Section 2 FILE CONCEPTS

In this section: See page

In this section (cont): See page

Summary

This section describes the functions provided by the File System. It discusses the File System conventions used with disk, magnetic tape, and unit record device files, and describes the remote file access capability.

INTRODUCTION

File

A file is a logical unit of data composed of a collection of records. The principal external devices available for storing files are:

- Disk devices (for example, diskettes, cartridge disks, cartridge module disks, fixed (sealed) disks, and mass storage units)
- Magnetic tape units (for example, 1/2-inch tapes and 1/4-inch cartridge tapes).

Volume

The media residing on the external devices are referred to as volumes (for example, disk volume, tape volume).

Various conventions have been established to identify and locate files stored on disk and magnetic tape. These conventions facilitate the orderly and efficient use of the data stored on the volumes.

Unit Record Devices

Unit record devices (such as card readers, card punches, printers, and terminals) also use the file concepts. However, since unit record devices cannot be used to store files, there is less need to establish conventions for identification and location. A unit record file is simply the data that is read or written at any one time (for example, a line entered at a terminal).

DISK FILE CONVENTIONS

You must be able to specify an access path to any given file on a disk volume that contains multiple files. Files must therefore be organized on the volume in some predictable fashion. The File System provides a set of volume organization conventions by which the system can locate any element that resides on the volume.

The principle elements of this organization, aside from the files themselves, are directories. The access path to any given element on a volume is known as a pathname.

Directories

Files on disk devices are logically arranged by the File System in a tree-structured hierarchy. The basic elements of this hierarchy are special files known as directories. The directories are used to point to the location of data files, which are the endpoints of the tree structure.

A directory on a disk volume is an index that contains the names and starting locations (sector numbers on the volume) of files or other directories (or both). The elements in the directory are said to be "contained in" or "subordinate to" the directory. Therefore, the organization of a disk volume is a multilevel structure. The complexity of the access path to any given element in the structure depends on the number of directories between the root and the desired element.

A sample directory structure is illustrated in Figure 2-1. The base directory on a volume is termed a root directory. In Figure 2-1 the root directory is VOL01. Root directory VOL01 contains two subordinate directories, DIR1 and DIR2. Directories DIR1 and DIR2, in turn, contain data files FILEA, FILEB, FILEC, and FILED.

The root directory and other special types of directories are described in the following paragraphs.

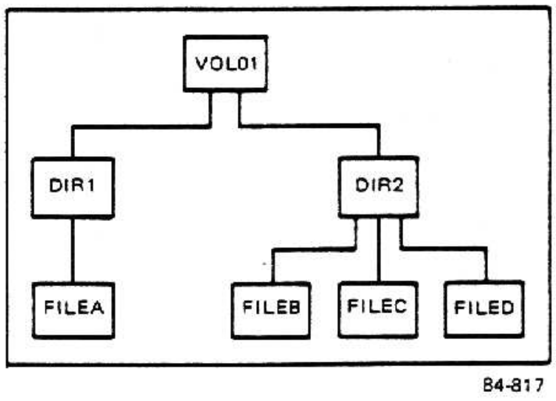


Figure 2-1. Example of Disk Directory Structure

Root Directory

The File System maintains a tree structure for each disk mounted at any given time. At the base of each tree structure is a directory known as the root directory. This is the directory that ultimately contains every element that resides on the volume, either immediately or indirectly subordinate to it. The root directory name is the same as the volume identifier of the volume on which it resides. The directory VOL01 in Figure 2-1 is the root directory on the disk volume VOL01.

System Root Directory

One or more disk root directories can be known to the system at any time during its operation. One of these, the System Root Directory (SRD), is required at all times. The absolute access paths of files in the SRD can start with two greater-than signs (>>). The volume used by the operator to initialize the system establishes the SRD. The boot volume must contain the SRD; it also normally contains system programs, commands, and other routinely used elements. The SRD must contain a number of directories and files that the system needs to perform its functions, including Z3EXECUTIVEL, SID, AID, HIS, and USER_REG. For more information, refer to the *System Building and Administration* manual.

User Root Directory

The File System can recognize one User Root Directory (URD), which you define through the Change System Directories command with the -ROOT argument. Files in the URD can have absolute access paths that start with a single "greater-than" character. The URD contains items such as UDD, LDD, MDD, FORMS, PROGS, and TRANS. For more information, refer to the *System Building and Administration* manual.

The URD and SRD can reside on different volumes or on the same volume. The installation can also have user volumes created to meet the installation's own particular needs. These volumes may contain user application programs and their associated data files, application program source and object code files, listing files, and anything else a user might want to store temporarily or permanently.

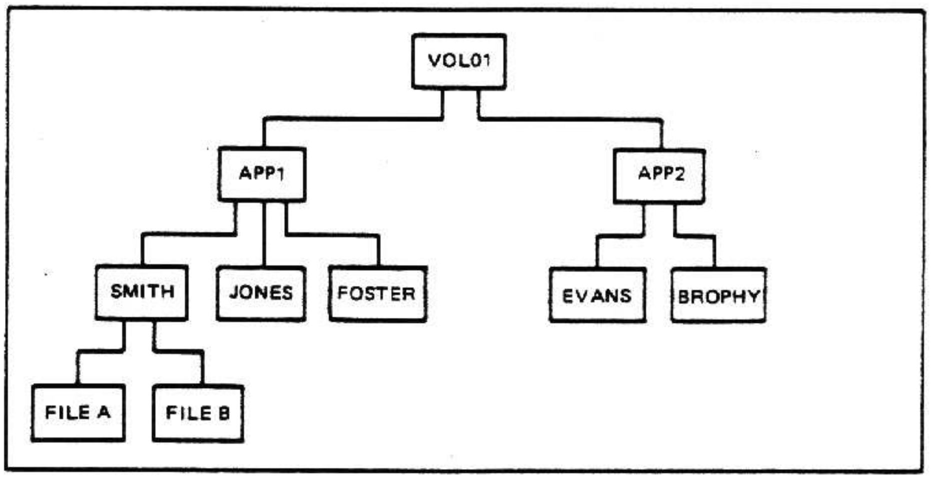
Refer to "Links" later in this section for information on another way to distribute software (system or user) onto more than one volume.

Intermediate Directories

When you first create (format and name) a volume under the File System, it contains only a root directory. Within this directory, you can create any additional directories required to satisfy the needs of the installation.

Consider, for example, a volume that is to contain data used by two application projects, each of which has several users associated with it. Each user has one or more files of interest to him or her. The volume has been initialized and contains a root directory name. Two directories can be created subordinate to the root directory, each identified by the project name. Then, subordinate to these directories, a directory can be created for each user associated with each project.

The data files are all contained within the personal directories. This sample directory structure is illustrated in Figure 2-2.



84-818

Figure 2-2. Sample Directory Structure

Deleting Directories

When the need for a user-created directory no longer exists, the directory can be deleted from the File System (deleted from the disk). The space it occupies, as well as the space occupied by its attributes in the immediately superior directory, is then available for reuse. A directory must be empty before it can be deleted. All directories and files subordinate to the one to be deleted must have been previously deleted by explicit commands.

Working Directory

The File System always starts at a root directory when it searches for a disk file or a directory. At times the search for an element residing on a disk volume may traverse a number of intermediate directory levels before the desired element is located, and the File System must be supplied with the names of all the directories it must pass on the way.

Frequently all files of interest to a user doing work on the system are contained in a single directory that is three or four levels deep in the hierarchy. It is convenient to be able to refer to files in relation to a directory at some arbitrary level in the hierarchy rather than in relation to the root directory. The File System allows this to be done by recognizing a special kind of directory known as a working directory.

A working directory establishes a reference point that enables you to specify the name of a file or another directory in terms of its position relative to the working directory. If the access path of the working directory is made known to the File System, and if the desired element is contained in that directory, the element can be specified by just its name. The File System concatenates this name with the names of the elements of the working directory's access path to form the complete access path to the element.

Disk Directory and File Locations

The File System has total control over the physical location of space allocated to directories and files. You need never be concerned about where a directory or file resides on a volume. When a volume is first initialized, space is allocated to elements in essentially the order in which they are created. But, after the volume has been in use for some time, elements may have been deleted and the space they occupied made reusable. Then, when a new element is created, it is allocated the first available space. If more space is needed, it is obtained from the next free area.

Disk Directory and File Naming Conventions

Allowable Characters

Each disk directory and file name in the File System can consist of the following American Standard Code for Information Interchange (ASCII) characters:

- Uppercase and lowercase primary character set alphabetics (A-Z, a-z)
- Digits (0-9)
- Underscore (_)
- Hyphen (-)
- Period (.)
- Apostrophe (')
- Uppercase and lowercase characters whose hexadecimal equivalents are from C0-FE (Western European Latin alphabet, also called the extended character set).

Characters C0-FE

The characters in the extended character set cannot be used in volume identifiers.

If the terminal is not capable of processing 8-bit data, characters from the extended character set are displayed as periods or as their 7-bit equivalents.

Uppercase and Lowercase

When volumes, files, and directories are created, their identifiers are stored on disk exactly as entered, in uppercase and lowercase characters. For both the primary and extended character sets, the system considers uppercase and lowercase characters to be equivalent (for example, "DATA", "Data", and "data" all refer to the same file).

Forming Names

The first character of any name must not be the character FF (hexadecimal). The underscore character can be used to join two or more words that are to be interpreted as a single name (for example, DATE_TIME). The period character and one or more following alphabetic or numeric characters are normally interpreted as a suffix to a file name. This convention is followed, for example, by a compiler when it generates a file that is to be listed. The compiler identifies this file by creating a name of the form FILEA.L.

Name Length

The name of a root directory (the volume identifier) can be from one through six characters in length. The names of other directories and files can be from 1 through 12 characters in length. The length of a file name must be such that any system-supplied suffix does not result in a name containing more than 12 characters.

Unique Names

Within the system at any given time, the access path to every element must be unique. This requirement leads to the following rules for naming files:

- Only one volume with a given volume identifier can be mounted at any given time. (The system notifies you of an attempt to mount a volume having the same name as one already mounted.)
- Within a given directory, every immediately subordinate directory or file name must be unique. (The Create Directory and Create File commands notify you of an attempt to add a duplicate name.)

Note that uppercase/lowercase differences do not constitute uniqueness. As previously mentioned, "DATA", "Data", and "data" all refer to the same file.

Pathnames

The access path to any File System entity (directory or file) begins with a root directory name and proceeds through zero or more subdirectory levels to the desired entity. The series of directory names (and a file name if a file is the target entity) is known as the entity's pathname. The construction of a pathname is described below.

Length

The total length of any pathname, including all symbols, cannot exceed 57 characters. A working directory pathname, however, cannot exceed 44 characters.

The last (or only) element in a pathname is the name of the entity upon which action is to be taken. This element can be a device name, directory name, or file name, depending on the function to be performed. For example, in the Create Directory command a pathname specifies the name of a directory to be created. The last element of this pathname is interpreted by the command as a directory name; any names preceding the final name are names of superior directories leading to it. An analogous situation occurs in the Create File command, except that in this case the final pathname element is the name of a file to be created.

Symbols Used in Pathnames

The following paragraphs describe the symbols used to construct pathnames.

^	A circumflex (^) is used at the beginning of a pathname to identify the name of a disk volume root directory (for example, ^VOL011).
^>	A circumflex preceding a "greater-than" sign (^>) is used at the beginning of a pathname to identify the root directory of the current working directory (for example, ^>DIR1>FILEA is equivalent to ^VOL011>DIR1>FILEA if the current working directory is on VOL011).
>	A "greater-than" sign (>) is used at the beginning of a pathname and between the names in a pathname.
>name	When used at the beginning of a pathname, the element whose name follows the > symbol is immediately subordinate to the root directory of the user-root volume (it resides under the URD). Honeywell Bull supplied programs assume the URD contains the UDD, LDD, FORMS, MDD, PROGS, and TRANS directories. The correct way to refer to a directory in the URD is to precede the directory name by one "greater-than" sign (for example >UDD).
name>name	When used between names in a pathname the > symbol indicates movement in the hierarchy away from the root directory. The symbol is used to connect two directory names or a directory name and a file name. Each occurrence of the > symbol denotes a change of one hierarchical level. The element to the right of the symbol is immediately subordinate to the element on the left. Reading a pathname from left to right thus indicates movement through the tree structure in a direction away from the root directory. For example, if the root ^VOL011 contains a directory named DIR1, the pathname of DIR1 is ^VOL011>DIR1. If the directory named DIR1 in turn contains a file named FILEA, the pathname of FILEA is ^VOL011>DIR1>FILEA

>>	Two consecutive "greater-than" signs (>>) are used at the beginning of a pathname to specify entities that are subordinate to the SRD. Honeywell Bull supplied programs assume the SRD contains the Z3EXECUTIVEL, SID, AID, HIS, and USER_REG directories. The correct way to refer to a directory in the SRD is to precede the directory name by two "greater-than" signs (for example >>SID). SYSLIB1 and SYSLIB2 can reside in either the SRD or the URD.
<	A "less-than" sign (<) is used at the beginning of a pathname to indicate movement from the working directory toward the root directory. Consecutive symbols can be used to indicate changes of more than one level; each occurrence represents one level change. One or more "less-than" symbols may precede only a pathname that assumes a directory without actually referring to it explicitly. Such a pathname is called a relative pathname.
space	An ASCII space character (hexadecimal 20) is used to indicate the end of an encoded pathname in a program. When represented in memory, a pathname must end with a space character.
Summary	The use of these symbols at the beginning of a pathname can be summarized as follows:

Symbol	Meaning
^volume	Any volume or root directory
^>	Under root of current working directory volume
>	Under URD root
>>	Under SRD root
<	Movement away from current working directory toward volume root

Absolute and Relative Pathnames

Full Pathname

A full pathname is one that begins with a circumflex. A full pathname contains all necessary elements to describe a unique access path to a File System entity, regardless of the type and location of the device on which it resides or where your working or assumed directory is. You use a full pathname to locate directories and files that reside on a device other than that on which the system volume (the volume from which the system was initialized) is mounted.

The File System uses a full pathname when referring to a directory or file. However, it is frequently unnecessary for you to specify all of these elements. The File System can supply some of them when the missing elements are known to it and the abbreviated pathname is used in the appropriate context. An understanding of these conditions and contexts requires an understanding of absolute and relative pathnames.

Absolute Pathname

An absolute pathname is one that begins with a circumflex (^) or one or more "greater-than" symbols (>).

If an absolute pathname begins with a circumflex, it is a full pathname.

If an absolute pathname begins with one "greater-than" symbol, the first element named in the pathname is assumed to be immediately subordinate to the URD.

If the pathname begins with two "greater-than" symbols, the first element named in the pathname is assumed to be directly subordinate to the SRD.

Relative Pathname

A relative pathname is a shortened version of the absolute pathname and assumes the working directory (or a higher directory in the structure) without explicitly referring to it. A relative pathname is one that begins either with a file or directory name or with one or more "less-than" symbols.

If the pathname begins with a name (for example, DIR1>FILEA or FILEB), the name (DIR1 or FILEB) is immediately subordinate to the working directory.

If a relative pathname begins with a "less-than" symbol (for example, <FOSTER), the name following the "less-than" symbol identifies an element that is immediately subordinate not to the working directory, but to the directory to which the working directory is immediately subordinate. If the pathname begins with two "less-than" symbols (for example, <<APP2), APP2 is immediately subordinate to a directory two levels higher than the working directory.

A relative pathname contains one or more names. If it contains more than one name, each name except the last must be a directory name, the first being immediately subordinate to the current working directory level (or to a higher level, as specified by one or more "less-than" symbols), the second immediately subordinate to the first, and so on. The last or only name can be a directory name or a file name, depending on the function being performed.

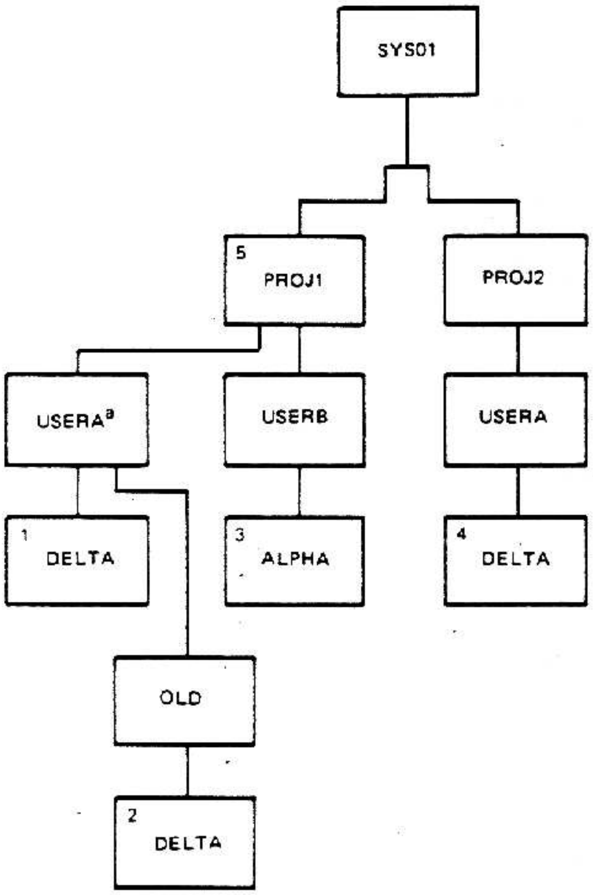
Simple Pathname

A simple pathname is a special case of the relative pathname. A simple pathname consists of only one name: the name of the desired element that is immediately subordinate to the working or assumed directory.

You can refer to a file or directory that is on the same volume (but not subordinate to the working directory) by using an absolute pathname or by using any of the described forms of a relative pathname.

Figure 2-3 shows some relative pathnames and the full pathnames they represent when the working directory pathname is >PROJ1>USERA.

BOX NUMBER	RELATIVE PATHNAME[a]	FULL PATHNAME
1	DELTA	^SYS01>PROJ1>USERA>DELTA
2	OLD>DELTA	^SYS01>PROJ1>USERA>OLD>DELTA
3	<USERB>ALPHA	^SYS01>PROJ1>USERB>ALPHA
4	<<PROJ2>USERA>DELTA	^SYS01>PROJ2>USERA>DELTA
5	<	^SYS01>PROJ1



[a]ASSUME CURRENT WORKING DIRECTORY IS ^SYS01>PROJ1>USERA.

84-819

Figure 2-3. Sample Pathnames

Disk Device Pathname Construction

A special pathname convention is used to specify an entire disk volume. (This pathname convention is typically used in volume copy, create, and dump requests.) The special pathname consists of an exclamation point (!) followed by the symbolic device name and, optionally, the name of the disk volume. The general form of the disk device pathname is:

```
!dev_name[>vol_id]
```

where dev_name is the symbolic device name defined for the disk device at system building time, and vol_id is the File System name of the disk volume, without the circumflex (for example: !MSM00>VOL01).

Exclusive Reservation

If the vol_id is not supplied, reservation of the disk is exclusive (meaning that the reserving task group has read and write access but other users are not allowed to share the volume). This pathname form is used when a new volume is being created.

Read/Share Reservation

If the vol_id is specified, reservation is read/share (meaning that the reserving task group has read access only, other users may read and write). This pathname form is used when copying a volume, when referencing the system volume, or when dumping selected portions of a volume without regard for the hierarchical File System tree structure.

Links

Links are names you create through the Link Name command to refer to files and directories in other volumes or directories as if they were in your working directory (or any other specified directory). Instead of copying a file from one directory to another, you can link to it. You can also link to devices or to other links.

For example, once you have established a link between the name A (in a given directory) and the pathname ^VOLID>MYDIR>MYFILE, you can perform file operations using the link-name A as if it were the pathname. Instead of having to issue the command:

```
    MFA ^VOLID>MYDIR>MYFILE -RECOVER
```

you can issue the command:

```
    MFA A -RECOVER
```

(Assuming you have defined A as a link-name in your current working directory.)

For additional information concerning the Link Name and Unlink Name commands, refer to the *Commands* manual.

Automatic Disk Volume Recognition

The automatic volume recognition facility dynamically notes the mounting of a disk volume. This feature allows the File System to record the root directory name in a device table. All references to disk files and directories begin, explicitly or implicitly, with a root directory name; therefore, every mounted file is automatically accessible to the File System software.

Disk File Comments

You can include a comment as a file attribute when creating or modifying a file, directory, or index. A file comment can range from 1 to 72 ASCII characters in length, surrounded by quotes. The List Names (LS) and Peruse Directory (PD) commands display comments. You create or revise a comment with the -COMMENT argument of the Create File (CR), Create Index (CX), Create Directory (CD), Modify File Attributes (MFA), or Modify Directory Attributes (MDA) command.

You can allow multiple comments to be applied to a single file by assigning a record type to a file comment with the -COMMENT_TYPE argument of the above commands.

Disk File Organization

Since no one disk file organization can meet the needs of all users at all times, the system supports several different organizations, each of which is well suited to a particular application. Most of the supported organizations are based on the concept of a control interval (a unit of transfer between memory and disk) and are referred to as Unified File Access System (UFAS) files. UFAS file organizations provide file processing compatibility with other Honeywell Bull systems.

You establish the organization of a data file when you create the file through the Create File command or program call. You read and write the file using commands, statements, and macrocalls provided by the compilers and Assembler.

The following paragraphs summarize the disk file organizations. Refer to the *Data File Organizations and Formats* manual for detailed descriptions of each organization.

UFAS Sequential Disk File Organization

Logical records are normally read from or written to a sequential file in consecutive order. Records must be written sequentially although the file can be positioned for writing through the use of a simple key. Records can be read, modified, or deleted directly when you specify their exact control interval and record address (simple key). Records cannot be inserted; they can be appended to the end of a file. Fixed- or variable-length records can be used. If a record is deleted, the position it occupied cannot be reused.

UFAS Relative Disk File Organization

A relative disk file can contain fixed- or variable-length records. If variable-length records are used, they occupy fixed-length slots (and the size of the largest record must be specified). Both sequential and direct access are supported; in direct access, simple and relative keys can be used. A record can be updated (rewritten), deleted, or appended to the file. If a record is deleted, the position it occupied can be used for a new record. A file can be created directly if you specify relative record numbers in random sequence.

UFAS Indexed Disk File Organization

In an indexed disk file organization each logical record contains a fixed-size key field that occupies a fixed position. Records are logically ordered by key value; they can be accessed sequentially in key sequence or directly by key value. Fixed- or variable-length records can be used. Variable-length records occupy variable-length slots. A record can be updated, deleted, or inserted in key sequence into available free space. When no space is available to insert a record in key sequence, the record is placed in an overflow area. When the file is initially loaded, the records must be supplied in sequence by key value.

UFAS Random Disk File Organization

In a random disk file organization records are accessed directly or sequentially. Variable-length records occupy variable-length slots. Direct access of records is performed through CALC keys, which are fixed in size and located within each record. Records are positioned according to a technique involving an arithmetic derivation of their CALC keys. This derivation is called a hashing algorithm (and is carried out by the system). Insertions, updates, and deletions are handled according to key value. When the file is initially loaded, records can be supplied in random key value sequence.

UFAS Dynamic Disk File Organization

A dynamic disk file can contain fixed- or variable-length records and supports inventory information to describe available space. The main purpose of this file organization is to provide an efficient storage organization for records to be accessed through alternate indexes (explained below). Records are accessed sequentially or directly. Variable-length records occupy variable-length slots. Records can be accessed indirectly through alternate indexes or directly by specifying their exact control interval and record address (simple key). Records are inserted into the file according to inventory information on a "best fit" basis. When the file is initially loaded, records can be supplied in random key value sequence.

Non-UFAS Relative Disk File Organizations

Non-UFAS relative disk file organizations are not compatible with other Honeywell Bull systems. These file organizations have fewer functional capabilities than UFAS files but require little or no space overhead. The non-UFAS file organizations are fixed relative and string relative.

Fixed Relative A fixed relative disk file can contain only fixed-length records. All records in the file are considered active; the file cannot contain deletable records. A fixed relative file can be accessed directly or sequentially. New records can be inserted anywhere in the file.

String Relative A string relative disk file can contain variable-length records. All records in the file are considered active. A string relative file can be accessed directly or sequentially. The ASCII line feed character (hex 0A) is automatically appended to the end of each record.

A DOS compatible string relative file is also available. It supports all the capabilities of a standard string relative file, but includes the ASCII carriage return and line feed characters (hex 0A0D) at the end of each record.

Pipes

A pipe is a special kind of UFAS sequential file that is used for synchronizing and passing information among multiple cooperating tasks. Pipes are accessed (Reserved, Opened, Read, Written, Closed, and Removed) just as in any other sequential file. Pipes provide a synchronization and queuing facility, and offer a convenient way of organizing and distributing work.

One or more tasks write into the pipe while others read from it. If the pipe is empty but open for writing, Read requests are suspended until data (a logical record) is available. A Read implicitly deletes the logical record just read from the pipe. When the pipe is empty and no longer open for writing, Read actions return the normal end-of-file status.

Alternate Indexes

Alternate indexes allow you to define any number of alternate record keys to provide any number of different logical orderings of keyed records within a single disk file. In effect, alternate indexes provide different orderings (views) of the same data. The same data file can be viewed in many different ways by having more than one alternate index. For example, an application could have a UFAS relative file containing employee information with alternate indexes for employee numbers, employee names, and social security numbers. You could access such a file as a relative file, as an indexed file ordered by employee numbers, as an indexed file ordered by employee names, or as an indexed file ordered by social security numbers.

File Types

The alternate index capability exists in addition to the normal access mode based on type of file. You can establish an alternate index for any UFAS relative, indexed, random, or dynamic disk file.

Access

A file with more than one index can be accessed in a number of ways. The manner in which the file is reserved (through the Get File command) determines how the file is accessed. If the data file itself is reserved, the file can be accessed normally (according to file organization) or by a key that is supported by one of the indexes. When the data file is reserved through an alternate index, the contents of the file can be accessed as a standard indexed file.

Indexes as Keys

Additionally, if more than one index exists, the indexes can be used as alternate keys to refer to the data. When an alternate index is used for file reservation, that index is used as the primary key. The remaining indexes can be used as alternate keys. Any index can be selected as a primary index. When one index is used to access the file, it and the other indexes are automatically updated as the file is updated.

Dynamic Files

UFAS dynamic disk files contain inventory information to manage available file space. Therefore, in highly volatile file environments that include many insert and delete operations, dynamic disk files are the ideal data files to be used with alternate indexes.

Key Types

Character string, signed binary, signed unpacked decimal, and signed or unsigned packed decimal key types can be used. Single component keys, ordered in ascending or descending sequence, are supported. Duplicate keys (more than one record in a file with the same key value) are supported on an index-by-index basis.

Index Creation An alternate index is created with the Create Index command. Arguments of this command specify the name of the index and the name of the data file with which it is to be associated. The system creates the index on the same directory as the data file and, unless otherwise specified, with the same control interval size as that of the data file.

Refer to the *Data File Organizations and Formats* manual for further information.

Disk File Structure

A disk file is a logical unit of data composed of a collection of records. The unit of transfer between memory and disk is called a control interval.

Record A record is a user-created collection of logically related data fields. Records are treated as units and can be fixed or variable in length.

Control Interval A control interval is the unit that is transferred between memory and disk. The size of a control interval is user-specified (a multiple of 256 bytes) and remains constant for the file. A UFAS file is composed of control intervals that are numbered, starting at one. The control interval also determines the buffer size (which must be a multiple of the control interval size).

Disk File Protection

The File System provides facilities that enable you to control the access to files and directories, to control the concurrent access to files, and to control the contention for records within shared files.

Access Control

Access control is an optional File System feature that allows the creator of a file or directory to specify which users (if any) are to be granted access to the file or directory and what types of access these users are to be granted.

There are two general forms of access control: Access Control Lists (ACLs) and Common Access Control Lists (CACLs). ACLs apply directly to a file or directory; CACLs apply equally to all immediately subordinate entries in a directory. Entries in the ACLs and CACLs are managed through Set Access, Delete Access, and List Access commands.

Access control is a file or directory attribute. The File System maintains in each directory a list of users and the type of access each user is allowed. If a directory does not contain such a list, the items contained within it are not protected and are accessible to all users. (Access control applies only to disk files and directories. Tape files and other device-type files such as terminals and card readers cannot be protected through the access control facility.)

File Access Types

Access types for files are read, write, and execute. These access types allow the following operations:

- Read: Peruse a file, but not change it.
- Write: Read, modify, create, or delete a file.
- Execute: Execute a program.

Directory Access Types

Access types for directories are list, modify, and create. These access types allow the following operations:

- List: List the contents of a directory.

- Modify: Modify the contents of a file in a directory, but not create or delete a file.

- Create: List, modify, create, or delete files and subdirectories in the directory.

A null access type applies to both files and directories. Null access indicates that no access is to be granted.

User Id

Access control assumes that access to the system is controlled by a login process in which every user has a unique user id. This user id is composed of three elements that are specified at login and that remain unchanged during the time the user is logged in. The three elements are:

person.account.mode

person

Name of individual who may access the system.

account

Name of account to which work is charged.

mode

Further identification of the user (optional). Can name the mode in which the user is working (for example, interactive, absentee, or operator).

The elements of the user id can consist only of the ASCII uppercase and lowercase alphabetic characters (A-Z, a-z), digits (0-9), underscores (_), dollar signs ($), apostrophes (') and the uppercase and lowercase graphics whose hexadecimal equivalents are C0-FE (extended character set). Apostrophes and the characters whose hexadecimal equivalents are C0-FE can be used only in the person and account elements. For both the primary and extended character sets, uppercase and lowercase characters are equivalent (for example, JOHN.SYSTEM.AB is the same user id as JohN.sySTEM.ab).

The elements are separated with periods (.). When referencing user ids, you can replace any or all elements by asterisks (*); for example:

```
*.account.mode
person.account.*
*.*.*
```

When an asterisk appears in an element position, it is interpreted to mean any value that may exist. No test is performed to match this element of the user id. For example, if two persons (Smith and Jones) are registered in an account named FILE_SYS, the user id *.FILE_SYS.* matches either person in any possible mode. (The user id *.FILE_SYS.* matches all individuals registered to use FILE_SYS in any mode.)

Access Control

There are four kinds of access control lists: file ACLs, directory ACLs, file CACLs, and directory CACLs.Lists list

File ACL

A file ACL is a type of access control list that applies to a specific file and is considered to be a file attribute. It contains a list of those users who can access the file and their specific access rights (Read, Write, Execute).

Directory ACL

A directory ACL is a type of access control list that applies to a specific directory and is considered to be a directory attribute. It contains a list of those users who can access the directory and their specific access rights (List, Modify, Create).

File CACL

A file CACL is a type of access control list that applies to all files immediately subordinate to a directory. A file CACL is considered to be a directory attribute that applies only to files contained in that directory. A file CACL contains a list of file users and their specific access rights (Read, Write, Execute). Use of file CACLs can save disk space and search time if all or most files in a directory have the same access requirements. A file CACL does not override individual file ACLs set on files in the directory.

Directory CACL

A directory CACL is a type of access control list that applies to all directories immediately subordinate to a directory. A directory CACL is considered to be a directory attribute that applies only to immediately subordinate directories. A directory CACL contains a list of directory users and their specific access rights (List, Modify, Create). Use of directory CACLs can save disk space and search time when all or most subdirectories have the same access requirements. A directory CACL does not override individual directory ACLs set on the subdirectories.

The Create Directory command allows a directory CACL to be established as a global directory attribute. The directory CACL is automatically passed down to subsequently created subordinate directories.

Checking Access Rights

When you reserve a file (through the Get File command or system service macrocall), the File System checks your right to access that file. You are said to be on the access control list if your user id matches an entry on the ACL or CACL in any of the forms noted below.

Universal Access
Universal access (no access restriction) is implied if neither an ACL nor a CACL exists for the file being reserved. If either list is present, it is scanned by access control.

Priority
The checking priority is ACL first, CACL second. If a match is found in the ACL for a fully specified user id (all three elements explicitly stated), the CACL is not inspected. If a match is found on a partially specified user id (one or more elements specified as an asterisk), the CACL is inspected for a more explicitly stated user id. The following list indicates the inspection hierarchy of user id formats in order of decreasing priority. For example, if you are granted access by an ACL entry in format 3, you can be denied access only by an ACL or CACL entry in format 1 or 2.

1. person.account.mode
2. person.account.*
3. person.*.mode
4. person.*.*
5. *.account.mode
6. *.account.*
7. *.*.mode
8. *.*.*

Target File Access
Access is checked only for the target file or directory; the access rights set on directories that may be traversed in reaching the target file are not checked. You may be denied access at some intermediate directory level and still gain access to a subordinate directory or file.

Operator Access
Access control lists do not prevent the system operator from accessing files and directories. It is suggested that physical access to the operator terminal be restricted.

File Concurrency Control

Concurrent Read or Write use of a file among task groups is established by the task group that first reserves the file. Concurrency control performs the following functions:

- Establishes how tasks in the reserving task group intend to access the file (Read, Write, or Execute).
- Establishes what the reserving task group allows other task groups to do with the file.

File Reservation

If the file is already reserved, a task group's concurrency request (reservation) is denied if its intended access conflicts with the access permitted by a prior reserver. The concurrency request is also denied if what it allows others to do conflicts with the access already established by another task group. For example, if a task group reserves the file exclusively, other task groups are denied access. If a task group permits read-only access but does not permit write access, other readers are allowed but writers are denied access.

Get File Command

Concurrency is controlled through the Get File command or system service macrocall. The possible combinations of access intended for the reserving task group and sharability permitted other task groups are given in Table 2-1. Table 2-1 also shows the Get File command arguments that establish the various concurrencies.

Table 2-1. Disk File Concurrency Control

Reserving Task Group	Other Task Groups	Get File Arguments
Read only	Read only (read share)	-ACCESS R -SHARE R
	Read or write (read/write share)	-ACCESS R -SHARE W
Read or write	No read, no write (exclusive use)	-ACCESS W -SHARE N
	Read only (read share)	-ACCESS W -SHARE R
	Read or write (read/write share)	-ACCESS W -SHARE W

System Concurrency

Compiler-generated programs, commands, sort operations, and other system software always request exclusive concurrency for files reserved for users. Since the operator terminal must be reserved with read/write shared concurrency to allow concurrent access by many task groups, it cannot be specified as the path of the -COUT argument of a command that invokes a compiler.

The command-in, user-in, user-out, and error-out files are associated with the Menu Processor and Command Processor (refer to Section 3). If the command-in and user-in files are on disk, they are reserved with read-only shared concurrency; if assigned to a user terminal, they are reserved with exclusive concurrency. You can use File Out commands to specify the concurrency with which the user-out and error-out files are to be reserved.

Access Control/Concurrency Control Relationship

In an environment that employs access control, users must have certain minimum types of access privilege to obtain the specific type of concurrency control they specify in Get File commands or system service macrocalls.

Table 2-2 summarizes the relationship between access control and concurrency control for disk files, disk directories, and disk volumes. (Note that access control does not exist for other types of devices.)

Table 2-2. Access Control/Concurrency Control Relationship

Object	Desired Concurrency	Minimum Access
Disk files	Read Read/write	Read Read/write
Disk directories	Exclusive use Nonexclusive use	List/modify List
Disk volumes	Read or read/write	Modify access to root directory

Shared File Protection (Record Locking)

Record locking is a File System option that provides interference protection so that cooperating users can share and update file data. For example, with record locking in effect there can be many task groups running COBOL applications that Read, Write, and Update record data in the same file or same set of files.

Need for Locks

User applications often employ standard data management services to lock records as they access them. The purpose of the locks is to prevent other users from simultaneously getting access to these records. If other users could access the records, they might get information that is only partially updated or, as a result of some programming decision or error condition, may soon be removed from the file. Also, if there were no locks, two users could update the same records at the same time. In this situation the second updater would inadvertently remove any modifications made by the first updater.

For reasons such as these, record locking is a necessary feature in most file sharing environments. Moreover, in many file sharing environments it is important that more than one lock be simultaneously maintained. For example, an "update" transaction to a parts inventory file may involve multiple record updates--subtracting from some records and adding to others. These multiple record locks may even involve access to multiple files.

Note that record locking is not necessary to prevent a file from being physically corrupted by several applications performing multiple writes. Whether or not record locking is present, the File System maintains indexes and record chains properly so that the file structure is consistent. However, without record locking there is no synchronization, and the file data can be logically corrupted by two or more users who update the same data records. Also, without record locking, data can be viewed in a partially updated or inconsistent state.

Record Locking Function

The record-locking option provides synchronization mechanisms to lock out record data as it is accessed, thereby making the data inaccessible to other applications until it is explicitly unlocked via a cleanpoint call (a call to the ZCLEAN utility in higher-level languages, or $CLPNT macrocall in Assembly language).

Lock Lists

The File System locks records by maintaining lists that describe which file control intervals are locked, who has them locked, and who is waiting for them to be unlocked.

Deadlock

The File System also provides a mechanism to recognize (and signal) whenever a deadlock condition occurs. A typical example of a deadlock is when one user owns (has locked) record A and wants to lock record B while another user already has record B locked and is waiting for record A to be unlocked.

Standard Locking

Two methods of record locking are available. These are standard record locking (-LOCK) and extended record locking (-LOCKX). With standard record locking, if a file is open to allow update operations, records are locked on an exclusive basis. Read, Write, Rewrite and Delete-record calls automatically lock the records accessed until they are explicitly unlocked with a cleanpoint call.

Extended Locking

Extended record locking provides record locking on a shared-read or exclusive-write basis, which allows many simultaneous readers but only one writer at a time. Readers will wait until a writer finishes (issues a cleanpoint call) and vice versa. Read-record operations set read locks; write-record, rewrite-record, and delete-record operations set write locks.

Since, with extended record locking, the type of lock is determined dynamically at each access request rather than once at open time, more readers can access more file data at the same time. Using -LOCKX will improve performance if requests commonly involve reading or searching through large amounts of data. It is especially useful in situations which involve few updates or update conflicts.

Because readers can gain access to to records they can subsequently update, increasing the possibility of deadlock errors, extended record locking requires the -RECOVER argument (see "File Recovery" in Section 6). If two or more readers attempt to update the same data, -LOCKX can issue rollback calls to recover the file data to a known consistent state.

To Set Locking

You can set standard record locking (-LOCK) or extended record locking (-LOCKX) as a permanent file attribute when you create the file, with the Create File (CR) command, or modify its attributes, with the Modify File Attributes (MFA) command. You can also set standard record locking temporarily when you reserve the file for processing with the Get File (GET) command. You can change from locking to no locking as a permanent file attribute with the -NO_LOCK or -NO_LOCKX argument of the MFA command.

Dirty Reader

A file having the -LOCK or -LOCKX attribute can be reserved without record locking through the -NO_LOCK argument of the Get File command. This is a special "Dirty Reader" option that lets you read data even though the data may be locked or may be in the process of being updated by some other user. The consistency and integrity of any data read is not guaranteed. The "Dirty Reader" option is available only on a logical file number (LFN) basis. The associated LFN is read-only, ignores any existing record locks, and cannot set any locks. (LFNs are internal file identifiers associated with file pathnames at the command or source program level; refer to "Logical File Numbers" in Section 5 for further information.)

No-Wait Option

A file with the -LOCK or -LOCKX attribute can also be reserved with the No-Wait option through the -NO_WAIT argument of the Get File command. If the No-Wait option is specified, the File System returns an error status rather than causing you to wait for a record to be unlocked. The No-Wait option is available only on an LFN basis.

Programming Hints

When using record locking, you should be aware of the following points:

- To efficiently use record locking, your applications must be written to be transaction-oriented so that records are not locked for a long period of time (for example, while waiting for terminal I/O) and so that as few records as possible are locked to satisfy the request.
- You should consider using other file integrity features (described in Section 6), especially file recovery, which allows data to be reset (rolled back) to the state it was in at the start of the transaction. In many situations, file recovery is a necessary feature to maintain data integrity in the event of system failure, record deadlock, application failure, terminal failure, and so forth.
- To develop an efficient multiuser application that shares and updates data in standard files, you must examine where and how the application is accessing file data and design the file structure carefully. In addition, you must pay careful attention to error conditions involving data recovery and program or transaction restarts.
- Applications that receive a deadlock notification must be prepared to back out of the current "transaction" and free up the locks they concurrently own. If the file is recoverable, this is done through the rollback call (a call to the ZCROLL utility in higher-level languages; a $ROLBK macrocall in Assembly language).

- When a record is locked, the entire control interval in which the record is contained becomes locked. When defining control interval size, you should consider not only I/O transfer size and memory buffer usage, but also the number of records that may be locked out.

- If the No-Wait option is selected, central processor time must be given up so that other users who have the record locked get a chance to unlock it. You may need to add a "suspend for time interval" function to applications using the No-Wait option to allow other task groups enough time to finish their I/O and unlock records (issue a cleanpoint call).

- Closing a file, issuing a cleanpoint call, or issuing a rollback call frees up all records locked by the task group since the last cleanpoint. If a task group abort occurs, the system issues a rollback call automatically as part of the task group cleanup process. Likewise, if a system failure occurs, the operator will issue the Recover command after the system is restarted. When this command is issued, the File System (in effect) performs a rollback call for all task groups that were active at the time of the failure.

Multivolume Disk Files

In most applications a disk file resides on a single volume. However, there may be situations in which you want to extend a file over more than one physical volume. The need for multivolume files could arise from any of the following:

- You want an endless sequential file capability similar to that available with magnetic tape.

- You want to define a single file too large to be contained on one volume.

- You want to improve access time to a file by spreading the file data over several volumes and/or separating the index portion of an indexed file from the data portion and placing the portions on separate volumes.

File Section A multivolume file is treated as a collection of file sections. A file section is that part of the file that is contained on one volume.

File Set A file set is all of the sections making up the multivolume file.

Multivolume Sets

A multivolume set is a disk file that resides on more than one volume. A volume is identified as being part of a multivolume set when the volume is created through the Create Volume command.

Each multivolume set has a root volume (in which the set begins) and a number of additional volumes. All volumes that are part of the set are called members.

The name of a multivolume set is independent of the names of the volumes it contains. A volume is established as a member of a set when the set name and a sequential member number are specified at volume creation. The root volume is always member number 1.

There are two types of multivolume sets: online and serial. Online multivolume sets are used for all nonsequential multivolume files. They may also be used for sequential multivolume files. Serial multivolume sets are an alternative for large sequential files. They are also used for files that require an endless sequential capability similar to that of magnetic tape.

The types of multivolume sets and files are described in detail below.

Online Multivolume Set

A volume is designated as part of an online multivolume set at volume creation. An online multivolume set has the following characteristics:

- All members of the set must be mounted and available while the set is in use.
- Member volumes, other than the root volume, can be used independently of other members in the set to contain single-volume files and directories.

Online Multivolume File

A file is designated an online multivolume file when it is created under a directory in the root volume of an online multivolume set. An online multivolume file has the following characteristics:

- Can have any UFAS file organization.
- Can be located by any type of pathname.
- Can skip set members when continuing to another volume.

Figure 2-4 illustrates the combination of files and volumes used by a sample online multivolume set. Multivolume files FILEA, FILEB, and FILEC must begin on VOL1. FILEX, FILEY, and FILEZ are single-volume files because they do not begin on VOL1. The pathnames used to access the files are shown at the bottom of the figure.

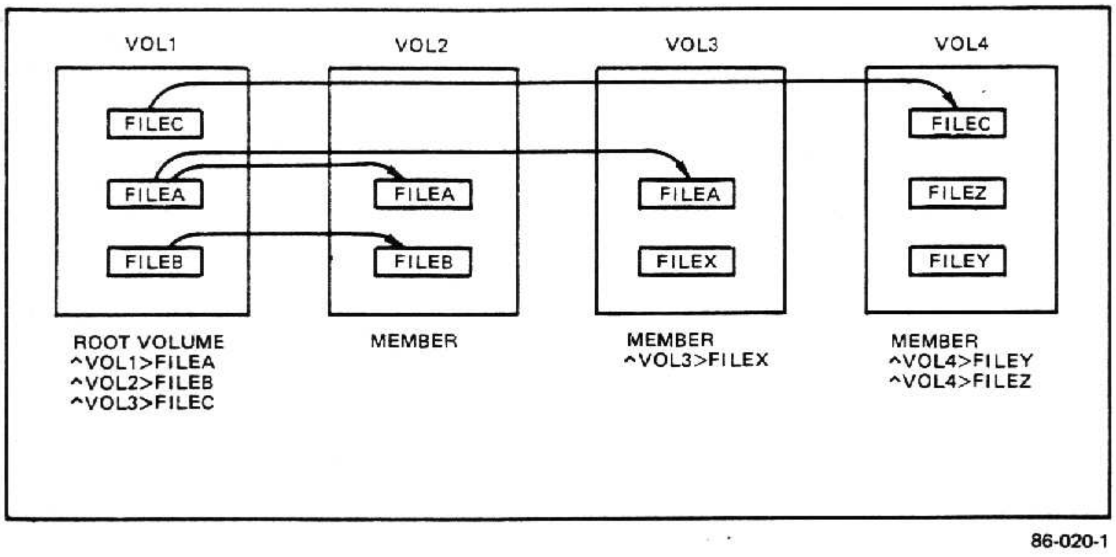


Figure 2-4. Example of Online Multivolume Set

Serial Multivolume Set

A volume is designated as part of a serial multivolume set at volume creation. A serial multivolume set has the following characteristics:

- No member of the set need be mounted until the data on it is required for processing.
- Any member of the set, including the root volume, can be used independently of other members of the set to contain single-volume files and directories.

Serial Multivolume File

A file is designated as a serial multivolume file when it is created in the root directory of a volume in a serial multivolume set. A serial multivolume file has the following characteristics:

- Must be a UFAS sequential file.
- Must be cataloged in the root directory of the volume on which it starts. More than one serial multivolume file can belong to a set, and each such file can begin on a different volume if desired.
- Must be located through a pathname of the form ^volid>filename.
- Must continue serially from one volume to the next (cannot skip set members).

Figure 2-5 illustrates the combination of files and volumes used in a sample serial multivolume set. Serial multivolume file A begins in VOL1. Serial multivolume file B begins in VOL2. Both continue in other volumes of the set. Files C, D, and E are single-volume files. The pathnames by which the files are located are shown at the bottom of the figure.

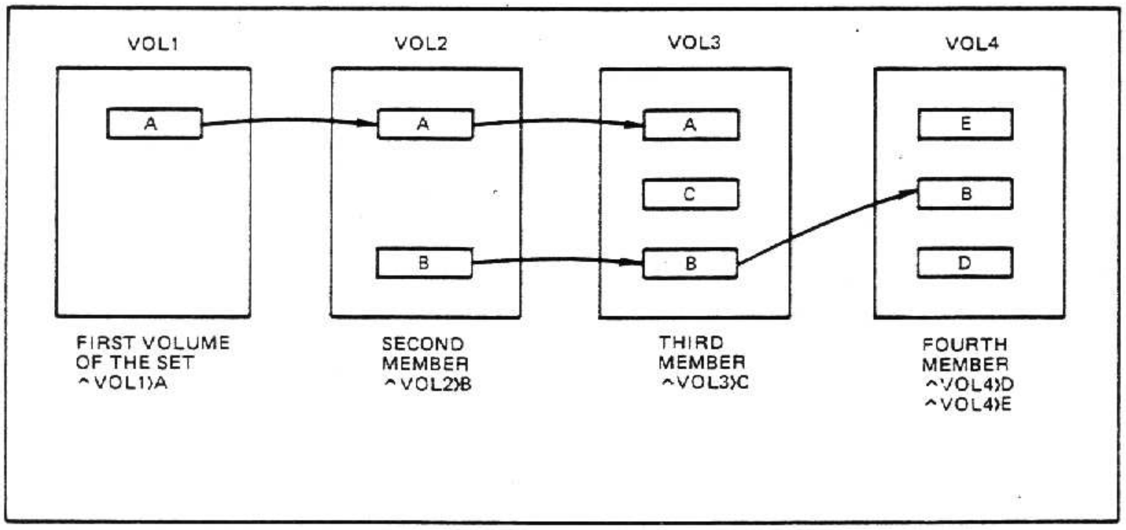


Figure 2-5. Example of Serial Multivolume Set

Disk File Buffering

A buffer is a memory storage area used to compensate for a difference in the rate of data flow, or time of occurrence of events, during transmission of data from one device to another. Buffering is the process of allocating and scheduling the use of buffers. In some applications, overlap of input operations and processing can be achieved by anticipatory buffering, where the next block of data is read into the program's memory area before it is needed. The program can then process records from block n while block n+1 is being read into the memory area.

File Access Levels

Disk files can be processed at either block or record level. In block level access, data is transferred directly between the file and a buffer in your program. Your program must perform all buffer management operations. In record level access, the system assigns disk files to buffer pools when your program opens the files. The system buffering facilities are used to perform all buffer management operations.

Buffer Pools

When a file is opened for Read, Write, Rewrite, or Delete operations, the File System assigns the file to a particular buffer pool. A buffer pool is a collection of buffers that provides a method of conserving memory for disk file access. Buffer pools are designed to:

- Reduce the amount of memory required for buffers by all users.
- Reduce the number of I/O operations in a random access environment.
- Provide more flexibility for shared file applications.

All buffers in a pool are the same size. Any number of files with matching control interval sizes can be assigned to the same buffer pool. A particular file, however, can be assigned to only one pool.

Each buffer in a buffer pool can store a disk control interval. When an application program issues a read instruction and the desired record is not in any buffer in the buffer pool, the next empty buffer in the pool is filled with the control interval containing the record. When all buffers in the pool are filled, an active buffer is selected for the next different control interval according to a least-recent-usage algorithm.

In addition to conserving memory when disk files are accessed, buffer pools eliminate the need for each user to define private buffer areas. One or more system-wide buffer pools should be created at system startup (through a startup EC file; see Section 3). Users who have special buffering requirements can create their own buffer pools for files they reserve exclusively.

Buffer Pool Creation

Each buffer pool is created as either a public or a private buffer pool, and can be considered file-specific or general. Buffer pools are created by the Create Buffer Pool command and deleted by the Delete Buffer Pool command. When creating a buffer pool, you specify the number of buffers it is to contain, the buffer size, and (optionally) the name of the buffer pool.

Public Pools

Public buffer pools are those created by the operator or the system startup EC file. Public buffer pools reside in system memory and are available to all files and task groups. A disk file is assigned to a public pool if its control interval size (specified in the command that creates the file) matches the pool's buffer size.

In many environments, three or four public buffer pools corresponding to three or four common file control interval sizes are sufficient for all performance and buffering needs.

If the system volume is associated with the disk cache processor, heavily used directories, as well as files that are read sequentially, are likely to be resident in the disk cache buffer. Buffer pools for these directories and files may not be needed.

Private Pools

Private buffer pools can be created by each user. Private buffer pools reside in the task group's memory space and are available only for disk files reserved exclusively by that task group. A disk file is assigned to a private pool if the file is reserved for exclusive use and its control interval size (specified in the command that creates the file) matches the pool's buffer size. Private buffer pools should be created only if necessary to meet specific buffering needs. Public buffer pools should be sufficient in most cases.

File-Specific Pools

When you reserve a disk file with the Get File command, you can specify the number of buffers (using the -BUF argument) to be used when accessing the file. When the file is opened, a buffer pool is automatically created for use only by that file. This file specific pool is created in the task group's memory if the file is reserved exclusively, or in system memory if the file is reserved as shareable. The -BUF argument should be used carefully since it prevents a file from being assigned to a public or private buffer pool.

Buffer Pool Optimizing

The File System collects a set of statistics on the use of each buffer pool. The installation can use this information to optimize disk I/O operations. Buffer pool statistics are obtained through the Buffer Pool Status and Buffer Pool Information commands. The Buffer Pool Status command provides a summary of the public or private buffer pool status. The Buffer Pool Information command provides a detailed status report on a particular buffer pool.

The installation should analyze applications and their associated file usage to fully utilize the advantages offered by buffer pools. Only a limited number of control interval sizes should be allowed for user files. In general, buffer and control interval sizes should be chosen to evenly distribute high and low activity files over the various buffer pools, thus reducing the amount of contention in the pools. The initial determinations will provide an acceptable level of performance and provide the basis for further analysis.

The Adjust Buffer Pool command can be used to temporarily alter the number of buffers in a private buffer pool. Once the most efficient buffer pool size has been established, it should be permanently fixed through the Create Buffer Pool (CBP) and Delete Buffer Pool (DBP) commands. In general, the system START_UP.EC file should specify a CBP command for directory I/O, with the -DIR argument, and for space allocation, with the -ALLOC argument. Also include one or more CBP commands for commonly accessed disk files. This will result in higher performance, less system memory fragmentation and more memory available to users. This may require configuration of a larger system memory pool and smaller user memory pools.

MAGNETIC TAPE FILE CONVENTIONS

The magnetic tape file conventions discussed in the following paragraphs include file organization, naming conventions, pathnames, and buffering operations.

Magnetic Tape File Organization

The following information applies only to 1/2-inch, 9-track magnetic tapes.

Magnetic tape supports only the sequential file organization. Fixed- or variable-length records can be used. Records cannot be inserted, deleted, or modified, but they can be appended to the end of the file. The tape can be positioned forward or backward any number of records.

Blocks

The unit of transfer between memory and a tape file is a block. Block size varies depending on the number of records and whether the records are fixed or variable in length.

A block can be treated as one logical record called an "undefined" record. An undefined record is read or written without being blocked, unblocked, or otherwise altered by data management. Spanned records (those that span across two or more blocks) are supported. No record positioning is allowed with spanned records.

Labeled Tapes

A labeled tape is one that conforms to the current tape standard for volume and file labels issued by the American National Standard Institute (ANSI). The following types of labeled tapes are supported:

- Single-volume, single-file
- Multivolume, single-file
- Single-volume, multifile
- Multivolume, multifile.

Unlabeled Tapes

The following types of unlabeled tapes are supported:

- Single-volume, single-file
- Single-volume, multifile.

Magnetic Tape File and Volume Names

Allowable Characters	Each tape file and volume name in the File System can consist of the following ASCII characters: Uppercase alphabetics (A through Z), lowercase alphabetics (a through z), digits (0 through 9), exclamation point (!), double quotation marks ("), dollar sign ($), percent sign (%), ampersand (&), apostrophe ('), left parenthesis ((), right parenthesis ()), asterisk (*), plus sign (+), comma (,), hyphen (-), period (.), slash (/), colon (:), semicolon (;), less-than sign (<), equal sign (=), question mark (?), and underscore (_).
Forming Names	Any of the characters defined above can be used as the first character of a file or volume name. The underscore character can be used as a substitute for a space. If a lowercase alphabetic character is used, it is converted to its uppercase counterpart ("DATA", "Data" and "data" all refer to the same file).
Name Length	The name of a tape volume can be from 1 through 6 characters in length. Tape file names can be from 1 through 17 characters.

Magnetic Tape Device Pathname Construction

As previously mentioned, magnetic tape volumes can be labeled or unlabeled (refer to "Magnetic Tape File Organizations" above).

Unlabeled Tape Pathnames

You must use a tape device pathname when referring to an unlabeled tape. The general form of a tape device file pathname is:

```
!dev_name
```

where dev_name is the symbolic name defined for the tape device at system building time.

Labeled Tape Pathnames

You can refer to labeled tapes either by the tape device pathname convention or by the tape volume id convention.

Tape Device File

The tape device file pathname convention is:

```
!dev_name>vol_id[>filename]
```

where dev-name is the name of the tape device as specified at system building time, vol_id is the name of the tape volume, and filename is the name of the file on the volume. This convention requires that the volume be mounted on the specified device.

Tape Volume Id

The tape volume id convention is:

```
-vol_id[>filename]
```

where vol_id is the name of the tape volume and filename is the name of the file on the volume. This convention allows the volume to be mounted on any available tape device.

Automatic Magnetic Tape Volume Recognition

Automatic volume recognition dynamically notes the mounting of a tape volume. This feature allows the File System to record the volume identification in a device table, thus making every tape volume accessible to the File System software.

Magnetic Tape Buffering

The -BUF argument of the Get File command can be used with magnetic tape files to reserve one or two buffers. If -BUF is not used, the File System attempts to allocate two buffers. If two buffers are allocated, the File System does "double buffering." When the tape file is being read, the File System unblocks one buffer while an anticipatory read is done into the other buffer. Similarly, when the tape file is being written, the File System blocks records into one buffer while a previously filled block is written out of the other buffer. This allows application code to execute in parallel with I/O transfers.

UNIT RECORD DEVICE FILE CONVENTIONS

Unit record devices (card readers, card punches, printers, terminals, and paper tape reader/punches) are used only for reading and writing data. They are not used for storing data, and thus do not require conventions for file identification and location.

Unit Record Device Pathname Construction

The pathname of a unit record device consists of the symbolic device name defined at system building time preceded by an exclamation point (!). The pathname format is:

```
!dev_name
```

where dev_name is the symbolic device name of the unit record device.

Unit Record Device Buffering

All printers and most interactive terminals are provided with one File System buffer. (The operator terminal cannot be buffered.) By providing a File System buffer, application code can execute in parallel with I/O transfers.

Tab Stops

All printers and all terminals (except the operator terminal) have a tabbing capability through software that converts the tab into spaces. Default tabulation stops are set at position 11 and at every tenth position thereafter for the line length of the device.

Unit Record Read Operations

When an application task issues a logical read to a File System buffered device, one of the following actions occurs:

- If the buffer is full from a prior anticipatory read, the data in the buffer is transferred into the application task's area and a physical I/O transfer (an anticipatory read) is performed in parallel with continued execution.
- If the buffer is not full, task execution stalls until the anticipatory read is completed.

The timing of the initial anticipatory read performed for the card reader is different from that of the interactive terminals; for other read actions it is the same.

Card Reader

Immediately after the Open is complete, the File System performs an asynchronous anticipatory read into the system buffer while the application continues execution. All Open calls are synchronous.

Interactive Terminal

The anticipatory read allows an application to control input from more than one interactive terminal, each of which represents a data entry terminal. By testing the status of the system buffer before a Read or by checking for the appropriate status return after a Read, the application will not be stalled if the terminal operator is not present at the time of the Read request. Instead, the application can continue to poll other terminals.

File Status

Immediately after the Open is complete, a physical connection is made while the application continues execution. Depending upon the language the application is written in (for example, FORTRAN or Assembly language), it may be able to check the status of the Open to see if a Read can be issued without stalling application execution. The File System issues an asynchronous anticipatory physical read when the status check following the physical connect is complete. The file status remains busy until the physical read is done and the system buffer is full. At this point, the file status is "not busy" (the anticipatory read is successfully completed), and the application can issue a Read with the assurance of receiving data immediately.

Synchronous Read

If at any point after the Open is issued, the application issues a Read before the physical connect and anticipatory read have been completed, the Read is synchronous and further central processor execution is stalled on the application until the anticipatory read is complete.

Buffer Status

To avoid stalling on a Read or to avoid status check looping to test the input buffer status, applications should put themselves in the wait state, thus making the central processor available for lower priority tasks.

COBOL Programs

After the Open, an application written in COBOL must issue Read requests. The application will be put in the wait state if it is executing I/O statements in synchronous mode. Otherwise, the COBOL run-time package performs status checks and returns a 9I status until successful completion. The program can either loop on the Read or continue other processing.

Unit Record Write Operations

A buffered write operation to a unit record device works on behalf of the application program in the same logical manner as a read operation. The program is permitted to execute in parallel with the physical I/O transfer to the device. To achieve this parallel processing, no special operation occurs on an Open call and no distinction is made between interactive and noninteractive devices.

Status Check

Each Write call is completed by moving data from the application buffer to the File System's buffer (performing any detabbing, if requested), initiating the transfer, and returning control to the application program. If the program performs a second Write while the system buffer is still in use for a previous transfer, the application is stalled until the buffer is available and new data is moved into it again. The application can avoid stalling execution when writing to an interactive terminal by doing one of the following:

- Checking the status of the system buffer before issuing the Write to see if the interactive terminal is still in use.
- Testing for a particular status return after the Write.

Write with Read

If a Write call is issued while data is being entered into the system buffer (because of a Read), the following sequence of events takes place:

- The Read is allowed to complete.
- Input data is saved in the system buffer.
- A synchronous Write is reissued by the File System.
- Output data is transferred directly from the application buffer.

Tabs

Note that tab characters are not expanded into spaces.

Errors

If a physical I/O error occurs while data is being transferred from the system buffer to the device, you must be aware that the error occurred on the previous write operation. Furthermore, if any type of error occurs, the application program may need to have saved (or be able to retrieve) the data record so that it can be repeated.

REMOTE FILE ACCESS

Remote file access is a File System facility that allows applications to access remote data as if it were local. Remote objects such as files, volumes, magnetic tapes, and printers physically reside in some other computer system (node) but, through remote file access, appear to be attached to your system. The remote file access facility captures references to remote objects and interfaces with the appropriate networking software (DSA for example) to get the desired function performed remotely.

Statements

When accessing data at another node, you may employ any File System function through macrocalls or higher-level language I/O statements. No special naming conventions are necessary. You supply the same kind of pathname you would to access local data. The File System checks to see if the object identified in the pathname is online (located on your node). If the object is not online, the File System checks a remote file catalog to see if the object is located at some other computer node.

Commands

The Remote File Catalog command is used to manage catalog information. This command allows a system operator or administrator to define, update, and display information about remote and local objects, nodes, and networking software.

The Remote File Access command is used to initiate the remote file access facility. This command allows the system operator or administrator to start the facility, retrieve network status information, and open and close connections between nodes.

Remote File Catalog

Each node has its own remote file catalog to identify objects it can reference through remote file access. The catalog contains only those objects of interest to the node. A remote file catalog contains information about both remote and local objects.

Remote Objects

Remote object information consists of a list of remote volumes and devices along with the node at which they are currently located. The File System allows you to define your own names for remote objects. For example, the line printer known as LPT04 in NODE3 can be cataloged as LPT01 in NODE1. Any reference to LPT01 in NODE1 will result in a search of the catalog and subsequent use of the printer through the remote file access services. The catalog can be updated dynamically by an operator or system administrator to configure new remote devices or to reconfigure existing ones.

Local Objects

Local object information consists of a list of your volumes and devices that can be accessed from other nodes. This information is used by the other nodes to verify the existence of what is to them a remote object. It enables the File System to automatically update the catalog when volumes are moved from one system to another.

Volume Identification

Disk volumes can be exchanged or moved from one node to another. The remote file catalog contains enough information to uniquely identify a remote disk volume and to recognize when it has moved to another node. This information consists of:

- A Node of Birth (NOB) field identifying where the volume was originally cataloged for remote access.
- A Date of Birth (DOB) field identifying the date and time the volume was originally cataloged for remote access.
- A Node of Residence (NOR) field identifying the node where the volume is currently located (cataloged as a local object).
- A Node Migration Number (NMN) field identifying the number of times a volume has moved from one node to another.

The remote file catalog maintains the relationship between a local name, its current location (NOR), and the actual name. When a volume is moved to another node, only its NOR is changed, no change is made to the local name.

When a node connection is established, the two systems involved exchange local object information. A node ignores any volume information received that is out of date with respect to what it already has in its catalog. If a node has been off-line for some time, any old information it has will be discarded and any new information it received will be factored in.

Catalog Creation

Establishing a remote file catalog is usually a one-time operation. The steps involved in setting up the catalog are as follows:

1. Create the catalog.

   You create the catalog by using the Remote File Catalog (RFC) command with the -CAT argument. This step is performed only once.

2. Catalog a node for remote access.

   You define the nodes with which you are to communicate by using the RFC command with the -NODE argument. This step is repeated once for each node.

3. Catalog a local object to be accessed remotely..

   You define a local object that is to be accessed from other nodes by using the RFC command followed by the local pathname of the object. This step is repeated once for each local object to be addressed remotely. Any device to be cataloged must be configured on your system. A disk volume to be cataloged must be mounted.

4. Enable the remote file access facility.

   You invoke the remote file access facility by using the Remote File Access (RFA) command with the -STARTUP argument. This step configures and initializes the facility for communicating between nodes. It must be performed at the local node and each remote node whose objects are to be cataloged.

5. Establish communication with a remote node.

   You establish communication with the remote node by using the RFA command with the -OPEN argument. This step must also be performed at each remote node whose objects are to be cataloged.

6. Catalog a remote object to be accessed locally.

   You define a remote object that is to be accessed from your node by using the RFC command followed by the local name to be used to reference the object and the name of the node at which the object is located. If you wish to define a local name that is different from the name of the object as it is known at the remote node, you must use the -ROBJ argument. This step is repeated once for each remote object to be addressed locally. Note that communication must have been established with the remote node through the RFA command.

Initiating Remote File Access Operations

Once a remote file catalog is set up, only two steps must be performed on a day-to-day basis before you can access remote devices and data (assuming that the network is up).

1. Enable the remote file access facility.

   You and the nodes with which you are to communicate must issue the RFA command with the -STARTUP argument.

2. Open the remote nodes.

   You must issue the RFA command with the -OPEN argument for each node with which you are to communicate. Those nodes that are to access objects at your node must also issue the RFA command with the -OPEN argument.

After you have enabled the remote access facility and opened the remote nodes, you can perform any operation on the remote data that you would perform on local data. Whether you use system commands or your own application programs, the data will appear to be located at your node.

Remote File Access Security

The following paragraphs describe the way in which the remote file access facility handles access control, record locking, and data commitment.

Access Control Lists

Access control lists define which users have access to data and what kind of access they have. When files are accessed remotely, the same level of file protection exists as when files are accessed locally. If a file is protected by an access control list, no local or remote user can access the file unless the user is given permission through the access control list.

Record Locking

Record locking prevents other users from simultaneously getting access to records that you are accessing. In many applications record locking involves multiple record locks on multiple files and, in networking environments, may involve locks to multiple files in multiple nodes.

A typical deadlock condition can occur if one user has locked some records and is trying to lock others while another user has these other records locked and is trying to lock the records already locked by the first user. The File System on your computer node knows about the record locks on local data files and is able to detect deadlocks. Since the File System does not know about record locks on remote data files, it prevents deadlocks from occurring by using a time stamp algorithm.

Users are assigned time stamps when they start to access remote data. The time stamps are passed to remote nodes by the remote file access facility. At a remote node, a user may only wait for records that are held by younger users (users whose time stamp is later). If the application attempts to lock a record that is already locked by an older user (a user whose time stamp is earlier), it receives a "deadlock has occurred" return status. The application must then abort, backtrack, or restart. If a record is held by a user who is local to that node, the local user is always considered the younger.

Data Commitment

For purposes of data integrity, an application that accesses and updates remote (and local) data may be structured in phases known as commitment units. The end of a commitment unit is a point at which the user is willing to commit changes to the data base. This type of application is said to be transaction oriented. It may complete successfully (commit) at a program-defined commitment point or it may fail (abort), in which case any updated data must be returned to its initial pre-transaction state. To ensure reliability, the transaction must either complete in its entirety or not complete at all.

In remote file access, data commitments are performed in two phases: precommit and commit.

- Precommit - All data is recorded on disk with an indicator to show that the data is in a precommitted state. This step is done locally. Remote file access then sends messages to precommit data at all remote nodes.

- Commit - Once messages have been received from all affected remote nodes, indicating that all data is in the precommit state, local data is committed (unlocked and made available to other users). Another round of messages is then sent via the remote file access facility to commit data in the remote nodes.

If a system or node failure occurs in any intermediate step, there is enough information available so that, on restart, a decision can be made to commit or recover the data. More detailed information on file recovery is presented in Section 6.