corruption, and MCP program errors. These errors, which are very infrequent, have the following characteristics.

If a disk I/O error message immediately precedes or follows a directory error, then you can infer that the I/O error might be the root of the directory error. You can also make finer distinctions. If the I/O error occurred during a write operation, the message indicates that a file header that was just added, removed, or changed in the directory might not be usable the next time it is referenced. If the error occurred during a read operation, the message indicates that the system was attempting to access a directory record (such as a disk file header) and that directory record is unreadable. Repeated references to that same file presumably would cause repeated read errors.

Some I/O errors can be easily corrected. For example, you may be able to solve a NOT READY or WRITE LOCKOUT error by turning the proper switch to the proper setting. On the other hand, parity errors on read operations usually are not correctable. In fact, the system only reports a parity error if several attempts to perform the read operation fail. Correction of parity errors depends on the exact nature of the error. If the data truly has a parity error, then that directory record is lost. But if the parity error is caused by a faulty disk drive or a disk drive controller malfunction, fixing the hardware restores access to the directory record.

It is difficult to distinguish directory data corruption from MCP errors. A typical directory data corruption error might cause a CHECKSUM, HDRMARKER, or HDRLOCATION error message. Note that all read errors also appear to the MCP directory subroutines as directory data corruption; read errors also cause error messages for CHECKSUM, HDRMARKER, HDRLOCATION, and so on. On the other hand, a typical MCP error results in a memory dump by FILEHANDLER. These are

only tendencies, however. MCP errors can cause HDRLOCATION errors, and data corruption can cause FILEHANDLER memory dumps.

When the system detects a directory error, it reports the problem with one or more messages. The system then often attempts some type of automatic error recovery. The system usually reports the progress or failure of the recovery with additional messages. These various messages are clues as to what happened. You should try to answer the following list of questions after a directory error occurs. The answers to these questions can help you decide what further corrective actions you might need to take.

- Did the system read the disk label and open the base pack directory, or did the system issue an error message to indicate that the disk cannot be used? If such an error message appears, there is a fundamental problem with the disk and not a problem with individual files on the disk.
- Does the problem appear to affect only one or a few programs or files? Does it appear to affect every file on a family? Does it appear to affect every file on the system? These questions can help you isolate the problem. If only one or a few programs or files encounter errors, the problem usually is within these programs or files. If only one family encounters errors, the problem is within that family.
- Does the problem appear to be in the flat directory on a given family, or does it appear to be in the catalog or access structure? Such problems are usually caused by I/O errors. If the word FLAT appears in the error message, then it indicates that the flat directory may have a problem. If the words PAST, FAST, CATALOG, VAST, or VOLLIB (Volume Library) appear, it almost always means there is a problem in the catalog or the access structure.
- Does the problem appear to be an I/O error, or does it appear to be a case of data corruption or an MCP error? If it appears to be an I/O error, try to correct the problem as discussed above. If it appears to be a case of data corruption or an MCP error, contact your Unisys field engineer or file a User Communication Form (UCF) with Unisys.
- Does the problem appear to be intermittent, or has the problem been there for a long time? Does the same kind of problem crop up occasionally for different files, or does the problem appear only for a particular test case? If you can isolate the test case, you can show it to your Unisys field engineer.
- Does the problem go away after a family rebuild? If it does, then the rebuild probably took care of the problem.

Be sure to locate the original error message, in SYSTEM/SUMLOG if necessary. Often one error triggers several error reports. The first error message can be the best clue. The LOGANALYZER utility often is helpful in solving I/O error problems because all I/O errors and an audit of any retry attempts are recorded in SYSTEM/SUMLOG. You can use LOGANALYZER to examine the errors of a particular disk by entering the following syntax:

```
LOG MAINT PK <unit number>
```

The variable <unit number> is the unit number of the disk drive on which the disk is mounted. LOGANALYZER then prints out a list of all I/O error retries and messages that have occurred for that disk drive since the log was initiated. These I/O error

messages can help you identify the problem, and you can then contact your Unisys field engineer if necessary.

The Family Rebuild Process

For each disk family, the system uses a special index structure called the file access structure table (FAST) to access disk files. When a family is first brought online, the system builds a FAST for it. Each time files are added, changed, or removed from the family, the system updates the FAST accordingly; sometimes the system rebuilds the FAST when disk errors occur. This discussion describes the family rebuild process, the effects it can have on the FAST, and how you can respond to errors in the family rebuild process.

There are three types of family rebuilds: rebuilds when a base pack is first placed online, rebuilds invoked by the RB (Rebuild Access) ODT command, and rebuilds for error recovery. On cataloging systems, catalog rebuilds also occur during error recovery. Family and catalog rebuilds normally do not occur on a cataloging system when a volumed disk is placed online.

Family Rebuilds for a New Base Pack or Halt/Load

The first type of family rebuild occurs when a base pack is first placed online or at halt/load time. Regardless of whether or not a rebuild is needed, the system reads the entire flat directory on the disk to determine what sectors of the family are already allocated to files and what sectors of the family are available for new files. If a rebuild is necessary, then the FAST is rebuilt at the same time.

Responding to Errors

The MCP issues error messages if any errors are detected during this reading of the flat directory or rebuild of the FAST. Immediately after the first such error message, the MCP displays the following message on the ODT:

```
PKuu OK TO ERASE BAD RECORDS? (SYSTEMDIRECTORY/nnn ON packname)
```

This message indicates that the MCP has just detected an error; there might be more errors as the rebuild progresses. The MCP is asking for permission to erase all flat directory records that caused errors. You can enter one of the following three replies.

- DS

  If you enter the DS (Discontinue) system command, processing of the flat directory halts and the system marks the disk as being offline.

- OF

  The safest reply to the message is to use the OF (Optional File) system command. This reply causes the system to process the rest of the directory, but not to erase the bad records.

The advantage of entering OF is that it gives the system a chance to process the rest of the directory without accidentally erasing it. For instance, if the cause of the original problem was that the disk was being read on a faulty disk drive, then erasing directory records could erase good records from the directory.

When you enter OF, the system marks the disk as not having any available space for new files. This feature prevents a new file from accidentally being allocated in areas of the disk that overlap the areas in use by the files that apparently were bad. If you need to add records or files to the disk, you will have to remove existing files to make room for the new records or files.

After the system reports one bad record in the flat directory, the system often reports a few subsequent consecutive directory records as also being bad. Directory records are variable in length and several can be accessed together in one block, so additional *bad* records are often all part of the original bad record. If only a few extra bad records are reported, you usually can assume that there is actually just one bad record.

- OK

  You can enter the OK (Resume) system command to erase corrupted directory records. This reply could lead to the loss of some files on the disk. Use this reply unless you suspect that the disk I/O hardware is defective and might be causing the problem. If you enter OK, the system proceeds with a family rebuild and then marks the disk online.

Terminating the Rebuild

If the rebuild is being done while readying a disk or halt/loading, you can terminate the rebuild by entering *CL PK <unit number>*. The system stops the rebuild and marks the system as being offline. This can be a useful option if the disk becomes not ready during the rebuild or you decide the wrong disk was readied.

Family Rebuilds Initiated by Your Installation

This type of rebuild occurs when you use the RB (Rebuild Access) system command to build a new access structure. You can use the RB command when you think that the access structure is corrupt or does not actually describe the files on a disk.

This type of rebuild reconstructs the FAST and the AAST (for the archive directory of the family); it does not build an available disk table, does not ask the operator questions, and does not erase flat directory records (good or bad). You cannot terminate this rebuild by issuing the CL (Clear) system command.

Family Rebuilds for Directory Error Recovery

This type of rebuild occurs when the directory error recovery procedure ERRORHANDLER is invoked. ERRORHANDLER is invoked automatically when an error is detected while accessing the directory of an online family.

This rebuild only reconstructs the FAST; it does not build an available disk table, does not ask the operator questions, and does not erase flat directory records (good or bad). You cannot terminate this rebuild by issuing the CL (Clear) system command.

Family Rebuild Errors

When problems occur during a family rebuild process, the system reports them through error messages. Errors reported during one rebuild process can lead to other problems later. For example, files might be reported as missing when the disk is used, or there might be conflicting or duplicate file names when a subsequent rebuild process is executed on the same family. If the rebuild process reports readable records as unreadable because of software or hardware errors, those records might again be readable at some later time. The rebuild process does not erase file records that appear, even temporarily, to be unreadable, unless you respond to the "OK TO ERASE BAD RECORDS?" system message with an OK or a DS reply.

When the family rebuild process has finished executing, it is possible for other programs to add files to the family directory. These files can have the same titles as the files in which unreadable records were found during the rebuild process. Because the system is not aware that the files have the same titles, the unreadable records are not removed. When another rebuild process is executed, these formerly unreadable records might be read successfully. During this subsequent rebuild, the system discovers the duplication of file titles and issues the "DUPLICATE FILE TITLE DURING REBUILD" error message.

It is important to find and correct the root cause of any rebuild errors as soon as possible, perhaps by repairing the disk unit. When you have found and corrected the cause of the error, immediately issue the following system command:

```
RB ON <family name>
```

This command enables the system to build an accurate list of the names of the files that reside on the disk family, before too many duplicate file names appear.

Working with Damaged or Incorrect Archive Directories

The MCP maintains a separate archive directory on the catalog family for each online disk family at your installation. Each archive directory consists mainly of records that identify the names and locations of files that have been copied to backup tapes through archive subsystem operations. Because the archive directory retains records of up to four backup operations for each file copied to tape, you might find references to old tapes and earlier generations of files in this directory.

Like any directory or file, the archive directory can be damaged. For example, records in an online archive directory can be corrupted by hardware or system software errors. Moreover, an archive directory might be unavailable to you. This sometimes occurs during halt/load procedures or when the *RY* (Ready) system command is run to ready a disk.

If you find that an archive directory is damaged, or that some of its records have been corrupted or inadvertently removed, you can follow one of the methods described below.

Note: The effectiveness of these methods relies on regularly performed backup procedures. If your backup directories do not contain current data, these procedures might replace your damaged or missing directory with an out-of-date version.

Restoring Archive Directories

If you find that a significant portion of an archive directory is damaged or missing, you can use a backup copy of the directory as a replacement for the damaged or missing one. To restore an archive directory to disk, perform the following procedure:

1. Use the WFL *COPY* statement to copy the backup version of the archive directory to the catalog family. Be sure to include the FAMILYINDEX clause in the COPY command syntax; this clause ensures that the entire directory is placed on a single pack of the catalog family. The following example shows a COPY statement, in which the DL catalog family is named CPACK:

```
COPY & COMPARE ARCBACKUP/DATABANK/ØØ1 FROM
       ARCBACKUP TO CPACK (FAMILYINDEX=1)
```

2. When the system has finished executing the COPY statement, you can use the ODT system command ARCREPLACE (Archive Replace). This command changes the name of the defective directory and activates the backup copy in its place. An example of the ARCREPLACE command follows:

```
ARCREPLACE DATABANK ARCBACKUP/DATABANK/ØØ1 ON CPACK
```

These steps change the name of the active archive directory for the family DATABANK to OLD/SYSTEM/ARCHIVE/DATABANK, and the name of the replacement directory from ARCBACKUP/DATABANK/001 to SYSTEM/ARCHIVE/DATABANK/001. The file named SYSTEM/ARCHIVE/DATABANK/001 is then opened for use as the archive directory for the disk family.

Restoring Damaged Archive Directory Records

If you find that a few archive records are damaged or missing but the directory itself is largely intact, you can restore those records with the aid of a DCALGOL utility program. You can code a program that uses GETSTATUS and SETSTATUS calls to delete damaged or incorrect records and replace missing records.

The following DCALGOL program provides an example of how such code might look. This program restores all archived records that have been copied through a GETSTATUS call (TYPE 3 and SUBTYPE 12), and saved in the following file name:

```
ARCSAVE/PACK ON DISK
```

This program example restores archive records that have been copied and saved as described above, for the files associated with the usercode UCDEPT. You might use a

program like this one if, for example, someone in your installation inadvertently purges all archive records associated with a particular usercode, and you had preserved a copy of the records as described above.

```
BEGIN
  ARRAY ARY [Ø:3ØØØ];     % SETSTATUS PARAMETER ARRAY
    BOOLEAN RSLT;         % SETSTATUS RESULT

   % INPUT DISK FILE-- PRODUCED BY A CALL ON
   % GETSTATUS (3 & 12 [15:8], ....);
  FILE D (KIND=DISK, NEWFILE=FALSE, TITLE= "ARCSAVE/PACK.",
          FILETYPE=1, INTMODE=SINGLE, DEPENDENTSPECS=TRUE);
   ARRAY  BUF [Ø:2Ø49];    % INPUT RECORD
     DEFINE    % ABBREVIATED LAYOUT OF ARCHIVE RECORDS
         % EVERY LINK WORD IS LAYED OUT AS FOLLOWS
         ARLINKEDF = [47:1] #,    % 1 MEANS WORD HAS ARSIZEF & ARLINKF
         ARSIZEF  = [23:12] #,    %   LENGTH OF EACH ENTRY (WORDS)
         ARLINKF  = [11:12] #,    %   INDEX OF STUFF IN VARIABLE PART
       ARTITLEX = 6 #;                % LINK TO FILE NAME
     POINTER PSFN;          % POINTER TO FILENAME IN "BUF"


  % LOOP READING RECORDS FROM BACKUP OF ARCHIVE DIRECTORY
  WHILE NOT READ (D, 2Ø49, BUF) DO
    BEGIN
       % NOTE, INDEXES INTO "BUF" ARE INCREMENTED BY 1 BECAUSE FIRST
       % WORD OF RECORD READ FROM "D" FILE IS FILETYPE 1 LENGTH INFO.
      PSFN := POINTER (BUF [1 +  BUF [1+ARTITLEX].ARLINKF]);
      IF REAL (PSFN+1, 1).[1:2] = 3 THEN     % HAS USERCODE
        IF PSFN+3 = 48"Ø6" "UCDEPT" THEN      % USERCODE=UCDEPT
          BEGIN
            ARY [Ø] := 6;
            ARY [1].[38:6] := 3;
            REPLACE POINTER (ARY [2]) BY 48"Ø4" "PACK";   % FAMILY NAME
            REPLACE POINTER (ARY [6]) BY
                      POINTER (BUF [1]) FOR BUF [Ø] WORDS;
            RSLT := SETSTATUS (3, 2, Ø, ARY);
            IF RSLT THEN
              DISPLAY ("SETSTATUS ERROR RESTORING ARCHIVE RECORD");
          END;
    END OF READING FILE LOOP;
END.
```

To find detailed discussions of the available kinds of GETSTATUS and SETSTATUS calls, refer to the *A Series GETSTATUS/SETSTATUS Programming Reference Manual.*

Appendix A
Layout of Archive Directory Records

This appendix presents the layout and content of the data records in the archive directory. This information is presented here to help you use the GETSTATUS and SETSTATUS intrinsics more effectively, and to show you the form in which information is passed to the selector procedure when you issue WFL *ARCHIVE* statements.

Archive records are used by the following system interfaces:

- When you retrieve archive records with certain GETSTATUS directory calls.
- When you add a record to an archive directory, using existing archive directory records, with a SETSTATUS call.
- When you display archive records by using the FILEDATA system utility with an ARCHIVEINFO request that uses the *RAWHEADERS* SYSTEM/FILEDATA task request.
- When the archive subsystem passes archive records as parameters to the selector procedure of the archive support library. This process occurs when you issue a WFL archive statement.

Layout of the Archive Data Records

The layout shown in this section presents the overall layout of the data records in the archive directories. In the provided example, comments are included to clarify the meaning of particular items in the archive records. All comments are preceded by a percent sign (%).

Note: *The SYMBOL/ARCHIVESUPPORT file contains the DEFINE definitions that are shown in this layout. If you need to, you can copy them from that file.*

The definitions under DEFINE describe the words, fields, and values used in this structure. In general, the archive records share the following characteristics:

- The length of each record is variable. This characteristic enables each record to contain variable length items efficiently. For example, file titles can be from 5 to 255 characters long, while tape serial number lists can be from 1 to 256 words long. In either case, format changes to the record are not required.
- Each record can refer to as many as four backup tape sets for the same file.
- The records are designed to enable future additions of defined words without causing previous Mark release systems to fail.

```
      % ALL WORDS DEFINED IN THIS STRUCTURE BEGIN WITH "AR" AND END WITH
      % "X"; ALL FIELDS HEREIN BEGIN WITH "AR" AND END WITH "F"; AND
      % ALL VALUES HEREIN BEGIN WITH "AR' AND END WITH "V".

    DEFINE
     ARØX = Ø #,                      % MARK, HDRLOCATIONF, HDRBLOCKLENGTHF
                                      %  HDRBLOCKLENGTHF  & HDRLOCATIONF
     ARTYPEX = 1 #,                   % DETERMINES STRUCTURE OF RECORD
       ARIDF = [47:24] #,             % DISTINGUISH ARCHIVE DATA BLOCK
                                      % FROM ARCHIVE "FAST" BLOCK
        ARIDV = 'ARC' #,              % "BLOCKNOLENGTH" NEVER EQUALS THIS
                                      %   (Ø FOR DISK AND SCRATCH TAPES)
       ARLEVELF = [23:8] #,           % MCP RELEASE LEVEL FOR THIS ARCHIVE
                                      %    DIRECTORY DATA BLOCK STRUCTURE
         ARLEVELV = 39 #,
       ARVARF = [11:12] #,            % INDEX TO START OF VARIABLE LENGTH
                                      %    STUFF;  RECORDS PRODUCED BY
                                      %    DIFFERENT MCP LEVELS MAY VARY
                                      %    THIS.
     ARTIMESTAMPX = 2 #,              % DATE & TIME WHEN THIS RECORD WAS
                                      %   LAST CHANGED OR UPDATED.
     ARSRX = 3 #,
       ARSRF = [11:12] #,             % INDEX TO LINKED LIST OF SUBRECORDS;
                                      % THERE IS ONE SUBRECORD PER
                                      % GENERATION TRACKED. (IN THE 3.9
                                      % RELEASE ONLY ONE GENERATION IS
                                      % TRACKED.
        % SEE BELOW FOR LAYOUT OF THE SUBRECORDS
     ARCHA1X = 4 #,                   %         R F E
     ARCHA2X = 5 #,                   %         R F E

  % END OF ORIGINAL 3.9 FIXED STUFF.  NEXT THERE ARE THE SPECIAL WORDS.
  % THERE ARE TWO KINDS OF SPECIAL WORDS: THOSE WITH ARLINKEDF ON, AND
  % THOSE WITH ARLINKEDF OFF.  WORDS WITH ARLINKEDF ON ARE LINKS TO
  % STUFF IN THE "VARIABLE LENGTH AREA" THAT STARTS AT WORD NUMBER
  % "XXX [ARTYPEX].ARVARF".  WORDS WITH ARLINKEDF ON ARE LAYED OUT
  % THUS: Ø & firstwordno ARLINKF & wordcount ARSIZEF & 1 ARLINKEDF.
  % WORDS WITH ARLINKEDF OFF ARE NEW BUCKETS THAT CAN BE INVENTED ON
  % POST 3.9 MCPS TO HOLD NEW BITS (EACH NEW WORD WOULD HAVE 46 NEW
  % BITS AVAILABLE FOR DEFINITION).  FOR EXAMPLE, IF ON THE 4.1 RELEASE
  % YOU WANTED TO INVENT SOME NEW 1Ø BYTE ITEM YOU COULD ALLOCATE
  % TWO WORDS FOR IT (INCREASE ARVARST DEFINE TO BY 2), AND STORE THE
  % TEN BYTES IN THOSE TWO WORDS (BEING CAREFUL TO LEAVE BIT 47 -- THE
  % ARLINKEDF BIT = Ø IN BOTH WORDS).

  % WE NEED THE ARLINKEDF BIT SO THAT AN OLD MCP CAN SLIDE STUFF AROUND
  % IN THE "ARVARSTX" AREA OF ANY RECORD AND BE ABLE TO ADJUST ALL
  % ARLINKF INDEXES CORRECTLY.  THE OLD MCP DOESN'T NEED TO KNOW WHAT'S
  % POINTED TO BY NEW ARLINKEDF/ARLINKF/ARSIZE WORDS; IT ONLY NEEDS TO
  % KNOW THAT ANY WORD WITH THE ARLINKEDFBIT ON MAY NEED ADJUSTMENTS
  % TO ITS ARLINKF VALUE.
```

```
  % EVERY LINK WORD IS LAYED OUT AS FOLLOWS:
  ARLINKEDF = [47:1] #,       % 1 MEANS WORD HAS ARSIZEF & ARLINKF FOR
  ARSIZEF  = [23:12] #,       %   LENGTH OF EACH ENTRY (WORDS)
  ARLINKF  = [11:12] #,       %   INDEX OF STUFF IN VARIABLE PART

ARTITLEX = 6 #,               % LINK TO FILE NAME

% ON FUTURE RELEASES NEW ITEMS & LINKS CAN BE DEFINED HERE.  NOTICE
% THAT IN THE INITIAL 3.9 RELEASE THE MCP DOES NOT HAVE TO TEST FOR
% THE PRESENCE OF NEW WORDS.  FOR EXAMPLE, WE KNOW THAT ALL RECORDS
% HAVE AN ARTITLEX SLOT BECAUSE THE INITIAL 3.9 RELEASE HAD THAT
% DEFINE.  BUT FOR ALL POST 3.9 INTRODUCED WORDS THE MCP MUST BE
% CAREFUL TO TEST THAT THE RECORD IN HAND HAS THE NEW WORD BEFORE
% IT CAN BE ACCESSED:  IF XXX [ARTYPEX].ARVARSTF GTR ARnewwordX THEN
% RECORD HAS THE WORD...

ARVARSTX39 = 7 #,     % ALL LEVELS OF ARCIOV RECORDS HAVE AT LEAST
                      % THIS MUCH.
ARVARSTX = 7 #,
  % START OF VARIABLE LENGTH STUFF POINTED TO BY LINK WORDS.  THE
  % DATA IN THIS AREA MUST APPEAR IN THE SAME ORDER AS THE LINK
  % WORDS THAT POINT TO IT AND THE DATA ITEMS MUST BE CONTIGUOUS
  % (NO INTERVENING HOLES).  THESE RULES MAKE VALIDITY CHECKING BY
  % "VALIDITY OF ARHANDLER" AND SIZE CHANGES BY "STRETCH OF ARHANDLER"
  % EASIER.  IMMEDIATELY AFTER THE VARIABLE LENGTH STUFF FOR THE MAIN
  % "AR..." RECORD THERE APPEAR THE "ARG..." SUBRECORDS.  THERE IS
  % ALWAYS AT LEAST ON "ARG..." SUBRECORD:
% LAYOUT OF "ARG..." SUBRECORDS LINKED FROM "X [ARSRX].ARSRF" AND
% "X [ARGX].ARGNEXTF".  ONE SUBRECORD PER GENERATION:
 ARGX = Ø #,
   ARGNEXTF    = [23:12] #,    % INDEX OF NEXT SUBRECORD OR Ø
   ARGVARF     = [11:12] #,    % INDEX TO START OF VARIABLE LENGTH STUFF
 ARG1X = 1 #,
   ARGFKNDF = [47:12] #,       % FILEKIND
   ARGSECTYPEF = [35:8] #,     % SECURITYTYPE
   ARGSECUSEF  = [27:8] #,     % SECURITYUSE
 ARG2X = 2 #,
   ARGORGF = [47:8] #,         % FILE ORGANIZATION (E.G. KEYEDIO)
   ARGSAVEFACTORF = [19:2Ø] #, % SAVEFACTOR
 ARGTIMESTAMPX = 3 #,          % TIMESTAMP OR FILE
 ARGGENX = 4 #,
   % % GENEALOGYF  =[37:22]#   % (CYCLE + VERSION).
   % %   CYCLE     =[37:14]#   %   CYCLE ATTRIBUTE.
   % %   GENVERSN  =[23:8]#    %   VERSION ATTRIBUTE.
 ARGSIZEX = 5 #,               % SIZE OF FILE IN MEGA-BYTES
 ARGCREATEX = 6 #,             % CREATION DATE AND TIME
 ARGACCESSX = 7 #,             % ACCESS DATE AND TIME
```

```
ARGCHAX = 8 #,              % R F E
ARGBKSX = 9 #,              % FOUR BACKUP INFO WORDS; AT LEAST
                            %   ONE BACKUP MUST EXIST OR SUBRECORD
                            %   (& RECORD) SHOULD BE REMOVED FROM
                            %   SYSTEM/ARCHIVE/... DIRECTORY.
  ARGREELF = [47:8] #,     % STARTING REEL NO. OF BACKUP

  ARGTYPE = [37:6] #,      % TYPE OF BACKUP
    ARGEMPTYV = Ø #,        %   BACKUP ENTRY NOT USED (OR DELETED)
    ARGFULLV = 1 #,         %   ARCHIVE FULL ...
    ARGDIFFV = 2 #,         %   ARCHIVE DIFFERENTIAL ...
    ARGINCRV = 3 #,         %   ARCHIVE INCREMENTAL ...
    ARGROLLV = 4 #,         %   ARCHIVE ROLLOUT ...
    ARGMERGEV = 32 #,       %   ARCHIVE MERGE ...
      %  FOLLOWING ARE ACTION CODES TO "ARCHIVELIST" PROCEDURE, BUT
      %  THEY ARE NEVER STORED IN RECORDS:
    ARCRESTOREV = 33 #,     %   ARCHIVE RESTORE ...
    ARCRESTOREADDV = 34 #,  %   ARCHIVE RESTOREADD ...
    ARCAUTORESTOREV = 35 #, %   AUTORESTORE
  ARGBADF = [31:1] #,       % ON IF WE FIND OUT TAPE IS OVERWRITTEN
ARGNAMESX = 13 #,           % FOUR SETS OF 18-CHARACTER TAPE NAMES
ARGBOTTSX = 25 #,           % FOUR DATES AND TIMES OF BACKUP
                            %  BOT.  AN "ARGOTTSX" VALUE CAN BE
                            %  USED TO IDENTIFY FILES BACKED UP IN
                            %  THE SAME RUN.  ALSO PROPAGATED TO THE
                            %  VOLUME DIRECTORY "VSBOTTSX" WORD
ARGSNSX = 29 #,             % FOUR LINKS TO SERIALNO LISTS
ARGLOCX = 33 #,             % FOUR WORDS RESERVED FOR FUTURE EXPANSION
ARGVARSTX39 =37 #,         % MINIMUM SUBRECORD FIXED INFORMATION

 % END OF ORIGINAL 3.Ø FIXED STUFF.  ALL (NEW) WORDS FROM HERE TO
 % "buf [ARGX].ARGVARX - 1" MUST BE IN "LINK WORD FORMAT".  THAT
 % IS "ARLINKEDF" IS ON FOR LINKS TO NEW VARIABLE LENGTH STUFF; AND
 % "ARLINKEDF" IS OFF FOR SIMPLE NEW WORDS.

ARGVARSTX = 37 #;           % START OF VARIABLE STUFF POINTED TO
                            % BY "ARLINKF" WORDS IN SUBRECORD

 % AFTER "ARG..." VARIABLES FOR THIS SUBRECORD, NEXT SUBRECORD (IF
 % THERE ARE MORE) APPEARS.  AFTER LAST SUBRECORD AND ITS VARIABLE
 % LENGTH STUFF AN EXTRA WORD IS USED FOR THE CHECKSUM OF THE WHOLE
 % "AR..." & "ARG..." COMBINATION RECORD.

 % MAXRECSIZE = AR FIXED + FILENAME + ARG FIXED + TWO 256 WD SN LISTS
 %              + CHECKSUM WORD
```

About Word Allocation in Archive Records

Each archive record includes at least one reference to a library maintenance tape set. In the ARG portion of the archive records, words are allocated in sets of four. Each word corresponds to a backup tape set. When there is only one backup tape set, information referring to it can appear in any one of the four words. The only places in the ARG

records in which words are not allocated in sets of four occur in the ARGNAMESX slots for tape names; here, tape names are allocated as four sets of three words.

You can detect the presence or absence of a backup reference only by checking the ARGTYPEF field of the [ARGBKSX] through [ARGBKSX + 3] words in the ARG portion of the array. For example, suppose an array named ARCREC contains an archive record. Backup 0 is valid if

```
ARCREC [ARCREC [ARSRX].ARSRF + ARGBKSX + Ø].ARGTYPEF NEQ AGREMPTYV
```

Backup 1 is valid if

```
ARCREC [ARCREC [ARSRX].ARSRF + ARGBKSX + 1].ARGTYPEF NEQ ARGEMPTYV
```

Backup 2 is valid if

```
ARCREC [ARCREC [ARSRX].ARSRF + ARGBKSX + 2].ARGTYPEF NEQ ARGEMPTYV
```

Backup 3 is valid if

```
ARCREC [ARCREC [ARSRX].ARSRF + ARGBKSX + 3].ARGTYPEF NEQ ARGEMPTYV
```

The disk file name takes the standard form. It begins in the word

```
ARCREC [ARCREC [ARTITLEX].ARLINKF]
```

The backup tape names take a substandard form. That is, each name tape consists of one byte-length character, followed by a name that is from 1 to 17 characters long. Tape names occupy the following words:

```
ARCREC [ARCREC [ARSRX].ARSRF + ARGNAMESX + Ø]
  "      "       "      "        "          1
  "      "       "      "        "          2

ARCREC [ARCREC [ARSRX].ARSRF + ARGNAMESX + 3]
  "      "       "      "        "          4
  "      "       "      "        "          5

ARCREC [ARCREC [ARSRX].ARSRF + ARGNAMESX + 6]
  "      "       "      "        "          7
  "      "       "      "        "          8

ARCREC [ARCREC [ARSRX].ARSRF + ARGNAMESX + 9]
  "      "       "      "        "         1Ø
  "      "       "      "        "         11
```

The words that contain dates and times, such as *ARTIMESTAMPX, ARGCREATEX, ARGACCESSX,* and *ARGBOTTSX,* return the date and time in the same format that the ALGOL intrinsic "TIME (6)" does.

Glossary

This glossary defines terms as they are used in this guide. These terms may have a broader definition outside the scope of this guide.

A

AAST
See archive access structure table.

access structure
The central disk file that the system uses to determine the location of files stored on the disk subsystem. On noncataloging systems, the access structure contains entries that are used to locate files that are accessible to the system. On cataloging systems, the access structure contains entries that are used to locate all available versions of a file. The access structure is also called the catalog on both cataloging and noncataloging systems.

address
The identification of the location of a disk sector.

alternate halt/load family
A family that contains a backup copy of the Master Control Program (MCP) object code file and is not the current halt/load family. If the current halt/load family fails, the alternate halt/load family can be used to operate the system.

alternate halt/load unit
A disk that contains a duplicated Master Control Program (MCP) object code file and is a member of the current halt/load family.

archive access structure table (AAST)
Special records in an archive directory that the system uses to access archive data records by file name.

archive backup information
Information that is generated and used by Work Flow Language (WFL) *ARCHIVE* statements. Archive backup information describes to what tapes backup copies of disk files were copied and contains the file attributes of those backup files at the time they were saved.

archive directory
A directory file that the system uses to store archive backup information for disk files. Archive backup information is generated and used by Work Flow Language (WFL) *ARCHIVE* statements. The system stores archive directories on the catalog family. Each disk family on the system has one archive directory, although duplicates can be created.

ARCHIVING
The system option that, when enabled with the OP (Options) system command, enables the system to store and retrieve information about old generations of cataloged files in a database.

area
The amount of contiguous disk space that is allocated at one time to a disk file as it is being created or expanded. Synonym for row.

AREASIZE
A file attribute that refers to the number of records allocated to a particular area of disk.

automatic display mode
A display mode that can be initiated at an operator display terminal (ODT) through the use of the ADM (Automatic Display Mode) system command. In this mode, various types of information about the system are displayed at regular intervals.

B

backup file
(1) A copy of a file on a cataloging system that has been saved with the COPY & BACKUP Work Flow Language (WFL) statement. (2) A copy of a file that is stored offline so that it can be copied back onto the system if the original file becomes corrupted or inaccessible.

BADDISK file
A disk file with a FILEKIND of BADDISK. This kind of file is used by the system to cover areas of disks that might be defective. An area of disk that is covered by a BADDISK file is not available for allocation to any other disk files. BADDISK files are created by the system command XD (Bad Disk) and certain forms of the system command RES (Reserve). BADDISK files are sometimes a side effect of the Initialize, Verify, and Reconfigure (IVR) procedure and the system command RC (Reconfigure). Also, certain system programs, such as Library Maintenance, automatically create a BADDISK file when an I/O error occurs.

base pack
A disk that contains a copy of the system directory for the family of that disk and is currently being used by the system to identify and access the family.

block
A group of physically adjacent records that can be transferred to or from a physical device as a group.

BOOTCODE file
A file that contains the maintenance subsystem software, microcode, UTILOADER code, and diagnostic tests, and, on some A Series systems, the BOOTSTRAP program.

C

CANDE

See Command and Edit.

catalog

The central disk file that stores the information about the disks in the system and the disk files. This file is named SYSTEM/CATALOG/<family index number> on cataloging systems and SYSTEM/ACCESS/<family index number> on noncataloging systems.

catalog family

The disk family designated by the DL (Disk Location) system command as being the one on which SYSTEM/CATALOG or SYSTEM/ACCESS file is stored.

catalog level

An integer value that determines how many generations of each file an installation can have. The catalog level is established by assigning a value to the CATALOGLEVELSET define that is compiled into the Master Control Program (MCP). CATALOGLEVELSET can be assigned a value in the range 1 through 7. The default value is 3.

catalog rebuild

The process on a cataloging system in which the system updates the file access structure table (FAST) with information about backup copies of cataloged files.

cataloged file

A file that has been entered into the catalog. The user can enter a file into the catalog by using the *CATALOG ADD* or *COPY & CATALOG* Work Flow Language (WFL) statement, by assigning to the USECATALOG file attribute the value TRUE, or by enabling the system option USECATDEFAULT.

CATALOGING

A system option that, when enabled with the OP (Options) system command, allows the system to keep track of copies of files that have been backed up onto tape or disk.

checkerboarding

A situation in which only small areas are available between the in-use areas of storage. Many areas can be available, but the system might not be able to use them because none of the areas is large enough.

checksum

A directory test that performs a data integrity check of a directory record to ensure that it has not been corrupted.

code file

See object code file and source file.

Command and Edit (CANDE)

A time-sharing message control system (MCS) that allows a user to create and edit files, and to develop, test, and execute programs, interactively.

complementing
The process the system uses to construct an available disk table. The Master Control Program (MCP) starts with the entire available disk and then reads the disk file headers on a family to determine the space that is already allocated. The MCP then takes the complement of the allocated space to determine the space that is still available.

continuation pack
A disk that is not currently being used as the base pack for a multipack family. A continuation pack can have a copy of the system directory for the family.

COPY substatement
The part of the Work Flow Language (WFL) *COPY* or *ADD* statement that includes a HOSTNAME specification. Each source and destination host is treated as a separate substatement and is processed separately. In the command *COPY F TO T1, T2(HOSTNAME = X)*, substatement 1 is *COPY F TO T1*, and substatement 2 is *COPY F TO T2(HOSTNAME = X)*.

CYCLE
A file attribute that can be used with the VERSION file attribute to distinguish the generations of a file.

cylinder
All the tracks on all the platters of a disk that have the same radius.

D

data link processor (DLP)
A processor that serves as the system interface to a specific peripheral device, controller, or communications network.

Data Management System II (DMSII)
A specialized system software package used to describe a database and maintain the relationships among the data elements in the database.

destination host
In a file copy process, the host system to which files are being copied.

directory
(1) Special disk files used by the system that include archive directories, catalogs, and system directories. (2) The partial name of a disk file up to one of the following terminators: a slash followed by an equal sign (/=) or a right parenthesis followed by an equal sign ()=).

disk
(1) A random-access data storage device consisting of one or more circular platters that contain bits of information recorded in concentric circular paths called tracks. Data on a disk are accessed by movable read/write heads. Some disks are removable. (2) Synonym for disk pack, pack.

disk drive
The device on which a disk is mounted. The disk drive has movable read/write heads that access the data on the disk.

disk drive controller
The device that controls the disk drive units and transfers information between the host system and the disk drive units. On some systems, this device is also referred to as a D-Machine.

disk file
A file stored on a disk or disk pack.

disk file header
A data structure that contains information about a disk file, such as the physical location of the file on the disk and various file attributes. A disk file header is also referred to as a header.

disk pack
(1) A random-access data storage device consisting of one or more circular platters that contain information recorded in concentric circular paths called tracks. Data on a disk pack are accessed by movable read/write heads. Some disk packs are removable. (2) Synonym for disk, pack.

disk resource control (DRC) system
An optional feature of the disk subsystem that provides the ability to control disk space on a per user basis. The DRC system does not support interchange (IC) packs or installation-allocated disk (IAD) usage. DRC is not a security system, but normal security checking occurs.

DLP
See data link processor.

DMSII
See Data Management System II.

DRC
See disk resource control system.

E

EMS
See Entry and Medium Systems.

end of file (EOF)
A code at the end of a data file that signals that the last record in the file has been processed.

end of volume (EOV)
A code at the end of a tape volume that signals that the last record in the volume has been processed, but that the file is continued on another tape.

Entry and Medium Systems (EMS)
A designation referring to the Micro A and A 1 through A 10 systems.

EOF
See end of file.

EOV
See end of volume.

F

family
(1) One or more disks logically grouped and treated as a single entity by the system. Each family has a name, and all disks in the family must have been entered into the family with the RC (Reconfigure Disk) system command. (2) The name of the disk or disk pack on which a physical file is located.

family index
A 3-digit number the system assigns to a disk when the disk is added to a family. This family index value must be in the range 1 to 255, inclusive. The base pack is assigned the family index number, the first continuation pack is assigned 002, and so on. A family index is also referred to as a family index number.

family index number
See family index.

family member
A disk that is a base or continuation pack in a family.

family name
The name, consisting of up to 17 alphanumeric characters, assigned by an installation to identify a family of disks.

family rebuild
The process in which the system reconstructs the file access structure table (FAST) entry for a family by reading its system directory.

family substitution
A method for redirecting references to files on a disk family to avoid entering the actual family name in commands or file names. For example, if the user enters *FAMILY=PACK OTHERWISE DISK*, that user's file requests are checked on disk packs named PACK and DISK.

FAST
See file access structure table.

file access structure table (FAST)
Special records in the access structure or catalog directory that the system uses to locate disk files. The FAST contains a pointer to the header of each disk file in the system directory of each family.

file attribute
An element that describes a characteristic of a file and provides information the system needs to handle the file. Examples of file attributes are the file title, record size, number of areas, and date of creation. For disk files, permanent file attribute values are stored in the disk file header.

file name
A unique identifier for a file, consisting of name constants separated by slashes. Each name constant consists of letters, digits, and selected special characters. A file name can be optionally preceded by an asterisk (*) or usercode, and optionally followed by ON and a family name.

file title
The complete identifier for a file that consists of the file name, and the word ON, and the family name.

flat directory
See system directory.

G

genealogy
The ranking of a generation of a file relative to the other generations of that file. The generation with the highest CYCLE file attribute, the highest VERSION file attribute within that CYCLE, and the most recent timestamp within that CYCLE and VERSION is said to have the best genealogy.

generation
A particular, available copy of a file. The generation of a file is determined by the file attributes CYCLE and VERSION, and the timestamp of the file.

GENERATION
A file attribute on cataloging systems that allows the user to select a particular generation of a file.

guard file
A disk file created by the GUARDFILE utility program that describes the access rights of various users and programs to a program, data file, or database.

H

halt/load
A system-initialization procedure that temporarily halts the system and loads the operating system from a disk to main memory.

halt/load family
The disk family that contains the currently operative Master Control Program (MCP) object code file.

halt/load pack
A pack that contains a Master Control Program (MCP) object code file designated as the currently operative MCP. This pack is mounted on a drive that has been or will be designated as the halt/load unit.

halt/load unit
The drive on which the halt/load pack is mounted.

HDRLOCATION
A directory record test that ensures that the directory record has the correct record number.

HDRMARKER
A directory record test that ensures that the directory record is the correct type.

HDU
See host data unit.

header
A data structure that contains information about a disk file, such as the physical location of the file on the disk and various file attributes. A header is also referred to as a disk file header.

host data unit (HDU)
The A 12 and A 15 system host interface to the I/O subsystem. An HDU is configured with up to three host-dependent ports (HDPs), each of which supports two message level interface (MLI) cables.

I

I/O
Input/output. An operation in which the system reads data from or writes data to a file on a peripheral device such as a disk drive.

I/O controller
The processor that provides the interface between the host system and a peripheral device such as a disk drive controller.

I/O processor (IOP)
A specialized processor for moving data between system memory and the I/O subsystem.

I/O subsystem
The hardware and software that manage all transfers of information between the Master Control Program (MCP) and peripheral devices.

InfoGuard
The Unisys security-enhancement software for A Series systems. InfoGuard provides such features as password management, selective logging and auditing, tape volume security, and simplified system-security configuration.

initialize, verify, and relocate (IVR)
A maintenance procedure used to write sector boundaries and a blank label on a disk. The IVR procedure can be used to make a new disk pack usable by the system or a damaged disk reusable by eliminating defective sectors. The end product of an IVR is a master available table (MAT) of available disk sectors.

interchange disk pack
A disk pack with a directory format that allows files to be transferred from a Unisys A Series system to certain other Unisys systems.

IOP
See I/O processor.

IVR
See initialize, verify, and relocate.

L

label
The first 28 sectors on a disk, on which information about the disk is stored. This information includes the family name and serial number, the master available table (MAT), the family index number, information about the family base pack, and a pointer to the system directory if the disk contains a directory.

label error
An error that indicates that a label or system directory of a disk was not processed correctly when the disk was prepared for system use with the RY (Ready) system command.

LAST
See local access structure table.

library maintenance
A procedure that copies disk files from a disk to a disk, from a disk to a tape, from a tape to a disk, and from a tape to a tape. The procedure is invoked by Work Flow Language (WFL) ADD or COPY statements.

library maintenance tape
A tape created by library maintenance that contains backup copies of disk files.

local access structure table (LAST)
A special file located on the base pack that is used to update the file access structure table (FAST) each time a base pack is brought online. The main value of the LAST is that a time-consuming family rebuild is not necessary when a disk is readied on a noncataloging system.

logical record
The amount of data accessed in the execution of one read or write statement in a program.

look-ahead buffer
An area where the system stores records of input files that it expects to access soon, such as the next record of a file that is being read sequentially.

M

master available table (MAT)
A table stored on each disk that lists the valid sectors on the disk that were successfully processed by the initialize, verify, and relocate (IVR) procedure. Pointers to defective sectors are deleted from the MAT so that these sectors will not be accessed. Normally, the MAT shows the entire disk as being available, minus any defective sectors.

Master Control Program (MCP)
An operating system on A Series systems. The MCP controls the operational environment of the system by performing job selection, memory management, peripheral management, virtual memory management, dynamic subroutine linkage, and logging of errors and system utilization.

MAT
See master available table.

MCP
See Master Control Program.

Memory Disk
A Unisys software feature that enables the use of memory as if it were a disk unit and provides file access with extremely high data-transfer rates relatively little access time. Memory disk is supported on all A Series systems that use the Master Control Program/Advanced Systems (MCP/AS) and on A 15 systems that use the Master Control Program (MCP).

message level interface (MLI)
The interface between the host system, the I/O subsystem, and the data communications subsystem.

message level interface processor (MLIP)
See I/O processor (IOP) and Entry and Medium Systems (EMS).

mirror information table (MIT)
A system status table that contains information about all mirrored sets on the system. The MIT is saved in the directory of the halt/load disk. After a halt/load, the MIT is read into memory from the directory of the halt/load disk.

Mirrored Disk
A software feature on Unisys A Series systems with I/O based on a data link processor (DLP) that enables from two to four disks to be maintained as a mirrored set; that is, as exact copies of each other. The Mirrored Disk feature increases system availability and data integrity, decreases the possibility of data loss due to equipment or media malfunction, and can improve I/O throughput on disk subsystems that experience a high ratio of reads versus writes.

MIT
See mirror information table.

MLI
See message level interface.

MLIP
See message level interface processor.

multidisk family
A family that consists of more than one disk. The system treats the family as a single entity.

N

native-mode disk
The type of disk directory used on most A Series systems. The other type is interchange mode. When this guide refers to the term *disk*, it is referring to native-mode disks unless otherwise noted.

nonremovable disk
A disk that cannot be removed from the disk drive on which it is mounted.

nonresident file
A file stored on a backup tape or a backup copy of a file that is stored on a different disk family from that on which the primary copy of the file is stored. This term does not always pertain to the RESIDENT file attribute.

O

object code file
A file produced by a compiler when a program is compiled successfully. The file contains instructions in machine-executable object code.

ODT
See operator display terminal.

offline
Pertaining to the state of being not accessible by the operating system.

online
Pertaining to the state of being accessible by the operating system.

operator display terminal (ODT)
A system control terminal (SCT) configured for direct communications with the operating system. The ODT is used primarily by operations personnel for entering commands that control and direct the system and its resources.

outstanding write list (OWL)
A table used by a disk subsystem that uses the mirrored disk feature. The table is used to assure that data written to one disk in a mirrored set is also written to the remaining disks in the set.

OWL
See outstanding write list.

P

PA
See peripheral association.

pack (PK)
(1) A random-access data storage device consisting of one or more circular platters that contain information recorded in concentric circular paths called tracks. Data on a pack are accessed by movable read/write heads. Some packs are removable. (2) Synonym for disk pack, disk.

pack access structure table (PAST)
A table used by the system to locate disk families. It contains pointers into the file access structure table (FAST) that indicate where the entries for the files of a family are stored.

PAST
See pack access structure table.

peripheral association (PA)
A scheme used by operators to cause the job output submitted through a specified peripheral device to be directed to another specified unit.

peripheral device
A hardware device used for input, output, or file storage. Examples are magnetic tape drives, printers, disk drives, and operator display terminals (ODTs). In this guide, the term *peripheral device* refers to disk drives.

peripheral test driver (PTD)
A module of the Master Control Program (MCP) that executes maintenance tests for peripheral devices.

permanent disk file
See permanent file.

permanent file
A disk file that is closed so that its header is placed in the system directory of the family on which the file is stored; the file can be referenced later by its title.

PK
See pack.

presence-bit interrupt
An interrupt that notifies the Master Control Program (MCP) that an array or object code file segment is needed that is not in main memory. The system allocates space for that array or object code file segment. If an array is needed, the system creates a new array or reads an existing array in from the overlay file. When an object code file segment is needed, the system reads it into main memory from disk.

PTD
See peripheral test driver.

R

read/write head
The component of the disk drive that physically transfers data to and from the disk.

rebuild
In the disk subsystem, a concept that refers to either of the following: a family rebuild, in which the system constructs the file access structure table (FAST) entry for a family by reading its system directory; or a catalog rebuild (on a cataloging system) in which the system updates the family access structure table (FAST) with information about cataloged files.

removable disk
A disk that can be removed from the drive on which it is mounted.

resident file
The primary copy of the file that is stored on a disk, regardless of whether or not the disk is online. Backup copies of files stored on another disk family are not considered resident. This term does not always pertain to the RESIDENT file attribute.

resource management module (RMM)
A hardware module that interfaces with the I/O subsystem and schedules tasks on the E-mode processor (EMP) by way of a message protocol.

RMM
See resource management module.

row
The amount of contiguous disk space that is allocated at one time to a disk file as it is being created or expanded. The Master Control Program (MCP) uses the term *row* in its processing. See also area.

RSVP
A message issued by the system when it requires information about an operation; the system waits for a reply before proceeding.

S

scratch tape
A labeled magnetic tape (MT) whose label indicates that there are no files on the tape. Old data might remain on the tape, but this old data cannot be read unless the tape is read as an unlabeled tape. The old data present on a scratch tape is written over when new data is written to the tape.

sector
A subdivision of a track on a disk. A sector is the minimum addressable area on a disk pack. Unisys A Series system sectors are 30 words, or 180 bytes, long. Synonym for segment.

seek
The movement of the read/write heads of a disk drive to the specified track of the disk.

segment
A subdivision of a track on a disk. A segment is the minimum addressable area on a disk pack. Unisys A Series system segments are 30 words, or 180 bytes, long. Synonym for sector.

serial number
The 6-character field an installation assigns to a disk or magnetic tape to uniquely identify it. The serial number is stored on the label of the disk or tape.

source file
(1) A file in which a source program is stored. (2) A file containing instructions written in a programming language.

source host
In a file copy process, the host system from which files are being copied.

standby halt/load family
A disk family that contains a Master Control Program (MCP) object code file and is in the standby halt/load family list. When system state tables such as the mirror information table (MIT) are changed by the system, the MCP also updates these tables on all standby halt/load families. A disk can be made into a standby halt/load unit by using the clause + *STANDBY* with the CM (Change MCP) system command.

system control terminal (SCT)
A terminal used to enter information. An SCT can be used three ways: as an operator display terminal (ODT) to interface with the operating system, as a maintenance display terminal (MDT) to interface with the maintenance subsystem, or as a remote display terminal (RDT) to interface with remote support. The windows providing these uses are available once the automatic initialization sequence has finished.

system directory
(1) A special disk file on each disk family that the system uses to locate files on that family. The system directory, also referred to as the flat directory, contains a disk file header for each permanent file in the family. (2) The logical directory for nonusercoded files.

system file
A file that is stored on a disk and contains system software, such as the Master Control Program (MCP), system directories, and the access structure.

SYSTEM/ACCESS
The name of the file on noncataloging systems that contains the access structure.

SYSTEM/CATALOG
The name of the file on cataloging systems that contains the access structure and the catalog.

T

temporary disk file
See temporary file.

temporary file
A file that does not need to be saved. When a temporary disk file is closed, its disk space is returned to the system. The header of a temporary file is not stored in the system directory.

timestamp
An encoded, 48-bit numerical value for the time and date. Various timestamps are maintained by the system for each disk file. Timestamps note the time and date a file was created, last altered, and last accessed.

track
A circular path on the surface of a disk used to store data.

U

UCF
See User Communication Form.

unit
A peripheral device such as a disk drive or a tape drive.

unit number
A number assigned by an installation to a peripheral device, such as a disk drive, and used to identify the device. The unit number commonly appears in conjunction with an acronym indicating the type of unit, which provides a unique identifier for a particular peripheral.

unvolumed disk
A disk that has no matching entry in the volume library.

unvolumed family
A family disk that has no matching entry in the volume library.

USECATDEFAULT
The system option that, when enabled with the OP (Options) system command, assigns TRUE as the default value of the USECATALOG file attribute.

User Communication Form (UCF)
A form used by Unisys customers to report problems and express comments about Unisys products to support organizations.

V

VAST
See volume access structure table.

VERSION
A file attribute that can be used with the CYCLE file attribute to distinguish the generations of a file.

volume
The medium of a mass storage device such as a disk, disk pack, or tape reel. The term *volume* is not restricted to the volume library on a cataloging system or the volume directory on a system with tape volume security. For example, on the BTOS family of workstations, the hard disk is a volume, and each floppy disk is a volume. When a volume is initialized, it is assigned a volume name and an optional password.

volume access structure table (VAST)
A section of the catalog file on cataloging systems that the system uses to access the volume library.

volume directory
A section of the catalog that tracks the status of tapes on a system that uses the tape security subsystem.

volume library
A section of the catalog that keeps track of all tapes and volumed disks used on a cataloging system.

volumed disk
A disk that has been entered into the volume library with the VOLUME ADD Work Flow Language (WFL) statement so that the disk can be used to store cataloged files on a cataloging system.

volumed family
See volumed disk.

W

WFL
See Work Flow Language.

Work Flow Language (WFL)

A Unisys language used for constructing jobs that compile and run programs on A Series systems. WFL includes variables, expressions, and flow-of-control statements that offer the programmer a wide range of capabilities with regard to task control.

Bibliography

A Series CANDE Operations Reference Manual (form 8600 1500). Unisys Corporation.

A Series DMSII Utilities Operations Guide (form 8600 0759). Unisys Corporation.

A Series File Attributes Programming Reference Manual (form 8600 0064). Unisys Corporation. Formerly *A Series I/O Subsystem Programming Reference Manual* .

A Series GETSTATUS/SETSTATUS Programming Reference Manual (form 8600 0346). Unisys Corporation.

A Series I/O Subsystem Programming Guide (form 8600 0056). Unisys Corporation. Formerly *I/O Subsystem Programming Reference Manual* .

A Series Security Administration Guide (form 8600 0973). Unisys Corporation.

A Series System Commands Operations Reference Manual (form 8600 0395). Unisys Corporation.

A Series System Configuration Guide (form 8600 0445). Unisys Corporation.

A Series System Messages Support Reference Manual (form 8600 0429). Unisys Corporation.

A Series System Operations Guide (form 8600 0387). Unisys Corporation.

A Series System Software Support Reference Manual (form 8600 0478). Unisys Corporation.

A Series System Software Utilities Operations Reference Manual (form 8600 0460). Unisys Corporation.

A Series Task Attributes Programming Reference Manual (form 8600 0502). Unisys Corporation. Formerly *A Series Work Flow Administration and Programming Guide* .

A Series Task Management Programming Guide (form 8600 0494). Unisys Corporation. Formerly *A Series Work Flow Administration and Programming Guide* .

A Series Work Flow Language (WFL) Programming Reference Manual (form 8600 1047). Unisys Corporation.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

UNISYS

Help Us To Help You

Publication Title

Form Number

Unisys Corporation is interested in your comments and suggestions reguarding this manual. We will use them to improve the quality of your Product Information. Please check type of suggestion:

☐ Addition ☐ Deletion ☐ Revision ☐ Error

Comments:

Name Telephone number ( )

Title Company

Address

City State Zip code

Cut along dotted line

Tape

Please Do Not Staple

Tape

Fold Here

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS MAIL PERMIT NO. 817 DETROIT, MI

POSTAGE WILL BE PAID BY ADDRESSEE

UNISYS CORPORATION
ATTN: PUBLICATIONS
25725 JERONIMO ROAD
MISSION VIEJO, CA 92691-9826


86000668-000