Unisys e-@ction ClearPath Enterprise Servers

I/O Subsystem
Programming Guide

ClearPath MCP Release 7.0 SSP1

March 2002

Printed in USA
8600 0056–408

Unisys e-@ction ClearPath Enterprise Servers

I/O Subsystem
Programming Guide

UNISYS

ClearPath MCP Release 7.0 SSP1

March 2002

Printed in USA
8600 0056–408

Correspondence regarding this publication can be e-mailed to **doc@unisys.com**.

Unisys e-@ction
ClearPath Enterprise
Servers

I/O Subsystem

Programming Guide

**ClearPath MCP
Release 7.0 SSP1**

8600 0056–408

Unisys e-@ction
ClearPath
Enterprise
Servers

I/O Subsystem

**Programming
Guide**

**ClearPath MCP
Release 7.0
SSP1**

8600 0056–408

Bend here, peel upwards and apply to spine.

Contents

Section 3. Using Disk and CD-ROM Files in a Program

Section 4. Using Tape Files in a Program

Section 5. Using Printer Files in a Program

Section 6. Using Remote Files in a Program

Section 7. Using Card Files in a Program

Section 8. Using Operator Display Terminal (ODT) Files

Section 9. Accessing and Creating Files Using Distributed File Services

Section 10. Using Direct I/O Files

Section 11. Using HYPERchannel (HY) Files

Section 19. Exchanging Data

Section 20. Closing a Dialogue

Section 21. Understanding Port Services

Section 22. Using BASICSERVICE

Section 23. Using OSINATIVESERVICE

Section 24. Using OSISESSIONSERVICE

Section 25. Using BNANATIVESERVICE

Section 26. Using TCPIPNATIVESERVICE

Section 27. Using TCPNATIVESERVICE

Section 28. Using NETBIOSSESSIONSERVICE

Section 29. Understanding Virtual Files

Section 30. Using the REDIRSUPPORT IOHANDLER Library

Section 31. Using the STREAMIOH IOHANDLER Library

Appendix A. Device Types and Associated File Attributes

Appendix B. Format of Pack Labels

Appendix C. Disk File Headers

Appendix D. Format of Library Maintenance Tapes

Figures

Tables

Tables

Section 1
Introduction and Understanding File Handling

About This Guide

This guide explains what the I/O subsystem does and how file attributes interact with the I/O subsystem to define a specific file type. Additionally, the guide describes how language-specific I/O statements are used to manipulate file attributes in a program.

This guide also provides a general discussion of how file attributes and I/O statements work together to define the characteristics of files, but it does not explain how to manipulate a file in a specific language. Refer to the language programming reference manual for specifics about I/O statements.

This guide is intended to be used by any programmer who needs to understand how to describe the characteristics of a file in a program.

Use this guide in conjunction with the *File Attributes Reference Manual* and the programming reference manual of the language that is to be used. This section through Section 12 presents basic information about how the I/O subsystem and file attributes work together, as well as information about basic programming techniques. Additionally, information about how to describe a specific file in a program is presented. Sections 13 to 28 present information about programming for inter-process communication by using port files.

About POSIX Files

POSIX files are declared and manipulated differently from standard files declared under the master control program (MCP). Attribute descriptions in this manual refer to non-POSIX files unless otherwise specified.

Files, Records and Directories

What Is a File?

Before considering basic programming for files, you need to understand the characteristics of a file. A file is an ordered group of related records that exist apart from the program. A file is defined in a program that uses that file, or at least the file description is known to that program. A file description in the data division of a COBOL program is an example of defining a file in a program.

What Is a Record?

Most files have a particular structure that the program can determine called a *record*. A record is a group of logically related items of data in a file that are treated as a unit. A record within a file contains data made up of characters, binary data, or both. The I/O subsystem gives to the program or takes from the program one record at a time. The program then handles the subdivisions of the record.

Records of a file can all be the same length, known as *fixed-length records*, or of varying lengths, known as *variable-length records*. Length information must be declared or implied by the program and made available to the I/O subsystem. If the length is variable, the I/O subsystem must be able to obtain the actual length of each record so that it can process one record at a time, regardless of length, either from the record itself, or from the program processing the record. From the viewpoint of the program, records are ordered in some way. In the simplest case, they follow each other sequentially until the end of the file. In that case, the program is interested only in accessing the next record in sequence.

Files that contain only streams of bytes are not structured with records.

What Is a Directory?

A directory is sometimes referred to as a folder that contains files. There are three types of directories on the ClearPath NX servers and A Series systems. Refer to the *System Operations Guide* for more information about directories.

Physical and Logical Files

A file is viewed in two ways: as a physical file and as a logical file.

Physical Files

A file that exists on a recording medium is known as a *physical file*.

Except for a disk file with a FILESTRUCTURE value of STREAM, a physical file contains a group of physically adjacent records that are referred to as a *block*. Such a block can be transferred to or from the physical file as a group.

Note: *The term physical file is sometimes used in this manual to refer to the physical device rather than the data stored on its recording medium. The context usually makes clear which meaning is intended.*

In the case of disk files, a physical file can be either permanent or temporary. A permanent file is visible to all running programs and to the system operator. Access to a permanent file can be limited by security facilities. A permanent file remains visible until it is deliberately purged or, if it resides on removable media, until the operator dismounts the media from the system. A temporary file is one that exists only at the time of its original creation—it is of no further interest to any program. An example of a temporary file is a disk file that is used only as an intermediate step in a process. A temporary file is private to the program that creates it, has no visibility to the general system, and exists for the I/O subsystem only while the logical file that created it remains assigned. A newly created file becomes a permanent file when a program requests that the file be saved by closing the file with a disposition of lock, downsizearealock, or crunch, or by using the PROTECTION file attribute. At such a time, the I/O subsystem enters the name of the file in the disk directory. Refer to Section 2, "Understanding Programming for Files," for information about closing files and their associated dispositions.

Logical Files

A file that exists within a program is known as a logical file.

From the viewpoint of block-structured languages such as ALGOL and Pascal, a logical file exists only within the program block where it is declared or within blocks to which it has been passed as a formal parameter. A logical file has no inherent properties until it is described by file attributes or until it is associated with a physical file. A physical file inherits properties from the file attributes of the logical file that creates it. Multiple logical files can be associated with one physical file, and the attributes of those logical files need not be identical in all cases.

A logical file can be in one of four states:

- Open-assigned
- Closed-unassigned
- Closed-assigned (also known as closed-retained)
- Open-unassigned

Before data can be transferred between logical and physical files, the logical file must be open and the physical file must be assigned to the logical file. This assignment can be accomplished explicitly by opening the logical file with an OPEN statement, by means of the AVAILABLE attribute, or, in some languages, by invoking an I/O statement.

Additional File Characteristics

A file can be identified as an optional file by using the syntax of some languages or by assigning the OPTIONAL attribute a value of TRUE. When a file is identified as an optional file, an OPEN operation can leave the logical file unassigned to a physical file and proceed with processing.

A file can also be identified as a file that never executes physical I/O operations by assigning the DUMMYFILE attribute a value of TRUE. This feature is helpful for debug files or when the output of the program is not needed.

Finally, a file can be identified as an exclusive file by using the syntax of some languages or by assigning the EXCLUSIVE attribute a value of TRUE. When a file is identified as an exclusive file, an attempt to open a file that is currently in use results in the physical file being attached to the logical file, the logical file waits and remains closed until the file can be used only by your program. Similarly, if your program is using a file exclusively, other programs that try to open it wait until your program closes the exclusive file.

Opening a file explicitly does not cause data to be transferred between the logical and physical files, and the logical file can be closed without any I/O being performed upon it. The logical file can be closed with retention, which leaves the physical file assigned, or it can be closed with release, which severs the connection between the logical and physical files.

Naming a File in the MCP Environment

The MCP environment supports two file-naming conventions: the traditional file naming convention and the long file naming convention. This section describes the traditional naming convention. Even on systems which have enabled long file names, you should use traditional file names whenever possible.

You name a file in the MCP environment with a series of up to 12 nodes separated by slashes (/). The file name can be preceded by a usercode enclosed in parentheses or can be preceded by an asterisk (*). Each node can contain a 1- to 17-character identifier with the following characteristics:

- Any combination of EBCDIC uppercase letters A through Z or EBCDIC digits 0 through 9. Additionally, a hyphen (-) or an underscore (_) can be included in the identifier, but neither of those characters can be the first character.
- An identifier enclosed in quotation marks ('' ''). The EBCDIC characters can be any character that has a hexadecimal code greater than or equal to a hexadecimal 4"40" (space) and cannot be an EBCDIC character quotation mark.
- If the file is a disk file, a keyword that identifies a system-supplied character string preceded by a displayable character that is specified with the UNIQUETOKEN file attribute value. Refer to the FILENAME attribute in the *File Attributes Reference Manual* for more information about valid keywords.

The following are examples of valid file names in the MCP environment:

- ACCOUNTING/RECEIPTS/021689
- PERSONNEL/''EMPLOY.LIST''
- (KELLY)DOCUMENTS/CLIENT/6
- *SYSTEM/LOGANALYZER
- ACCOUNTING/''@MIXNO''/''@JOBNO''/RECEIVABLES (Disk files only)

The MCP environment provides an optional long file name feature that allows greater flexibility when naming disk files. When this feature is enabled, disk files can be named using names of up to 20 nodes, each of which contains up to 215 characters. This feature is primarily intended for use on ClearPath NX servers to allow interoperation with workstation programs that create files with names longer than those supported by the traditional MCP environment file system. Refer to the *System Operations Guide* for information on the benefits and limitations of the long file names features.

For POSIX, a pathname attribute PATHNAME can be set to provide an alternate, standards-based way of naming the system disk file. Refer to the *System Operations Guide* for more information on disk file naming conventions using POSIX pathnames.

A tape file name in the MCP environment has the same characteristics as a file name in the MCP environment, except that the I/O subsystem uses only the first and the last nodes of the name. If more than one file resides on a tape, all the files must have the same first node, known as the multiple file ID (MFID). The second node is known as the file ID (FID). The following are examples of valid tape file names in the MCP environment:

- RECEIVABLES
- ACCOUNTS/MASTER
- STUDENTS/GRADES
- STUDENTS/CLASSES

A port file name has a single 1- to 17-character node. That node can contain an identifier with the following characteristics:

- Any combination of EBCDIC uppercase letters A through Z or EBCDIC digits 0 through 9.
- An identifier enclosed in quotation marks ('' ''). The EBCDIC characters can be any character that has a hexadecimal code greater than or equal to a hexadecimal 4"40" (space) and cannot be an EBCDIC character quotation mark.

The following are examples of valid port file names:

- RESERVATIONS
- RESERVATION_REQ
- ''RESERVATION_REQ''

Refer to the FILENAME attribute in the *File Attributes Reference Manual* for more information about file names.

Identifying Files on Other Systems

When you are accessing a non MCP environment system through a network, enclose the name in apostrophes ('). Before the MCP environment system sends the file name information to the remote host, the apostrophes are removed and only the characters are sent to the remote host. The name can be up to 250 EBCDIC characters long, but cannot contain an EBCDIC apostrophe character or any EBCDIC character that has a hexadecimal code less than 4"40". A pair of apostrophes can be used to specify a single apostrophe in the file name.

The following are examples of valid foreign host file names:

- 'MY\FILE\031789'
- 'A:ACCOUNTS.PAY'
- '[Sys]<Sys>InstallSpl.Run'
- 'ACCOUNTING*PAYABLES.'

Understanding the Functions of the I/O Subsystem

In the MCP environment, you, as a programmer, do not need to handle the details of controlling the peripheral devices, nor do you need to be concerned with the connections to the peripheral units. The I/O subsystem handles those details for you and allows dynamic file definition; that is, the precise nature of a file need not be fully defined in the program using the file.

Some of the tasks the I/O subsystem is responsible for are

- Making a file assignment, namely, establishing a connection between the logical file of a requesting program and the corresponding physical file
- When a new file is to be created, finding and providing storage space or providing the address of a peripheral device of the requested or acceptable kind—for example, connecting an available tape drive
- Acting as an intermediary between the logical file and the physical file associated with it
- Using the disk subsystem to maintain the disk directory
- Reading and keeping track of all label information on physical files that have been loaded by the operator
- Automatically writing the necessary labels when a new file is created, including the external file name and other label information
- Checking for consistency between the specified attributes
- Checking the results of physical I/O operations, converting any errors to a standard format, and communicating the results by means of file attributes
- Dealing with the program, one logical record at a time, and executing physical I/O operations only when necessary

- Using buffers to smoothly expedite the flow of I/O operations. A *buffer* is an intermediate storage area, under control of the I/O subsystem, which is used to store data in transit between the physical file and the user work area. Typically, two buffers are used so that one can be dedicated to a peripheral transfer while the other is available for logical record operations

Understanding File Attributes

The I/O subsystem can define files dynamically because of file attributes. File attributes act as a communication channel between the program and the I/O subsystem. Sometimes a program communicates to the I/O subsystem by modifying an attribute value. At other times the program interrogates an attribute to determine the conditions under which the I/O subsystem is operating.

File attributes allow the program to accomplish the following tasks:

- Identify a file
- Describe the structure of a file
- Identify the status of a file
- Specify the security level of a file
- Control a printer file
- Control the translation of character sets
- Determine the current status of a file
- Allow interprocess communication
- Access a file on a remote host

If you are using ALGOL, you are responsible for modifying and interrogating file attributes. However, compilers for some languages, including COBOL74 or COBOL85, set the attribute values for you. You are responsible for interrogating any attribute that you might be interested in.

File attributes enable you to write a program that is not limited to a particular configuration nor bound to any hardware device, because the file attribute values declared in a program can be changed in any one of the following manners:

- The value is changed at compilation time
- Before the program is run, an operator changes the value by using the file equation capability of WFL
- The value is changed by your program while it is executing
- The value is changed by using the FA (File Attributes) system command

Modifying an attribute value is referred to as attribute assignment. Interrogating an attribute is referred to as attribute interrogation.

File equation is a mechanism for performing attribute assignment when a program is initiated or compiled. For example, as a program is initiated, the KIND attribute can be given a different value than the value already specified in the file declaration. A program can be written to handle any file of a specific type, and then a user of the program can indicate a particular file by file-equating the FILENAME attribute appropriately. For information about file equation and task initiation, refer to the *WFL Reference Manual*, and for information about the particular file attributes and their values, refer to the *File Attributes Reference Manual*.

Example

The following example demonstrates the use of file attributes. In this example, the WFL deck compiles and runs the ALGOL program and uses file equation in the process.

The ALGOL program symbolic whose file name on disk is FILE/PROGRAM is as follows:

```
BEGIN
FILE
    F(KIND=TAPE,MAXRECSIZE=90,BLOCKSIZE=360,NEWFILE=TRUE,     (1)
      FILENAME="TEST.",FRAMESIZE=8);
ARRAY
    A[0:14];

F.KIND := VALUE(DISK);                                        (4)
(4)
OPEN(F);
    (The entry of an FA (File Attribute) system command by the
     operator.)                                               (5)
REPLACE POINTER(A) BY " " FOR 90;
REPLACE POINTER(A) BY "THIS FILE'S FILENAME IS:  ", F.FILENAME;
F.SYNCHRONIZE:=VALUE(OUT);                                    (6)
WRITE(F,90,A);
CLOSE(F,CRUNCH);
END.
```

The WFL job is as follows:

```
?BEGIN JOB FILE/EXAMPLE;
COMPILE OBJECT/FILE/PROGRAM ALGOL LIBRARY;
  COMPILER FILE CARD(KIND=DISK,FILENAME=FILE/PROGRAM);
    FILE F(BLOCKSIZE=180,FILENAME=TEST2);                     (2)

RUN OBJECT/FILE/PROGRAM;
  FILE F(BLOCKSIZE=2520,NEWFILE=FALSE,FILENAME=TEST1);        (3)
?END JOB
```

The following file attribute actions are taken in the preceding example code:

Notation Number	Explanation
(1)	The values assigned in this statement are stored in the file declaration of the code file.
(2)	The values assigned at compilation time are stored with the code file as a compile-time file equation.
(3)	The values assigned when the program is run are stored with the job information.
(4)	The file attribute values stored at (1), (2), and (3) are applied to the file, with the values assigned in (2) superseding the values of BLOCKSIZE and FILENAME assigned in (1) and the values assigned in (3) superseding the values of BLOCKSIZE, FILENAME, and NEWFILE assigned in (1) and (2). Finally, the value of DISK is assigned to the KIND file attribute.
(5)	If an operator entered an FA system command because no file was available with the correct name, the value or values of any attributes included in the FA command override any values already assigned to those attributes at that point.
(6)	The SYNCHRONIZE file attribute value is changed to OUT as the program is running.

After the program attribute assignment statement *F.KIND := VALUE(DISK)* is executed, the following attributes take on the indicated values:

- The KIND attribute has a value of DISK.
- The MAXRECSIZE attribute has a value of 90 characters.
- The BLOCKSIZE attribute has a value of 2520 when the file is opened.
- The NEWFILE attribute has a value of FALSE.
- The FILENAME attribute has a value of TEST1.
- The FRAMESIZE attribute has a value of 8.

During execution of the OPEN statement, if the permanent physical file with the file name of TEST1 does not exist on disk when the assignment to the physical file is attempted, the program waits for an answer to the NO FILE message. In other words, the program displays the message "NO FILE TEST1" and waits for a response from either an operator or a programmer, or for the file with the file name TEST1 to be created on or copied to the appropriate disk device.

If an FA FILENAME = TEST/FILE system command is entered for the program, all the file attributes listed previously now have the values indicated in the previous list, except that the FILENAME attribute value is now TEST/FILE. When a physical file with a file name of TEST/FILE exists on disk, the OPEN process proceeds normally, assigning the logical file F to the physical file TEST/FILE on disk, setting up the logical file, and marking the logical file as open.

Section 2
Understanding Programming for Files

To read, write, and update information in a file, you specify file attribute values and invoke I/O statements in your program. The various programming languages provide ways of specifying file attribute information. If you do not explicitly specify file attribute values, the system provides reasonable defaults for required file attribute values. Additionally, each language has specific I/O statements that invoke the I/O actions.

You can modify most file attributes when the program is initiated from WFL. Since this is true, your program only has to declare each file. All of the file attributes values that need to have values other than default values can be modified when the program is initiated from WFL. As flexible as this method is, it does make it harder for a new programmer to understand the basic tasks of the program, so Unisys suggests that you declare the file attributes in the program and modify the values when the program is initiated, if necessary.

Throughout this section, examples for the ALGOL, COBOL74, and COBOL85 languages are given. Refer to your language manual for in-depth information about appropriate syntax. The following discussions identify the universal tasks that you should be concerned with when programming. Information about tasks that must be accomplished for a specific device type or for port files is provided in the appropriate device-type section or in Sections 13 through 29.

The following tasks are described in this section:

- Naming the file
- Specifying the peripheral device for the file
- Specifying the purpose of the file
- Identifying how data is transferred
- Establishing a record format
- Controlling the size field of a variable-length record
- Using byte files in a program
- Using a dummy file
- Opening a file
- Determining if a file exists or is available
- Moving data to and from a file
- Starting at a particular record

- Closing a file
- Modifying an attribute
- Interrogating an attribute
- Determining attribute conflicts
- Dealing with translation

Use the file declaration mechanism of your language to identify the characteristics of your file. Such a declaration associates the name of the file declaration with the INTNAME attribute and defines the logical file to be used by the program. Additionally, a file declaration can be used to assign values to the attributes associated with the file. Example file declarations in ALGOL, COBOL74, and COBOL85 follow:

ALGOL

```
FILE(KIND=DISK,NEWFILE=FALSE,DEPENDENTSPECS=TRUE);

FILE OUTPUT_FILE;
```

COBOL74 and COBOL85

```
FD  IN-FILE.

FD  UPDATE-FILE;
        VALUE OF DEPENDENTSPECS IS TRUE,
                 FILENAME IS "MASTER/UPDATE.".
```

Naming the File

The file declaration is used to identify the name of the physical file to be used. In ALGOL, the FILENAME or TITLE attribute is used to specify the name. Other languages might use the same mnemonic or something quite similar, but each language provides a mechanism to assign the FILENAME and TITLE attributes a value. If you do not assign values to the FILENAME or TITLE attributes, the system uses the value of the INTNAME attribute. By default the INTNAME value is the first 17 characters of the file identifier in the program. As a consequence, all file identifiers in a program should be unique through the first
17 characters.

The FILENAME and TITLE attributes are used to identify files on peripheral devices. When the file is on disk, you can identify the name of the family, as distinguished from the name of the file, by specifying a name to the FAMILYNAME attribute.

The FILENAME and TITLE attributes accept file names specified in the file name syntax of the traditional MCP environment. You can also specify file names using POSIX pathname syntax. Refer to the *File Attributes Reference Manual* for more information on the PATHNAME attribute.

The LFILENAME and LTITLE attributes accept file names that don't conform to the traditional MCP environment naming convention. File name nodes greater than 17 characters are truncated when assigned to the FILENAME and TITLE attributes, but not when assigned to LFILENAME and LTITLE. Refer to the *File Attributes Reference Manual* for more information about these attributes.

Specifying the Peripheral Device for the File

One of the most important characteristics that a file has is the class of peripheral device or devices associated with the logical file. The KIND attribute is used to identify the type of device and has the mnemonic values listed in Table 2–1.

Table 2–1. Mnemonic Values for the KIND Attribute

Mnemonic Value	Description
CD	A CD-ROM optical disk is requested.
DISK	A magnetic disk file is requested.
HC	A Host Control file that can link several systems together for data transfers is requested.
HY	A HYPERchannel file that can link several systems together for data transfers on a HYPERchannel network is requested.
ODT	An operator display terminal (ODT) file that allows information to be sent to an ODT or received from an ODT is requested.
PORT	A port file that is capable of interprocess communication is requested.
PRINTER	A printer file is requested. ***Note:*** *The contents of this file are normally spooled by the operating system rather than opening the device directly.*
READER	A card reader file is requested. ***Note:*** *The contents of this file are normally spooled by the operating system rather than opening the device directly.*
REMOTE	A remote file that allows the program to communicate with a remote device.
TAPE	A tape file is requested.
VIRTUAL	An abstract file supported by an IOHANDLER library. ***Note:*** *The contents and attributes of this file are defined and manipulated by a library known as an IOHANDLER, which is outside the operating system. With respect to language constructs, program access is the same as for other devices.*

Specifying the Purpose of the File

If you want to create a file, set the NEWFILE attribute to TRUE. It is not necessary to set NEWFILE to TRUE for printer files, since either one of these files always causes a new file to be created. If you want to ensure that the file is only written to, assign the FILEUSE attribute a value of OUT.

If you want to access an existing file, set the NEWFILE attribute to FALSE. If you want to access that file by using the record format information that is stored with the permanent file, such as record size, set the DEPENDENTSPECS attribute to TRUE. If the logical file you are defining is capable of having information either written to or read from the file, and you want to restrict the use of the logical file to one or the other, assign the FILEUSE attribute a value of OUT or IN.

Identifying How Data Is Transferred

You control how data is transferred to and from the user data area by assigning the FRAMESIZE attribute one of the following values:

Value	Number of Bits Transferred
4	Data is transferred in units of 4 bits or as hexadecimal characters. The INTMODE value must be HEX. This value is not supported by Host Services logical I/O or FTAM.
8	Data is transferred in units of 8 bits. The INTMODE value must not be HEX, BCL, or SINGLE because these INTMODE values do not have 8-bit characters.
48	Data is transferred in units of 48 bits or as full words. This value is compatible with all values of INTMODE.

The following are the default values assigned to the INTMODE attribute by the language compilers:

<table>
<tr><th>Language</th><th colspan="2">INTMODE Default Value</th></tr>
<tr><td>ALGOL</td><td colspan="2">EBCDIC</td></tr>
<tr><td>COBOL74 and COBOL85</td><td colspan="2">EBCDIC if the first 01-level entry of the file is USAGE DISPLAY. HEX if the first 01-level entry of the file is USAGE COMP</td></tr>
<tr><td rowspan="4">Pascal</td><td colspan="2">Depends on the component type:</td></tr>
<tr><td>Component Type</td><td>INTMODE Default Value</td></tr>
<tr><td>Packed array with 4-bit elements</td><td>HEX</td></tr>
<tr><td>Packed array of characters</td><td>EBCDIC or ASCII depending on the setting of the STRINGS compiler control option</td></tr>
<tr><td></td><td>All other component types</td><td>SINGLE</td></tr>
</table>

If you are programming for FTAM, refer to Section 9, "Accessing and Creating Files Using Distributed File Services," for information about assigning INTMODE values for an FTAM file.

Establishing a Record Format

The following information does not apply to port files or virtual files. Refer to Sections 13 through 29 for information about port file data transfer and to Section 30, "Understanding Virtual Files," for information about virtual file data handling.

In the MCP environment, you can create a file with the following record formats:

- Fixed-length unblocked records
- Fixed-length blocked records (for tape and disk)
- Fixed-length records, no blocks (for disk)
- Variable-length unblocked records
- Variable-length blocked records (for tape and disk)
- Variable-length records, no blocks (for disk)
- Byte streams (for tape and disk)

The BLOCKSTRUCTURE, MINRECSIZE, MAXRECSIZE, and BLOCKSIZE attributes are used to define the record format for a file.

Byte streams are a fixed length, blocked with MAXRECSIZE = 1, and ANYSIZEIO = TRUE.

Indicating the Record Size

Use the MINRECSIZE and MAXRECSIZE attributes to indicate the possible range of the record sizes. If you are defining a fixed length record, do not give a value to the MINRECSIZE attribute.

Table 2–2 shows the default value and maximum value for each type of device for the MAXRECSIZE attribute.

Table 2–2. MAXRECSIZE Default and Maximum Values

Device Type	Default Value	Maximum Value
DISK	30 words	65535 FRAMESIZE units
ODT	10 words	65535 FRAMESIZE units
REMOTE	12 words	1528 words
PRINTER	22 words	
READER	14 words	Same as default
TAPE	10 words	

Indicating the Size of the Blocks and Buffers

Grouping several physically adjacent records into one block reduces the I/O operation time when reading and writing records, because a block of records is brought into memory and then each record is made available to the program one at a time without requiring a physical I/O operation. Because blocked disk files are handled differently from other files, refer to Section 3, "Using Disk and CD-ROM Files in a Program," for information about handling blocks and buffering. To specify that you want to group your records into a block, assign the BLOCKSIZE attribute a value that reflects a length that can accommodate more than one record. Be aware that if the records are of variable length, you can waste space if you do not give BLOCKSIZE a value larger than the MAXRECSIZE value. When determining the block size, keep in mind that if you use large blocks, the I/O operations are efficient, but your program is tying up a large amount of main memory. On the other hand, if you use very small blocks, the I/O subsystem must perform more I/O operations.

The BLOCKSIZE attribute value should be specified in terms of the FRAMESIZE attribute value. If you are going to use Host Services logical I/O with this file, the BLOCKSIZE attribute must be less than 65486 characters.

Indicating the Type of Variable-Length Record

In the MCP environment, there are five types of variable-length records. Each type indicates the length of the record in a different way, and your program must have code that supports the indicated type of variable-length record. You indicate the type of variable-length record by using the BLOCKSTRUCTURE attribute.

Table 2–3 identifies the BLOCKSTRUCTURE mnemonics for the various types of variable-length records, where or how the size information is stored or indicated, and which distributed systems services (DSSs) support the type.

Table 2–3. Possible BLOCKSTRUCTURE Values for Variable-Length Records

Mnemonic Value	Length Information	Distributed Systems Services (DSSs)
EXTERNAL	Neither the record itself nor the structure of the file contains information about the length of the record. You must specify length information externally in the I/O statement, unless you are using unblocked files. Refer to "Understanding Record Length When BLOCKSTRUCTURE Equals EXTERNAL" later in this section for information about using unblocked files.	Host Services logical I/O for unblocked files and FTAM
LINKED	FORTRAN linked records. The link words are maintained by the I/O subsystem and are not part of the data. The INTMODE value of the file is assumed to be SINGLE, and software translation is never attempted.	None
VARIABLE	The record size is contained in the first four characters or first word of the record. The I/O subsystem can maintain the size field.	FTAM
VARIABLE2	The record size is contained in the first two characters or first word of the record.	None
VARIABLEOFFSET	The record size is contained in a fixed location in the record. You specify the location.	None

Understanding Record Length When BLOCKSTRUCTURE Equals EXTERNAL

If you select the EXTERNAL value for BLOCKSTRUCTURE, consider the following information as you program:

- If your program reads unblocked tape files, port files, remote files, or ODT files, the I/O subsystem determines the actual length of the record and returns the length information through the CURRENTRECORDLENGTH and STATE attributes.
- If your program reads a blocked file and specifies a record length that is longer than the remainder of the block, the record is truncated at the end of the block.

- If your program writes to a blocked file and specifies a record length that is longer than the remainder of the block, a new block is started. If the file is a tape file, the previous block is written as a short block.

Controlling the Size Field When BLOCKSTRUCTURE Equals VARIABLE

If you select the VARIABLE value for BLOCKSTRUCTURE, either your program or the system can be responsible for maintaining the size field of the record. If you want the system to be responsible for determining the size of the records, set the SIZEVISIBLE attribute to FALSE. When SIZEVISIBLE is FALSE, the record size field is not visible to the program. Files can be created with the SIZEVISIBLE attribute set to TRUE, and then they can be read or updated with the SIZEVISIBLE attribute set to FALSE, and vice versa.

When your program is responsible for maintaining the size field of the record, the first four characters or the first word of each record contains a decimal or binary number that indicates the number of FRAMESIZE characters in the record, including the size field itself. The INTMODE attribute value controls the size field characteristics and the maximum and minimum record size that can be used. Table 2–4 shows the size field characteristics and maximum and minimum values.

Table 2–4. Size Field Information Based on INTMODE Value

INTMODE Value	Place in Record	Numeric Representation	Maximum Value	Minimum Value
SINGLE	First word	Binary	MAXRECSIZE value	MINRECSIZE value
OCTETSTRING	First four octets	A decimal number represented in EBCDIC characters	9999	4
All others	First four characters	A decimal number represented in INTMODE characters	9999	4

You must increase the values of MAXRECSIZE, MINRECSIZE, and BLOCKSIZE to accommodate the size fields when you declare the values for these attributes.

If you set the SIZEVISIBLE attribute to FALSE, the I/O subsystem maintains the size field. The following information is important to know:

- When a WRITE operation is invoked, the size of the record is set to the length of the data written.

- When a READ operation is invoked, the size of the record is returned in the logical result descriptor and the CURRENTRECORDLENGTH attribute. Refer to the STATE attribute in the *File Attributes Reference Manual* for the location of the information in the logical result descriptor.
- The MAXRECSIZE attribute cannot be larger than 9995.
- The MAXRECSIZE, MINRECSIZE, and BLOCKSIZE you specified are adjusted upward by the operating system to accommodate the record size field.
- If you interrogate the MINRECSIZE and MAXRECSIZE attributes, the original values that your program specified are returned rather than the adjusted values.
- If you interrogate the BLOCKSIZE attribute, the adjusted block size is returned.

If you select the VARIABLE2 value for BLOCKSTRUCTURE, your program is responsible for maintaining the size field of the record. Represent the size as a binary number. If the INTMODE value is not SINGLE, the size information is in the first two characters of the record. If the INTMODE value is SINGLE, the size information is in the first word of the record.

Your program must define a record size that accommodates the size field.

Controlling the Size Field When BLOCKSTRUCTURE Equals VARIABLEOFFSET

If you select the VARIABLEOFFSET value for BLOCKSTRUCTURE, your program must perform the following tasks:

- Specify how the size field information is recorded by assigning a value to the SIZEMODE attribute. If the file you are defining is a disk or tape file, the value can be different from the INTMODE value. For all other devices, the value cannot be different from the INTMODE value.

  The following mnemonic values are available for the SIZEMODE attribute:

Mnemonic Value	Representation of Record Size
ASCII	8-bit decimal digits
EBCDIC	8-bit decimal digits
HEX	4-bit, packed decimal digits
SINGLE	Binary

- Specify where the size field is stored in the record in SIZEMODE units by using the SIZEOFFSET attribute. If you want the size field to start in the first position of the record, you do not need to assign a value to the SIZEOFFSET attribute, because the default is 0 (zero). For any other position, indicate how far into the record you want the size field to be.
- Specify how long, in SIZEMODE units, the size field is by assigning a value to the SIZE2 attribute. If you assign the SIZEMODE attribute a value of SINGLE, the SIZE2 value defaults to a value of 1. The size of the field has to be within the range defined by the MINRECSIZE and MAXRECSIZE values.

Writing on a File with Variable Length Records

A program performing an update write on a file with variable length records that is either blocked or has FILESTRUCTURE=STREAM will get a DATAERROR (DIFFERENTLENGTHRECORDS error for COBOL85 programs) if the record being written is not the same size as the record being replaced. The data in the file is not modified and the current record pointer is moved to the next record. Therefore, the next serial read or write operation is performed on the next record in the file. This behavior applies also to COBOL85 rewrite operations to unblocked files.

Using Byte Files in a Program

The I/O subsystem supports byte-oriented files, which are often referred to as stream files. These files can be generated by a number of softwares, including FTP, NetWare, and NX/Services on behalf of workstation users, and are commonly produced by a C program using POSIX interfaces.

Some attributes vary based on the exact requirements of the file but in general the following attribute values are required for a permanent disk byte file:

- BLOCKSTRUCTURE is FIXED.
- EXTMODE is a value, which reflects 8-bit characters (SINGLE, HEX and BCL are not allowed).
- FILESTRUCTURE is STREAM.
- FILEORGANIZATION is NOTRESTRICTED.
- MAXRECSIZE is 1.
- FRAMESIZE is 8.

It is possible for an ALGOL program to produce a byte file by setting file attributes to the appropriate values. The following attribute settings are required to open a file. Note that the value required for some of the attributes is the default value and need not be specified in the file declaration.

- ANYSIZEIO is TRUE.
- BLOCKSTRUCTURE is FIXED.
- FILEORGANIZATION is NOTRESTRICTED (default).
- FILESTRUCTURE is STREAM.
- FRAMESIZE is 8, or UNITS is not WORDS.
- INTMODE and EXTMODE are values, which reflect 8 bit characters (SINGLE, HEX and BCL are not allowed).
- MAXRECSIZE is 1.
- UPDATEFILE is FALSE (default).

Byte files differ in several ways from traditional files in the MCP environment (which are record files). Record-files are traditional disk files. Record-files have MAXRECSIZE and FRAMESIZE attributes that define their record layout. Examples of record files include ALGOL source files, SEQDATA and DATA data files, and code files.

Byte files are randomly accessed sequences of bytes with no further structure imposed upon them by the operating system. Byte-files are like the traditional file kinds on DOS and UNIX machines. Byte-files do not have any record layout specified by file attributes. They are sequences of bytes. The setting of MAXRECSIZE=1 and ANYSIZEIO=TRUE enables I/O operations to be of any length.

If you are not familiar with basic programming methods, review those methods in this section. You can identify all the file attributes that can be used with a byte file by reviewing the attributes shown for disk files in Table A–1. You can also find more information about any of the mentioned attributes in the *File Attributes Reference Manual*.

Types of Byte Files

There are three distinct types of byte files. They can exist as "normal" files, stored on disk along with a header and associated data areas. In addition, byte files are used for permanent directories and to provide special kinds of files based on the POSIX specification, namely FIFO files and files with the semantics of the /dev/null file. These files have no associated data storage areas on disk. They function only as a disk header that describes the attributes of the file. These files are distinguished from other byte files by their FILEKIND values. The FILEKIND values of 232 through 240 are reserved for POSIX special files, with 238 = DEVNULL and 240 = FIFO. The FILEKIND value of 190 is reserved for permanent directories. The other values are not currently used.

To treat a permanent disk file as one of the special files, it must include the following attributes in addition to the required attributes listed earlier:

- FILEKIND is FIFO or DEVNULL or PERMDIR.
- LASTRECORD is –1.

The following attribute settings must be used for the logical file:

- ALLOWSPECIALFILE is TRUE. (for FIFO and DEVNULL)
- The declaration does not specify a DIRECT file.

Permanent Directory Files

Permanent directories are files having security attributes that govern that can access sub-directories and files at the next directory level. Permanent directories have no data records, only a disk file header. Permanent directories can exist only in the permanent directory namespace consisting of files whose name starts with the node *DIR.

Permanent directories are created using the WFL MKDIR statement, logical I/O when FILEKIND is set to PERMDIR, and by Client Access Services on behalf of a workstation user doing a make directory operation in an MCP share. Read and write operations to permanent directories are not allowed.

FIFO Files

FIFOs are files that have no data storage on disk. While a FIFO is open it will have a data storage area in memory, which is used to implement a "first in first out" queue. Any number of logical files may simultaneously have a given FIFO open. Each logical file may have the FIFO open as read-only, write-only, or read/write.

Creating a FIFO

Most practical uses of a FIFO require that the FIFO be made permanent in the directory before it can be used. This allows multiple independent processes to access the FIFO. If you choose to do this, open a file with the required special byte file attribute values. In addition, you must

- Set FILEKIND to FIFO.
- Set NEWFILE to TRUE.
- Set FILEUSE to IO.
- Create the FIFO as a permanent file, either by setting PROTECTION to SAVE, or by closing the FIFO with LOCK.

The FIFO can also be created through use of the POSIX interface MKFIFO. See the *MCP/ AS ALGOL and MCP Interfaces to POSIX Features Programming Reference Manual* for more information on creating FIFOs.

Opening a FIFO

Perform the following tasks before you open the file:

- Specify the required special byte file attributes listed earlier in this section.
- Set the value of FILEKIND to FIFO.
- Specify whether you are going to use the FIFO for input only, output only, or both input and output by setting the FILEUSE attribute to IN, OUT, or IO.
- Set the value of NONBLOCK to provide the desired I/O action. (See the following information.)

The action taken when a program opens a FIFO depends upon the setting of the NONBLOCK attribute.

- If NONBLOCK is FALSE, an open for read-only will mark the physical file as open for reading, but the open will not finish (i.e. return control to the invoker) until the physical file is also open for writing. Likewise, if NONBLOCK is FALSE, an open for write-only will mark the physical file as open for writing, but the open will not finish (i.e. return control to the invoker) until the physical file is also open for reading.
- If NONBLOCK is TRUE, an open for read-only will mark the physical file as open for reading, and the open will finish (for example, a return control to the invoker) even if the physical file is not open for writing. If NONBLOCK is TRUE, an open for write-only will mark the physical file as open for writing if and only if the physical file is also open for reading. Otherwise, a NOFIFOREADERRSLT open error will result.

Perform the following tasks before you open the file:

- Set the value of FILEKIND to FIFO.
- Specify whether you are going to use the FIFO for input only, output only or both input and output by setting the FILEUSE attribute to IN, OUT, or IO.

FIFOs provide a mechanism to queue arbitrary sequences of bytes of data. Only serial reads and serial writes are permitted for FIFOs. Bytes of data are presented to read statements in the same order in which they arrived via write statements. Data written to a FIFO by any given write statement is never interleaved with data written by another write statement.

The semantics of read and write operations depend on the setting of the NONBLOCK file attribute and the size of the operation. POSIX defines a value called PIPE_BUF, which in the MCP environment is currently a value of 6144 (this value is available programmatically as _PC_PIPE_BUF, using the POSIX_PATHCONF function).

If NONBLOCK is FALSE:

- A write operation will not finish until all data has been written. If the number of bytes to be written is less than or equal to PIPE_BUF, then the data will be placed into the FIFO in a single operation. If the number of bytes to be written is greater than PIPE_BUF, then the data will be placed into the FIFO in multiple pieces, which are PIPE_BUF in length.
- A read operation will complete as soon as data is available. No more than the requested amount of data is read. If less than the requested amount of data is read, a short block logical result will be returned. Note that no more data than PIPE_BUF bytes is ever available for a single read.
- If several programs are waiting to write to a FIFO, they will be serviced in the order in which they were waiting. Likewise, if several programs are waiting to read from a FIFO, they will be serviced in the order in which they were waiting. The exception is that a program that is stopped by an operator "ST" or by a POSIX stop signal loses its place in line.
- If all writers and readers of a given FIFO specify the same data length, and that length is less than or equal to PIPE_BUF, then the reads will return data in the same chunks in which it was written.
- Read and write operations are interruptible by POSIX signals while they are waiting to transfer data (see the *POSIX User's Guide* for a description of POSIX signals). In that case the appropriate signal handler is invoked and the read or write operation might return an exception. If no data has yet been transferred a CANCELED will be returned; if data has been transferred then a short block logical result will be returned. This action is dependent on the specifics of the defined action for the signal; there are cases where signals will be ignored or where the task will resume waiting to transfer data after handling the signal.

If NONBLOCK is TRUE:

- A write operation with a requested length less than or equal to PIPE_BUF finishes immediately if there is room in the FIFO for all of the data. Otherwise, the write operation will return a NOBUFFERFORWRITE error.
- A write operation whose requested data length is greater than PIPE_BUF finishes immediately if there is room in the FIFO for any data at all. As much data as fits will be written; if all data does not fit then a short block logical result will be returned.
- A read operation finishes immediately if any data is available. No more than the requested amount of data is read. If less than the requested amount of data is read then a short block logical result is returned. If no data is available, then the read will return a NODATAFORREAD error.

A write operation will return an error (NOREADERS) if there are no readers. In addition, if the invoking process is signal capable, a SIGPIPE signal will be sent to it. A read operation will return an error (ENDOFFILE) if there are no writers.

The /dev/null File

The /dev/null file is a special kind of file intended primarily for use by POSIX programs. This file has no storage on disk. It is created automatically when the DL ROOT ODT command is entered.

The /dev/null file is used for file redirection. The POSIX semantics apply to the /dev/null file. The data from any write to /dev/null is discarded, and any read from /dev/null will return an ENDOFFILE error (no data is returned). The /dev/null file provides semantics very similar to the logical file attribute DUMMYFILE.

Creating a File with /dev/null Semantics

A user can create a file with the same semantics as /dev/null. In addition to the required special byte file attributes listed earlier, the FILEKIND must be set to DEVNULL.

COBOL85 Sample

The following COBOL85 example reads a DISK file of fixed length records and then writes it as a byte stream file. When writing the file, each record is terminated with a carriage-return and linefeed (CR–LF). Next the program reads the resulting file, displaying its records at the REMOTE.

```
$$ RESET NOLIMITS
 IDENTIFICATION DIVISION.
 PROGRAM-ID. STREAM-EXAMPLE.
 ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
   SELECT IN-FILE  ASSIGN TO DISK.
   SELECT OUT-FILE  ASSIGN TO DISK.
   SELECT VF       ASSIGN TO DISK.
   SELECT REM      ASSIGN TO REMOTE.
*
 DATA DIVISION.
 FILE SECTION.
 FD IN-FILE
      VALUE OF FILENAME      IS "SOME/DISK/FILE"
               DEPENDENTSPECS IS TRUE.
 01 IN-REC.
    03 IN-DATA PIC X OCCURS 1 TO 3000 DEPENDING ON REC-SIZE.


 FD OUT-FILE
      RECORD CONTAINS 1 TO 3000 CHARACTERS
      VALUE OF FILENAME      IS "SOME/STREAM/FILE"
               FILESTRUCTURE IS STREAM
               FILEORGANIZATION IS NOTRESTRICTED
               BLOCKSTRUCTURE IS FIXED
               FRAMESIZE     IS 8
               MAXRECSIZE    IS 1
               MINRECSIZE    IS 1
               ANYSIZEIO     IS TRUE
               NEWFILE       IS TRUE
               EXTMODE       IS ASCII.
 01 OUT-REC.
    03 OUT-REC-DATA  PIC  X OCCURS 1 TO 3000
                          DEPENDING ON REC-SIZE-OUT.
 01 OUT-REC-X.
    03 OUT-REC-DATA-X PIC X OCCURS 1 TO 3000
                          DEPENDING ON CHAR-CNT.


 FD VF
      VALUE OF FILENAME      IS "SOME/STREAM/FILE"
               DEPENDENTSPECS IS TRUE
               ANYSIZEIO     IS TRUE.
```

```
 01 VF-REC     PIC X(3000).


 FD REM.
 01 REM-REC    PIC X(80).


 WORKING-STORAGE SECTION.
 77 OUT-LEN                REAL.
 77 FRM-SIZE               REAL.
 77 REC-SIZE               REAL.
 77 REC-SIZE-OUT           REAL.
 77 REC-CNT                REAL.
 77 CHAR-CNT               PIC 999 COMP.
 77 EOF-INFILE             PIC X         VALUE "F".
 77 DISPLAY-FLAG           PIC X         VALUE "T".
 77 UNSTRING-CNT-OVERLAP   REAL.
 77 NEXT-UNSTRING          REAL.
 77 UNSTRING-START         REAL.
 77 UNSTRING-CNT           REAL.
 77 MOVED-CNT              REAL.
 77 MSG-LEN                REAL.
 77 DISPLAY-SIZE           REAL          VALUE 80.
 01 DUMMY-REC.
    03 DUMMY-REC-R PIC X OCCURS 3000.
 01 REC-ARRAY    PIC X(3000).
 01 REM-ARRAY.
    03 REM-ARRAY-R PIC X OCCURS 80.
*
 PROCEDURE DIVISION.
  MAIN-PARA.
    OPEN INPUT IN-FILE.
    MOVE ATTRIBUTE MAXRECSIZE OF IN-FILE TO REC-SIZE.
    MOVE ATTRIBUTE FRAMESIZE  OF IN-FILE TO FRM-SIZE.
    IF FRM-SIZE = 48 THEN
       MULTIPLY REC-SIZE BY 6 GIVING REC-SIZE.


    PERFORM CREATE-STREAM-FILE THRU CREATE-STREAM-FILE-EXIT.
*   We now have a STREAM file version of the DISK file to read
*   and display.


    MOVE 0 TO REC-CNT.
    OPEN INPUT VF.
    OPEN OUTPUT REM.
*   Now read and display the STREAM file.
    PERFORM READ-AND-DISPLAY-VF THRU READ-AND-DISPLAY-VF-EXIT.
```

```
      CLOSE VF.
      CLOSE REM.


    MAIN-PARA-EXIT.
      STOP RUN.


    CREATE-STREAM-FILE.
      OPEN OUTPUT OUT-FILE.
      COMPUTE REC-SIZE-OUT = REC-SIZE + 2.


    CREATE-STREAM-LOOP.
      PERFORM GET-NEXT-RECORD THRU GET-NEXT-RECORD-EXIT.
      IF EOF-INFILE = "F" THEN
         PERFORM FORMAT-FOR-STREAM THRU FORMAT-FOR-STREAM-EXIT
         PERFORM WRITE-STREAM-RECORD THRU WRITE-STREAM-RECORD-EXIT
         GO TO CREATE-STREAM-LOOP.


    CREATE-STREAM-FILE-EXIT.
      CLOSE OUT-FILE SAVE.


    GET-NEXT-RECORD.
      READ IN-FILE AT END MOVE "T" TO EOF-INFILE
                   NOT AT END
                          MOVE IN-REC TO DUMMY-REC.


    GET-NEXT-RECORD-EXIT.
      EXIT.


    FORMAT-FOR-STREAM.
      MOVE REC-SIZE-OUT TO CHAR-CNT
    *   Trim any trailing blanks and mark the end with CR-LF.
      PERFORM UNTIL (CHAR-CNT = 0) OR
                    (DUMMY-REC-R(CHAR-CNT) NOT = " ")
         SUBTRACT 1 FROM CHAR-CNT
      END-PERFORM
      MOVE @0D@ TO DUMMY-REC-R(CHAR-CNT + 1)
      MOVE @25@ TO DUMMY-REC-R(CHAR-CNT + 2)
      ADD 2 TO CHAR-CNT.


    FORMAT-FOR-STREAM-EXIT.
      EXIT.
```

```
  WRITE-STREAM-RECORD.
*   Now write the DISK file as a STREAM file.
    MOVE DUMMY-REC TO OUT-REC
    WRITE OUT-REC-X
    ADD 1 TO REC-CNT.


  WRITE-STREAM-RECORD-EXIT.
    EXIT.


  READ-AND-DISPLAY-VF.
    PERFORM GET-NEXT-STREAM-REC THRU GET-NEXT-STREAM-REC-EXIT.
    PERFORM REMOTE-DISPLAY THRU REMOTE-DISPLAY-EXIT.
    IF DISPLAY-FLAG = "T"
       GO TO READ-AND-DISPLAY-VF.


  READ-AND-DISPLAY-VF-EXIT.
    EXIT.


  GET-NEXT-STREAM-REC.
    READ VF
         AT END
            MOVE "F" TO DISPLAY-FLAG
            IF UNSTRING-CNT-OVERLAP > 0 THEN
*           Write any residue from previous overlap at end.
               WRITE REM-REC FROM REM-ARRAY
               MOVE 0 TO UNSTRING-CNT-OVERLAP
            END-IF
         NOT AT END
            MOVE ATTRIBUTE CURRENTRECORDLENGTH OF VF TO MSG-LEN.


  GET-NEXT-STREAM-REC-EXIT.
    EXIT.


  REMOTE-DISPLAY.
    IF DISPLAY-FLAG = "T" THEN
       MOVE VF-REC TO REC-ARRAY
       MOVE 1 TO NEXT-UNSTRING
       PERFORM WRITE-RECS-TO-REMOTE THRU
               WRITE-RECS-TO-REMOTE-EXIT UNTIL MSG-LEN <= 0.
$$ SET OMIT = NOLIMITS
*   Limit the volume of displayed output.
    IF REC-CNT > 9 MOVE "F" TO DISPLAY-FLAG.
$$ POP OMIT
```

```
  REMOTE-DISPLAY-EXIT.
    EXIT.


  WRITE-RECS-TO-REMOTE.
    MOVE 0 TO UNSTRING-CNT.
    MOVE 0 TO MOVED-CNT.
    MOVE NEXT-UNSTRING TO UNSTRING-START.
    IF UNSTRING-CNT-OVERLAP > 0 THEN
*      The last time thru here we didn't write all the data,
*      because it was an incomplete record.  Add new data to what
*      was left over for the next record we write.
       UNSTRING REC-ARRAY
                DELIMITED BY @0D25@
                INTO REM-ARRAY-R(UNSTRING-CNT-OVERLAP + 1)
                COUNT IN MOVED-CNT
                WITH POINTER NEXT-UNSTRING
       COMPUTE MOVED-CNT = MOVED-CNT + UNSTRING-CNT-OVERLAP
       MOVE 0 TO UNSTRING-CNT-OVERLAP
    ELSE
       MOVE SPACES TO REM-ARRAY
       UNSTRING REC-ARRAY
                DELIMITED BY @0D25@
                INTO REM-ARRAY
                COUNT IN MOVED-CNT
                WITH POINTER NEXT-UNSTRING.
*   MOVED-CNT    characters have been moved to REM-ARRAY.
*   NEXT-UNSTRING is next position in source after UNSTRING,
*   following the delimiter(s).


    SUBTRACT UNSTRING-START FROM NEXT-UNSTRING
             GIVING UNSTRING-CNT.
*   Decrement input message length by what we just consumed.
    SUBTRACT UNSTRING-CNT FROM MSG-LEN.


    IF MSG-LEN = 0 AND
          MOVED-CNT < DISPLAY-SIZE THEN
*      This display record needs more data from the next READ to
*      make it complete.
       MOVE MOVED-CNT TO UNSTRING-CNT-OVERLAP
    ELSE
       WRITE REM-REC FROM REM-ARRAY
       ADD 1 TO REC-CNT.


  WRITE-RECS-TO-REMOTE-EXIT.
    EXIT.


 END PROGRAM STREAM-EXAMPLE.
```

Using a Dummy File

In certain situations you might want to test the logic of a program, but you do not want to actually write to or read from a file. To identify such a file in your program, set the DUMMYFILE attribute to TRUE. As a result, the logical file is not assigned to a permanent file, no buffers are allocated, no I/O operations are performed, and all program READ operations return an end-of-file indicator. You should not set DUMMYFILE to TRUE if your program rereads records that have been written to the file in the current program, because the records will not be there.

An alternative to using the DUMMYFILE attribute is available if you are using the POSIX implementation. Disk files such as /dev/null, created by the DL ROOT system ODT command with the FILEKIND attribute set to DEVNULL, have very similar properties to logical files that specify a file with the DUMMYFILE value of TRUE.

Opening a File

Before a program can create data or access data contained in a physical file, an association between the physical file and the program must be made. You make this association by assigning a physical file to a logical file. An explicit OPEN statement does not cause data to be transferred between logical and physical files, and the logical file can be closed without any I/O operations being performed on the file.

Use the OPEN statement to accomplish the assignment task and to mark the logical file as open. I/O statements such as READ and WRITE require the logical file to be open before they can perform their functions. In some languages other than COBOL, a file can be opened implicitly by the first I/O operation to the file. After the OPEN operation is completed, the I/O statement is performed. In most cases, a physical file is assigned to the logical file when the OPEN statement is performed.

You can use certain forms of the OPEN statement to conditionally open a file, so that a logical file is opened only if a physical file satisfying specified matching criteria is found. In addition, you can specify the initial positioning of the logical file.

In some languages, you can also request an OPEN statement to return a result indicating that the OPEN operation was successful, or if the operation was unsuccessful, the reason for the failure. For a listing of the possible OPEN results returned by the system, refer to the *File Attributes Reference Manual*. If you do not request the result, the system takes default actions for each result that would have been returned.

The following are examples of an OPEN statement in ALGOL, COBOL74, and COBOL85:

ALGOL

```
OPEN(F);

I := OPEN(SOURCE_FILE,AVAILABLE);
```

COBOL74 and COBOL85

```
OPEN INPUT IN-FILE.

OPEN EXTEND UPDATE-FILE.
```

The rules of ALGOL require that you specify explicitly any necessary attribute values before opening a file. In some other languages, compilers can generate object code that assigns file attribute values without the programmer explicitly specifying them, in order to accommodate the semantics of implicit or explicit OPEN statements for the file. This is called an "implicit assignment." The attributes commonly assigned implicitly are as follows:

- MYUSE
- FILEUSE
- UPDATEFILE
- DIRECTION
- EXCLUSIVE
- NEWFILE

In addition, when using FORTRAN77, the following attributes may be assigned implicitly:

- PROTECTION
- DEPENDENTSPECS

In languages that use implicit assignments for these attributes, the compiler generates code based on the expectation that you will not change the attributes it has set. Explicitly assigning values to these attributes in these languages can interfere in unpredictable ways with the proper execution of the program.

If a file is opened implicitly, or the value returned by the OPEN function is discarded and an open error occurs, the program is terminated with the following error message:

```
FILE <title> OPEN ERROR: <error message>
```

All open errors are fatal unless the program has an error-handling routine to handle the result of the OPEN operation. The open error message numbers that appear in the error message or in the system log are different from the OPEN result numbers that you would have received if your program interrogated the OPEN result.

Determining the Existence or Availability of a File

In certain instances you might want to determine whether a file exists or is available without causing your program to wait if the file is not present on the system. You can use the following attributes for this task:

- Interrogate the AVAILABLE attribute to determine if a file is present or can be opened. If the file cannot be opened, a reason is returned, the program is not suspended, and an operator does not have to answer a "NO FILE" waiting message.

  The AVAILABLE attribute can be used as an option of the OPEN statement in some languages.

- Interrogate the PRESENT attribute to determine if a file is open, and if the file is not open, to try to open the file. If a disk file is open and is being used exclusively by another program, the logical I/O subsystem waits until the disk file is closed and then opens the file.

- Interrogate the RESIDENT attribute to determine if a permanent file exists or if a physical unit is available. TRUE is returned if a permanent file exists or a physical unit is available. Interrogating this attribute does not open the file.

Moving Data to and from a File

You are now ready to read data from your file, write data to your file, or both. Use the READ and WRITE statements of your language to accomplish this task.

There are two major types of READ operations: serial and random. Use a serial READ operation to obtain the data from the next record in the file. Use a random READ operation to obtain the data from a specific record.

In some languages, you can also request in a READ statement that a result be returned that indicates if the READ operation was successful, or if the operation was unsuccessful, the reason for the failure. Refer to the *File Attributes Reference Manual* for a listing of the possible I/O result enumerated values. If you do not request a result, the system takes default actions for each result that would have been returned.

All READ errors are fatal unless the program has an error-handling routine to handle the result of the READ operation. Your program is terminated if a fatal READ error is not handled, and the following message is displayed:

```
FILE <title>  I/O   ERROR: <error message>
```

The error message number in this message and the system log message number are different from the result enumerated values that your program would have received if the program had requested the result.

The following are examples of a serial READ statement and a random READ statement in ALGOL, COBOL74, and COBOL85:

ALGOL

```
READ(F,90,A);

B := READ(IN_FILE[REC_NUM],IN_RECORD_SIZE,
INPUT_POINTER);
```

COBOL74 and COBOL85

```
READ IN-FILE.

MOVE NEXT-FILE-INDEX TO RANDOM-KEY.
READ RANDOM-FILE;
    INVALID KEY GO TO DISPLAY-KEY-ERROR.
```

When a READ operation is initiated, data is transferred to your program from the buffer in memory. If no more records are found in the buffer, or, in the case of a random read, if the desired record is not in the buffer, the operating system does a physical READ operation to the I/O device.

There are two major types of WRITE operations: serial and random. Use a serial WRITE operation to place the data in the next record in the file. Use a random WRITE operation to place the data in a specific record in the file.

In some languages, the WRITE statement can request that a result be returned that indicates if the WRITE operation was successful, or if the operation was unsuccessful, the reason for the failure. Refer to the *File Attributes Reference Manual* for a listing of the possible I/O results. If you do not request a result, the system takes default actions for each result that would have been returned.

All WRITE errors are fatal unless the program has an error-handling routine to handle the result of the WRITE operation. Your program is terminated if a fatal WRITE error is not handled and the following message is displayed:

```
FILE <title>  I/O   ERROR: <error message>
```

The error message number in this message and the system log message number are different from the result enumerated values that your program would have received if the program had requested the result.

The following are examples of a serial WRITE statement and a random WRITE statement in ALGOL, COBOL74, and COBOL85:

ALGOL

```
WRITE(F,90,A);

B :=WRITE(OUT_FILE[REC_NUM],OUT_RECORD_SIZE,
OUTPUT_POINTER);
```

COBOL74 and COBOL85

```
WRITE OUT-RECORD.

MOVE EMPLOYEE-NUMBER TO UPDATE-KEY.
WRITE UPDATE-DATA;
     INVALID KEY PERFORM HANDLE-INVALID-KEY.
```

When a WRITE operation is initiated, data is transferred from your program to the buffer in memory. If the buffer is full, or in the case of a random WRITE, if the record does not belong in the current buffer, the operating system does a physical WRITE operation to the I/O device.

The I/O subsystem does extensive checking on physical I/O operations and translates error and exception information into a result descriptor that you can interrogate. If the file has more than one buffer, which is the usual case, and a logical I/O statement initiates a physical I/O operation, the physical I/O operation is completed asynchronously with the program. When a physical I/O operation is initiated, the buffers are rotated so that the program can continue to use the next buffer. The error analysis for a physical WRITE operation is performed when the buffer is rotated back to the top position, and therefore is always behind the position of the logical file by the number of buffers the file has. Because physical READ operations are initiated ahead of the logical file position in anticipation of sequential reading, error analysis for physical READ operations matches the logical file position.

The result descriptor is returned for each logical I/O operation and is directly accessible in some of the languages. You can also interrogate the STATE attribute to obtain information about the last I/O operation. If you do not request a result, the system takes default actions for each result that would have been returned.

If the SYNCHRONIZE attribute value is OUT or you use the SYNCHRONIZE option in a WRITE statement, you can determine which I/O operations are causing the error. Synchronous output causes the system to complete the I/O operation before going on; thus, you can determine where the error is occurring. However, using synchronous I/O processing is less efficient than using normal asynchronous buffer processing.

Certain styles of tape drives that have a long repositioning time (for example, the HS8500) might experience a significant impact on performance when using synchronous I/O.

Starting at a Particular Record

If you want to start processing records sequentially starting at a certain record without transferring the information into your program at that time, use the SEEK statement. The next serial I/O operation reads or writes that record. You can request that a result be returned indicating that the SEEK operation was successful, or if the operation was unsuccessful, the reason for the failure. The value is returned in the same format as that returned by the STATE attribute. For a description of the information returned, refer to the STATE attribute in the *File Attributes Reference Manual*. If you do not request that a result be returned, the system takes default actions for each result that would have been returned.

The following examples, in ALGOL, COBOL74, and COBOL85, show how to start at a particular record:

ALGOL

```
SEEK(F[0]);

R :=SEEK(F[SAVED_RECORD_NUMBER]);

SPACE(F,-1);
```

COBOL74 and COBOL85

```
MOVE NEXT-KEY TO UPDATE-KEY.
SEEK UPDATE-FILE.
```

Closing a File

Use the CLOSE statement to mark a logical file as closed. Generally, the physical file is disassociated from the logical file when the CLOSE statement is performed, although an option is provided to allow the association to be retained. You can also change the disposition of the physical file. Some languages require the file to be closed before the program is exited.

You can request that a result be returned that indicates whether the CLOSE operation was successful, or if the operation was unsuccessful, the reason for failure. Refer to the *File Attributes Reference Manual* for a list of CLOSE results.

All close errors are fatal unless the program accepts responsibility for handling the error by referencing the value returned by the CLOSE operation.

If you do not request a result, the system takes default actions for each result that would have been returned. If a file is closed implicitly, or if the value returned by the CLOSE operation is discarded, and a close error occurs, the system terminates the program and displays the following error message:

```
FILE <title>  CLOSE ERROR: <error message>
```

The error message number in this message and the system log message number are different from the CLOSE error message numbers that your program would have received if the program had requested the result.

The following examples show how to use the CLOSE statement in ALGOL, COBOL74, and COBOL85:

ALGOL

```
CLOSE(F);

CLOSE(F,CRUNCH);

CLOSE(F,REWIND);
```

COBOL74 and COBOL85

```
CLOSE UPDATE-FILE.

CLOSE OUT-FILE WITH SAVE.
```

When a CLOSE statement is invoked, the MCP initiates the actions to be taken. Those actions are based on the CLOSE operation requested by the program. The MCP gives an incoming CLOSE operation three different identifiers.

MCP Identifier	Value	Description
Type	Regular	The file is closed.
	Reel	The current reel is closed.
	Dontwait	Control is returned to the program as soon as possible after the port file or subport file has been updated to CLOSED, CLOSEPENDING, or CLOSEREQUESTED.
Association	Retain	Physical file remains assigned to logical file. The MCP changes the association from retain to release if the disposition is rewind for a tape file that has a FILESECTION file attribute value greater than 1 or that is making a reel switch.
	Release	Physical file is no longer assigned to logical file.
	Reserve	The unit can be only assigned to this stack. This association gets special handling for FORTRAN77 file declarations.
	Disable	The logical file cannot be reassigned.
Disposition	Rewind	The file is rewound.
	Lock	The file is locked.
	Save	The MCP changes the disposition to lock for disk files, and rewind for other files. For foreign files, the disposition remains save, so the foreign host can interpret the tape as it wants.
	No Rewind	The file is not rewound.
	Purge	The file is purged.
	Crunch	The file is crunched.
	Downsizearea	If the file consists of one area, the area size is reduced and the unused portion is returned to the system. In other ways this value behaves like Rewind. See additional conditions in the following text.
	Downsizearealock	The file's behavior is the same as downsizearea. The file is also locked.

The following conditions must be met to reduce the area size of a file. If one condition is not true, the downsize action is not performed and the close operation continues.

- The file is a disk file.
- The file consists of the first area only.
- The file has at least one record.
- The file's area size exceeds 60 sectors.
- The file is not crunched.
- The file's unused space exceeds 10 percent of the area size.
- The file is not opened by another logical file. If the file is opened by another logical file, the downsize action is delayed until the last logical file is closed. If the close of any of the logical files specifies crunch, then a crunch is done instead of a down size area.

The area is downsized to the file's current size (last record) plus 10 percent or 60 sectors, whichever is larger. If the file has a FILESTRUCTURE value of ALIGNED180 or BLOCKED, then the area size is increased to be a multiple of the file's block size.

Tables 2–5 through 2–16 identify the type of CLOSE requested by the specified language and the type, association, and disposition the MCP associates with the requested CLOSE operation. The CLOSE operations that are unique to port files are not identified.

Table 2–5. COBOL85 CLOSE Statement Actions

Statement Option	Type	Association	Disposition
REEL FOR REMOVAL	Reel	Release	Rewind
UNIT FOR REMOVAL	Reel	Release	Rewind
REEL	Reel	Retain	Rewind
UNIT	Reel	Retain	Rewind

Note: *These options can be used only with tapes.*

Table 2–6. COBOL85 CLOSE Statement Actions

Statement Option	Type	Association	Disposition
Default	Regular	Retain	Rewind
Default (Multifile‡)	Regular	Reserve	No Rewind
LOCK	Regular	Disable	Lock
SAVE	Regular	Release	Save

Table 2–6. COBOL85 CLOSE Statement Actions

Statement Option	Type	Association	Disposition
NO REWIND (Multifile) ***Note:*** *The MULTIPLE FILE clause is used in I-O Control.*	Regular	Reserve	No Rewind
NO REWIND	Regular	Retain	No Rewind
PURGE	Regular	Release	Purge
RELEASE	Regular	Release	Rewind
CRUNCH	Regular	Release	Crunch
REMOVE	Regular	Release	Save
REMOVE CRUNCH	Regular	Release	Crunch
NO WAIT	Dontwait	Retain	Rewind

Table 2–7. COBOL74 CLOSE Statement Actions

Statement Option	Type	Association	Disposition
REEL FOR REMOVAL	Reel	Release	Rewind
REEL	Reel	Retain	Rewind
REEL (Multifile) ***Note:*** *The MULTIPLE FILE clause is used in I-O Control.*	Reel	Reserve	Rewind
SAVE FOR REMOVAL	Regular	Release	Lock
REEL SAVE FOR REMOVAL	Reel	Release	Lock

Note: *These options can be used only with tapes.*

Table 2–8. COBOL74 CLOSE Statement Actions

Statement Option	Type	Association	Disposition
Default	Regular	Retain	Rewind
Default (Multifile) ***Note:*** *The MULTIPLE FILE clause is used in I-O Control.*	Regular	Reserve	No Rewind

Table 2–8. COBOL74 CLOSE Statement Actions

Statement Option	Type	Association	Disposition
REEL (Multifile) ***Note:*** *The MULTIPLE FILE clause is used in I-O Control.*	Reel	Reserve	Rewind
REEL LOCK	Reel	Disable	Lock
LOCK	Regular	Disable	Lock
SAVE	Regular	Release	Save
NO REWIND (Multifile) ***Note:*** *The MULTIPLE FILE clause is used in I-O Control.*	Regular	Reserve	No Rewind
NO REWIND	Regular	Retain	No Rewind
PURGE	Regular	Release	Purge
RELEASE	Regular	Release	Rewind
CRUNCH	Regular	Release	Crunch
REMOVE	Regular	Release	Save
REMOVE CRUNCH	Regular	Release	Crunch
NO WAIT	Dontwait	Retain	Rewind

Table 2–9. Contents of Column 70 of the RPG File Description Specification

Identifier	Type	Association	Disposition
R: Remove	Regular	Release	Save
C: Crunch	Regular	Release	Crunch
K: Crunch and remove	Regular	Release	Crunch

Note: *Can be used only with disks.*

Table 2–10. Contents of Column 70 of the RPG File Description Specification

Identifier	Type	Association	Disposition
P: Purge	Regular	Release	Purge
U: Unload	Regular	Release	Lock

Table 2–10. Contents of Column 70 of the RPG File Description Specification

Identifier	Type	Association	Disposition
N: No Rewind	Regular	Retain	No Rewind **Note:** Can be used only with tapes.
Blank: If input disk that does not allow addition or deletion: Regular	Regular	Release	Rewind
Blank: All other types of files: Unspecified	Regular	Release	Save

Table 2–11. Contents of the Result Field (Columns 43 through 48) of the RPG Calculation Specification When CLOSE Is Present in Columns 28 Through 32

Contents of Column 43 Through 48	Type	Association	Disposition
Blank	Regular	Release	Rewind
CRUNCH	Regular	Release	Crunch
LOCK	Regular	Release	Lock
NORWND	Regular	Retain	No Rewind
PURGE	Regular	Release	Purge
REEL	Reel	Retain	Rewind
REWIND	Regular	Retain	Rewind

Table 2–12. FORTRAN77 CLOSE Statement Actions

Statement Option	Type	Association	Disposition
"CRUNCH"	Regular	Release	Crunch
"DELETE"	Regular	Release	Purge
"KEEP"	Regular	Release	Lock
None, but the PROTECTION file attribute value is TEMPORARY	Regular	Release	Purge

Table 2–12. FORTRAN77 CLOSE Statement Actions

Statement Option	Type	Association	Disposition
None, but the PROTECTION value is SAVE or PROTECTED	Regular	Release	Lock

Table 2–13. FORTRAN77 CLOSE Actions without a CLOSE Statement

Statement	Type	Association	Disposition
REWIND calls support library, which does SEEK[0] if the file is a disk and the BLOCKSTRUCTURE file attribute value is not LINKED, otherwise the file is closed with the identified type, association, and disposition.	Regular	Reserve	Rewind
ENDFILE calls support library, which opens the file if not already open and the closes the file with the identified type, association, and disposition.	Regular	Reserve	No Rewind

Table 2–14. Pascal CLOSE Statement Actions

Statement Option	Type	Association	Disposition
Default	Regular	Release	Rewind
CRUNCH	Regular	Release	Crunch
NOREWIND	Regular	Retain	No Rewind
PURGE	Regular	Release	Purge
RESERVE	Regular	Reserve	No Rewind
SAVE	Regular	Release	Save

Table 2–15. ALGOL CLOSE Statement Actions

Statement Option	Type	Association	Disposition
Default	Regular	Release	Rewind
*	Regular	Retain	No Rewind
CRUNCH	Regular	Release	Crunch
LOCK	Regular	Release	Lock

Table 2–15. ALGOL CLOSE Statement Actions

Statement Option	Type	Association	Disposition
PURGE	Regular	Release	Purge
REEL	Reel	Retain	Rewind
REWIND	Regular	Retain	Rewind
DONTWAIT	Dontwait	Retain	Rewind
DOWNSIZEAREA	Regular	Release	Downsizearea
DOWNSIZEAREALOCK	Regular	Resease	Downsizearealock

Table 2–16. ALGOL CLOSE Actions without a CLOSE Statement

Statement	Type	Association	Disposition
LOCK (file name)	Regular	Release	Lock
LOCK (file name, CRUNCH)	Regular	Release	Crunch
REWIND (file name)	Regular	Retain	Rewind

Modifying an Attribute

You can modify the value of a file attribute, if allowed, for a particular purpose. You can change the value under the conditions allowed for by any given file attribute. Some attributes are not modifiable; they are read-only. Some attributes are modifiable only while the file is closed, and some are modifiable only while the file is unassigned. The attributes that are modifiable only while the file is closed are associated with the structure of the logical file. The attributes that are modifiable only while the file is unassigned are associated with the structure of the physical file or are used to specify the matching criteria for assigning the physical file to the logical file.

You can use the following methods to modify an attribute in ALGOL, COBOL74, and COBOL85:

ALGOL

```
F.NEWFILE := TRUE;

REPLACE TEST_FILE.FILENAME BY "TEST/PROGRAM.";

TERM.BLOCKSTRUCTURE := VALUE(EXTERNAL);

REPLACE PORTFILE(5).YOURNAME BY "YOU.";
```

COBOL74 and COBOL85

```
CHANGE ATTRIBUTE DEPENDENTSPECS OF
    IN-FILE TO VALUE TRUE.

CHANGE ATTRIBUTE FILENAME OF
    OUT-FILE TO "OUTPUT/MASTER.".

CHANGE ATTRIBUTE BLOCKSTRUCTURE OF
    TERMINAL-FILE TO VALUE EXTERNAL.
```

Interrogating an Attribute

Many file attribute values are changed because of conditions that occur during an I/O operation. You can find out the current value of any given attribute by interrogating that attribute. Based on the current value, your program can then use a specific procedure to handle the current condition. You can interrogate most attributes at any time. However, you can interrogate other attributes only under the following conditions:

- While the file is open. These attributes report part of the current state of the logical file.
- While the file is assigned. These attributes report part of the current state of the physical file.

Keep in mind that certain state-related file attribute values, such as LASTSUBFILE, STATE, and CURRENTRECORDLENGTH, change with every I/O operation performed on the logical file. If the logical file is being accessed by more than one task, there is no guarantee that the value of that file attribute when a given task interrogates the attribute is the value resulting from the I/O operation that the task invoked.

If you want to perform an I/O operation and then obtain information about that operation, you must establish a method of ensuring that no other task can perform an I/O operation on the logical file in the time interval between the I/O operation and the interrogation of one or more state-related file attributes.

One method you can use to request information about an I/O operation invoked by a task is to request the information in the I/O statement. For example, the following statement returns the subfile index (the value obtained by interrogating the LASTSUBFILE attribute) in INX and the result descriptor (the value obtained by interrogating the STATE attribute) in B. The current record length is found in field [47:20] of the result descriptor.

```
B := READ (PORTF[SUBFILE INX:0], 72, IOBUF)
```

The following table lists a few of the general purpose file attributes that you can interrogate:

Attribute	Information Obtained
BUFFERS	The number of buffers currently being used
CURRENTBLOCKLENGTH	The size, in FRAMESIZE units, of the block currently being used
CURRENTRECORDLENGTH	The number of FRAMESIZE units in the last record read or written
FILEEQUATED	Indication of whether one or more file attributes have been modified by file equation
IOCLOCKS	The accumulated I/O time for the file
NEXTRECORD	The current position in the file
RESULTLIST	A list of results caused by the most recent logical I/O operation performed

The following examples show syntax that can be used to interrogate a file attribute value in ALGOL, COBOL74, and COBOL85:

ALGOL

```
I := F(2).CENSUS;

B := F.ATTERR;

REPLACE POINTER(TEST_FILENAME) BY
     TEST_FILE.FILENAME;

IF IN_FILE.BLOCKSTRUCTURE NEQ VALUE(FIXED) THEN
     HANDLE_VARIABLE_FILE;
```

COBOL74 and COBOL85

```
MOVE ATTRIBUTE CRUNCHED OF IN-FILE TO
SAVE-CRUNCH.

MOVE ATTRIBUTE FILENAME OF OUT-FILE TO
    FILENAME-TEMP.

IF ATTRIBUTE BLOCKSTRUCTURE OF IN-FILE IS
        NOT EQUAL TO VALUE FIXED
    PERFORM HANDLE-VARIABLE-FILE.
```

Determining Attribute Conflicts

Sometimes you change attribute values to values that cannot be combined. As a result, you receive an attribute error. You can detect attribute errors within your program by using the ATTERR, ATTVALUE, and ATTYPE attributes.

Limiting Code File Execution

If you want only one copy of a code file to be executed, you can have the code within the code file open a companion data file with the EXCLUSIVE attribute set to TRUE, or have the code call a shared library that provides some form of exclusivity mechanism.

Dealing with Translation

This guide uses the term *translation* to mean the process of replacing one data character with a corresponding data character from another character representation, rather than the process of expressing the meaning of a group of words written in one language in another language.

When input data is, or output data must be, in a character set that cannot be processed effectively by your program or cannot be manipulated efficiently by the system, translation is necessary. The EXTMODE and INTMODE attribute values control whether or not such translation occurs. If the two values identify different character sets, and if the combination can be dealt with, translation is automatically initiated. Additionally, translation can be requested if the EXTMODE and INTMODE values are the same.

The EXTMODE value represents the character set of the physical file, while the INTMODE value represents the character set the program uses to manipulate the data. Table 2–14 shows the acceptable translation combinations. If the EXTMODE value is BINARY, translation can never occur.

Normally, when an existing physical file is opened, the EXTMODE value of the logical file unconditionally assumes the EXTMODE value of the physical file. Using the OVERRIDEEXTMODE attribute causes the logical file EXTMODE value to override the EXTMODE of the physical file, thus causing the MCP to treat the physical file as though it had the EXTMODE you specified. For more information on the effects of the OVERRIDEEXTMODE attribute on file translation, see the *File Attributes Reference Manual*.

Table 2–17. Possible EXTMODE and INTMODE Combinations

EXTMODE		INTMODE												
		0	**2**	**3**	**4**	**5**	**7**	**8**	**9**	**10**	**11**	**n**	**m**	**mm**
ISOGENERALSTRING	**7**	N	–	–	–	–	N	–	–	–	–	–	–	–
ISOGRAPHICSTRING	**8**	N	–	–	–	–	–	N	–	–	–	–	–	–
ISOVISIBLESTRING	**9**	N	T	–	T	T	–	–	N	T	–	T	–	–
IASTRING	**10**	N	T	–	T	T	–	–	T	N	–	T	–	–
OCTETSTRING	**11**	N	–	–	–	–	–	–	–	–	N	–	–	–

Note: *FTAM-specific values.*

Table 2–18. Possible EXTMODE and INTMODE Combinations

EXTMODE		INTMODE												
		0	**2**	**3**	**4**	**5**	**7**	**8**	**9**	**10**	**11**	**n**	**m**	**mm**
SINGLE	**0**	N	N	N	N	N	N	N	N	N	N	N	–	–
HEX	**2**	N	N	–	Y	Y	–	–	T	T	–	T	–	–
BCL	**3**	N	–	N	–	–	–	–	–	–	–	–	–	–
EBCDIC	**4**	N	Y	–	N	Y	–	–	T	T	–	T	T	–
ASCII	**5**	N	Y	–	Y	N	–	–	T	T	–	T	–	–
Other 8-bit sets	**n**	N	T	–	T	T	–	–	T	T	–	*	T	T
16-bit sets	**m**	–	–	–	T	T	–	–	–	–	–	T	*	T
Mixed multibyte sets	**mm**	–	–	–	T	T	–	–	–	–	–	T	T	*

Legend

–	Invalid combination; an OPEN error is issued.
N	Valid combination; no translation is performed.
Y	Valid combination; translation occurs using the translate tables of the operating system.
T	Valid combination; translation is performed if the appropriate translation tables are available in the CentralSupport library. If appropriate translation tables are not available, an OPEN error is issued.
*	If the INTMODE and EXTMODE values are the same, handled like an N; otherwise, handled like a T.

For information about acceptable translation combinations for natural language character sets identified as "Other 8-bit sets" in Table 2–14, refer to the *MultiLingual System Administration, Operations, and Programming Guide,* and for information about acceptable translation combinations for FTAM files, refer to "Using FTAM" in Section 9 of this guide.

Translation cannot occur in the following situations:

- When the EXTMODE or INTMODE value is SINGLE.
- When the BLOCKSTRUCTURE value is LINKED or VARIABLEOFFSET, or the FILETYPE value is 4 or 6.
- When the file is a direct I/O file.
- When a port file has a service other than BNANATIVESERVICE (Version 2 only).
- When either character set is mixed multi-byte (mixed 8-bit and 16-bit with FRAMESIZE = 8*)*.

If you are creating the file, you can assign values to both attributes, but if you are accessing a permanent file, the logical file EXTMODE value is overwritten by the EXTMODE value of the permanent file.

When translation is occurring, the values of the AREALENGTH, BLOCKSIZE, CURRENTBLOCKLENGTH, CURRENTRECORDLENGTH, MAXRECSIZE, and MINRECSIZE attributes are returned in terms of the logical units of the file as defined by the FRAMESIZE attribute. Data are processed by the I/O subsystem in terms of logical records, and both character- and word-oriented data transfers are allowed. In fact, the whole process is transparent to the program.

When the character frame sizes differ between the internal mode and the external mode, word-oriented data transfers require either contraction or expansion of the records; hence, the logical and physical record and block sizes are not the same. For example, if a file with an EXTMODE value of HEX were created with a physical record size of 5 words, the translation of the 60 characters to 60 EBCDIC characters for a file with an INTMODE value of EBCDIC would require 10 words in the logical record.

If you are a COBOL programmer, you should be aware that the value of the INTMODE attribute is dependent on the order of the 01-level items under a File Description (FD) entry. If two programs have the same record descriptions but the 01-level entries occur in a different order, the compiler sets the INTMODE values differently and causes translation to occur unexpectedly. For example, a first 01-level entry with USAGE DISPLAY sets the INTMODE value to EBCDIC, and a first 01-level entry with USAGE COMP sets the INTMODE value to HEX. Thus, make sure that the order of the 01-level items is the same for all programs using a given file. The record descriptions can be stored in a copy library file to ensure that all FD descriptions have the same order of record declaration.

You can determine whether or not translation is occurring by interrogating the TRANSLATING attribute. If the value is TRUE, translation is occurring. To specify that the code in your program will perform any necessary translation, change the TRANSLATE attribute value to USERTRANS. This action prevents the system from performing any translation.

In some languages, you can cause translation to occur by using translation tables that you provide and by performing the following tasks:

- Declare the appropriate translation table in your program. The table must be the first table in a list of translation tables or the only table in the declaration.
- If the INTMODE and EXTMODE values are not forcing software translation to occur, assign the TRANSLATE attribute a value of FORCESOFT.
- If the translation is to be done as the records are being read, assign the INPUTTABLE attribute the name of the translate table.
- If the translation is to be done as the records are being written, assign the OUTPUTTABLE attribute the name of the translate table.

Note: *Each time the logical file is closed, the INPUTTABLE and OUTPUTTABLE values are discarded.*

Double-Byte and Mixed Multi-Byte Character Sets

Double-byte (16-bit) character sets and mixed multi-byte (mixed 8-bit and 16-bit with FRAMESIZE = 8) character sets are implemented subject to the following rules and restrictions:

- KIND must be DISK or PRINTER.
- If KIND = PRINTER, BACKUPKIND must be DISK or PACK, BLOCKSTRUCTURE must be FIXED, and INTMODE must equal EXTMODE.
- FILEORGANIZATION must be NOTRESTRICTED.
- When either INTMODE or EXTMODE specifies a mixed multi-byte character set, translation can occur only when the file is a character-stream disk file. A character-stream disk file is a character-oriented file with FILESTRUCTURE = STREAM, ANYSIZEIO = TRUE, and MAXRECSIZE = 1. This type of file is also referred to as a FILECLASS = CHARACTERSTREAM file.
- For the purposes of mixed multi-byte translation, the entire character-stream disk file is treated as a single character string. Any escape into or out of a sub-character set takes place across individual I/O statements and applies to the file as a whole.
- When mixed multi-byte translation is in effect, only serial I/O operations are permitted, and the file must be used only for input or only for output, but not for both input and output.
- No individual WRITE statements can transfer more than 32,767 bytes from the record area of a program; an attempt to do so causes a data error result to be returned to the program.
- BLOCKSTRUCTURE must be FIXED, VARIABLE, or EXTERNAL

  If BLOCKSTRUCTURE = VARIABLE, the file must be character oriented (UNITS = CHARACTERS), and the file must have 8-bit EBCDIC-based, ASCII-based, UCS2, or UCS2NT characters. Refer to the *MultiLingual System Administration, Operations, and Programming Guide* for more information about these character sets.

- Host Services Logical (I/O) or FTAM File Services are not supported.
- Binary I/O is not supported.
- Synchronous I/O is not supported.
- Buffer sharing is not supported.
- User-defined translate tables are not supported.

Understanding Logical File Visibility in the Multiple Stack Situation

In most cases, only one stack can perform an operation on a logical file at a time, and other stacks that attempt simultaneous operations wait until the current operation is finished.

Understanding this concept is especially important when the operation being performed can take an indefinite period of time to complete. Examples of such operations are

- An OPEN or AWAITOPEN operation that can wait indefinitely for a file that is not present or that is inaccessible for some other reason
- A WRITE operation that might result in a reel switch (for tape files) or flow control action (for port files)

To avoid waiting indefinitely on an operation

- When opening files, use the AVAILABLE control option in your OPEN or AWAITOPEN statement.
- For port files, use the DONTWAIT control option in your I/O statements in ALGOL, or the NO WAIT control option in your I/O statements in COBOL74.

Section 3
Using Disk and CD-ROM Files in a Program

In the MCP environment disk files and compact disk files have different KIND file attribute values. Files that reside on magnetic disk files are identified with the PACK value and files that reside on CD-ROM media are identified with the CD value.

Files with a KIND Value of PACK or DISK

Disk files are stored on devices that have a storage medium that is identified by a family name of up to 17 characters. These family names are associated with the medium by using the RC (Reconfigure Disk) or LB (Relabel Pack) system commands. Some family names point to a group of disks that are logically grouped together as a multidisk family. Multidisk families allow your site to group large amounts of information together.

As your program writes records to a file, the disk subsystem places the records on the tracks of the disk that are physically divided into portions known as *sectors*. Each sector contains 30 words (180 bytes) and has a unique address that the system uses to identify the location of the sector. A sector is the smallest portion that can be read from or written to a disk.

Because storing all the records of a file in one contiguous sequence of sectors is not necessarily an efficient use of disk space, the disk space is allocated in pieces known as *areas*, which are allocated by the system as needed. Areas are allocated anywhere on a disk unit family or on any one of the disk units of a multidisk family. Allocation of areas is done on a rotational basis for a multidisk family, but a program can indicate that the areas of a file be allocated on a specific disk unit within the multidisk family when the file is created. Refer to "AREAS," "AREALENGTH," "AREASIZE," "FILESTRUCTURE," "FLEXIBLE," "LASTACCESSIBLEAREA," "MAXRECSIZE," and "MINRECSIZE" in the *File Attributes Reference Manual* for more information about understanding disk file size limits.

When the file becomes a permanent file, a disk file header describing the file is placed in the disk directory. A disk directory resides on the base unit of a multidisk family, unless the duplicate directories have been requested by using the DD (Directory Duplicate) system command. When duplicated directories exist, the base unit is the first member of the family containing a directory that the system is aware of when the family becomes available.

A disk file can be differentiated from another disk file with the same file name by the ANSI standard technique of recording a cycle and version for any file name. Different iterations of a file that have the same cycle value are of the same genealogy. Additionally, you can have multiple version numbers within a cycle. A file is said to have the *best* genealogy when it has the highest cycle value and the highest version value within the highest cycle. Only one file with the same name can be resident on a disk at any one time, but other iterations can be saved on library maintenance tapes.

In the MCP environment, there are two methods of locating records on the physical disk. One method groups records in blocks, whose size is defined by the program, and locates these blocks on the disk at sector boundaries. Such a file is identified by a FILESTRUCTURE attribute value of ALIGNED180 or BLOCKED. The other method allows a continuous stream of records that are located on the disk without regard for sector or area boundaries. Such a file is identified by a FILESTRUCTURE attribute value of STREAM. The default value of the FILESTRUCTURE attribute is ALIGNED180.

Files with a FILESTRUCTURE value of STREAM make the best use of disk space and are the easiest to use. However, they, as well as files with a FILESTRUCTURE value of BLOCKED, do not support some of the features that are available for files with a FILESTRUCTURE value of ALIGNED180. Refer to the discussion about the FILESTRUCTURE attribute in "Creating a New Disk File" later in this section for information about these restrictions.

A program should not assume that data written by a non-DIRECT WRITE operation has successfully reached the disk unless one of the following is true:

- The WRITE operation was a synchronous operation and a good result descriptor was returned.
- The WRITE operation was followed later by a synchronous WRITE operation and good result descriptors were returned for all the READ, WRITE, and SEEK operations to the logical file starting with the subject WRITE operation and ending with the synchronous WRITE operation.
- The WRITE operation was followed later by a CLOSE operation and good result descriptors were returned for the CLOSE operation and for all the READ, WRITE, and SEEK operations to the logical file starting with the subject WRITE operation and ending with the CLOSE operation.

If you are not familiar with basic programming methods, review those methods in Section 2, "Understanding Programming for Files," before reading the following procedures. You can identify all the file attributes that can be used with a disk file by reviewing Table A–1. For more information about any of the file attributes, refer to the *File Attributes Reference Manual*.

Creating a New Disk File

The task of creating a new disk file has been broken down into the following groupings:

- Tasks that are required
- Tasks that ensure file security
- Tasks that define complex record structures
- Tasks that allow for special handling of the file
- Tasks that store information with the file
- Tasks that create the file

Note: *You can create additional copies of a disk with the Mirrored Disk feature. Refer to the* System Administration Guide *for information.*

Required Tasks

You should perform all of the following tasks:

- Specify the KIND value as DISK. The default value is DONTCARE.
- Indicate that the file is being created by setting the NEWFILE attribute value to TRUE.
- Determine on which family the new file should be placed.
- If you want the physical file to have a name that is different from the internal name of the file, specify the physical file name by using the FILENAME or TITLE attribute.
- If you did not identify the family name of the disk that you want the file to reside on in the TITLE attribute, identify the family by using the FAMILYNAME attribute. If you did specify the family name by using the ON option, the KIND file attribute value is now PACK. If you do not specify the family name, the I/O subsystem uses the default value of DISK. If you use family substitution, the system can change the family name to the family designated by the family specification.
- If you want to create a file that contains a continuous stream of records that is not divided into blocks, assign the FILESTRUCTURE attribute a value of STREAM. Such files are easier to use than files with a FILESTRUCTURE value of ALIGNED180 or BLOCKED and make the most use of disk space, but they do not support the following features:
  - Update I/O with synchronization
  - Binary I/O
  - A BLOCKSTRUCTURE value of LINKED
  - A FILEORGANIZATION value other than the NOTRESTRICTED value
  - Checkpoint or restart capabilities
  - Duplicated files
  - CANDE work files

The buffer size of a file with a FILESTRUCTURE value of STREAM is automatically provided by the I/O subsystem. You can modify the buffer size by using the BUFFERSIZE attribute, but doing so is not normally recommended.

- If you want to create a file that contains records that are grouped into a block of records, assign the FILESTRUCTURE attribute a value of BLOCKED. The FILESTRUCTURE default value is ALIGNED180. If you choose the BLOCKED value, the following features are not supported:
  - Update I/O with synchronization
  - Binary I/O
  - A BLOCKSTRUCTURE value of LINKED
  - A FILEORGANIZATION value other than the NOTRESTRICTED value
  - Checkpoint or restart capabilities
  - Duplicated files
  - CANDE work files
  - Host Services logical I/O
- Identify how the data is going to be transferred by using the FRAMESIZE attribute. Refer to "Identifying How Data Is Transferred" in Section 2.
- If you want an INTMODE attribute value other than the default value assumed by your language, assign that value to INTMODE. Refer to "Identifying How Data Is Transferred" in Section 2 for language default information. The INTMODE value is assumed by the EXTMODE value. If the physical file must have a character encoding set that is different from the INTMODE value, assign EXTMODE the appropriate value. Refer to Table 2–14 for possible EXTMODE and INTMODE combinations.
- Identify the maximum size of any record by using the MAXRECSIZE attribute. You should express the MAXRECSIZE value in terms of FRAMESIZE units. A disk file defaults to 30 words.
- If you chose to have a FILESTRUCTURE value of BLOCKED or ALIGNED180, and if you want more than one record to be placed in a block, use the BLOCKSIZE attribute. The value you assign is dependent on the type of records and should be expressed in terms of FRAMESIZE units. If the records are variable-length records, the value should be greater than the MAXRECSIZE value. If the records are fixed-length records, the value should be a multiple of the MAXRECSIZE value. If you do not assign a value to BLOCKSIZE, BLOCKSIZE assumes the MAXRECSIZE value and only one record is put in each block.

  If you chose a FILESTRUCTURE value of BLOCKED, the buffer size in memory that is used by the physical I/O operation is dependent on the BUFFERSIZE attribute value. The default value for the BUFFERSIZE attribute is usually appropriate for most users. You can choose a BLOCKSIZE value that groups a certain number of records together, but be aware that more than one block of records can be read into the buffer that is controlled by the BUFFERSIZE attribute.

  If you chose a FILESTRUCTURE value of ALIGNED180 or accepted ALIGNED180 as the default value and the file resides on VSS-2 disks, write operations to the file can be optimized by setting the MAXRECSIZE or BLOCKSIZE attributes to span an even number of 30-word disk sectors.

If you chose to use the default FILESTRUCTURE value of ALIGNED180, the buffer size in memory is dependent on the BLOCKSIZE value. Keep the following information in mind as you determine what the BLOCKSIZE value should be:

- If you use large blocks, the I/O operations are efficient, but your program is tying up a large amount of main memory.
- If you use very small blocks, the I/O subsystem must perform more I/O operations.
- If you are going to use Host Services logical I/O with this file, the BLOCKSIZE attribute must be less than 65486 characters.

- When a new disk file is created if the specified value of the AREAS file attribute exceeds the number of areas occupied by a file with the largest possible end-of-file and the same area length, the value of AREAS is reduced.

  Use the LASTACCESSIBLEAREA attribute to determine the highest area number accessible in an open disk file. Refer to "AREAS" and "LASTACCESSIBLEAREA" in the *File Attributes Reference Manual* for more information.

  If you want to control the size of an area rather than use the default size, specify a value for the AREALENGTH attribute. This attribute should not be modified for files with a FILESTRUCTURE value of STREAM or BLOCKED, unless the program has special requirements. The value you specify should be in FRAMESIZE units and cannot be larger than 16777215 sectors. When deciding the value for the AREALENGTH attribute take into account the following:

  - Areas that are too small limit the number of records in the file because of the limit on the number of areas.
  - Areas that are too large make it more difficult for the system to find the contiguous disk sectors needed to store each area.

  The following default values for the AREALENGTH attribute are based on the FILESTRUCTURE value of the file:

FILESTRUCTURE Value	AREALENGTH Default Value
ALIGNED180	The MAXRECSIZE value multiplied by 1000 and rounded up so the value is also a multiple of the BLOCKSIZE value.
BLOCKED	A multiple of BLOCKSIZE closest to but not exceeding 184320 bytes, or BLOCKSIZE if BLOCKSIZE exceeds 184320.
STREAM	A multiple of MAXRECSIZE closest to but not exceeding 184320 bytes, or MAXRECSIZE if MAXRECSIZE exceeds 184320 bytes.

Complex Record Tasks

You should perform the following tasks if you want your file to have complex record structures, such as variable-length records.

- To establish a complex record format, refer to "Establishing a Record Format" in Section 2.
- If you want the file to be accessed in a certain manner, use the FILEORGANIZATION attribute. The following are the mnemonic values that you can select from:

Mnemonic Value	Meaning
KEYEDIOII	The physical file is created as a KEYEDIOII data file, and it is implemented through the KEYEDIOII library. For information about KEYEDIOII, refer to the *KEYEDIOII Reference Manual*.
KEYEDIOIISET	The physical file is created as a KEYEDIOII index file, and it is implemented through the KEYEDIOII library. For information about KEYEDIOII, refer to the *KEYEDIOII Reference Manual*.
NOTRESTRICTED	No restrictions are applied. This is the default value.
RELATIVE	The file has a relative file organization. This organization is derived from COBOL74.

- You can indicate the internal structure of the records of a file by assigning a value to FILEKIND. For example, files with a FILEKIND value of ALGOLSYMBOL are expected to contain properly formatted ALGOL source program records. If you do not specify a FILEKIND value, a value of DATA is used. Refer to the FILEKIND attribute in the *File Attributes Reference Manual* for an entire list of values.

Special Requirement Tasks

Consider performing the following tasks if you have special requirements:

- If you want the system to assign a unique file name every time the file is opened, do the following:
  - Specify the displayable character that identifies the variable information in the file name with the UNIQUETOKEN file attribute.
  - Specify a FILENAME value that includes the displayable character followed by a keyword that identifies the system-supplied character string that should be inserted.

  Refer to the FILENAME attribute in the *File Attributes Reference Manual* for more information.
- If you want your logical file to share buffers with another logical file, refer to "Sharing Buffers with Other Files" later in this section for instructions.

- If you know that the file will require more than 20 areas, the default, indicate the number of areas that the I/O subsystem can allocate by using the AREAS attribute. You can specify up to 15000 areas. Normally, the system adds more areas to the file, as they are needed. However, if you do not want the file to contain more areas than you specified with the AREAS attribute, set the FLEXIBLE attribute to FALSE. The FLEXIBLE attribute value is not stored with the file, so any updating programs that use the file must have the FLEXIBLE attribute set to FALSE, also.

  When the file is created, the number of areas you designated in the AREAS attribute is not automatically allocated. Instead, the system leaves an empty entry in the disk file header for each possible area. Then, as the system allocates an area to the file, it places the address of the area in the header.

- If you want to specify whether POSIX or native MCP environment rules are to be used in searching for an existing file or creating a new file, use the SEARCHRULE file attribute. Refer to the *File Attributes Reference Manual* for details on the SEARCHRULE file attribute.

- You can protect your disk file in two ways by using the PROTECTION attribute. The following values are available:

Mnemonic Value	Meaning
SAVE	The file is entered into the disk directory immediately after the file is opened, instead of when the file is closed with the SAVE option.
PROTECTED	The file is entered into the disk directory immediately after the file is opened, and special action is taken to ensure that the correct end-of-file pointer is maintained across a system failure. If the FILESTRUCTURE value of the file is STREAM, the end-of-file marker is placed at the end of the disk sector that was last written, even though the end of that sector might not be the end of a record. Do not use this value if you have set the SYNCHRONIZE attribute to OUT.

- If you want to ensure that your disk file cannot be removed or replaced after it is entered into the disk directory, and that the name of the file cannot be changed, set the LOCKEDFILE file attribute to TRUE. A permanent file with a LOCKEDFILE value of TRUE cannot be closed with the PURGE option unless you or a privileged user change the LOCKEDFILE value to FALSE.

- For multidisk families, the areas of the file are allocated on a rotating basis among the members. You can modify this behavior in one of two ways:

  – To place the file areas on a specific family member, identify the specific family member number by using the FAMILYINDEX attribute. Be aware that restricting allocation to a specific family member might make space more difficult to find.

  – To place all the areas on a single family member selected by the system, set the SINGLEUNIT attribute to TRUE.

- If you want your program to force a physical WRITE operation to happen every time a WRITE statement is invoked, and then to wait until the completion of the WRITE operation to the physical file, assign the SYNCHRONIZE attribute a value of OUT. Usually the data is written to disk in an asynchronous manner. Writing to a disk in a synchronous manner ensures that data is written, but adds overhead. The FILEORGANIZATION value of the file must be NOTRESTRICTED to use the SYNCHRONIZE attribute. The value of the SYNCHRONIZE attribute is not stored permanently with the file.

  If you want your program to occasionally ensure that a WRITE operation has completed before going on to the next instruction in the program, use a WRITE statement with the SYNCHRONIZE option each time you want this behavior.
- If you chose to set the FILESTRUCTURE value to STREAM and the BLOCKSTRUCTURE value to FIXED, you might also want to indicate that the transfer of any number of frames in a single I/O operation is not constrained by the MAXRECSIZE attribute. To do so, set the ANYSIZEIO attribute to TRUE. The ANYSIZEIO value is not stored permanently with the file.
- If your system is using cataloging and you want the file to be entered into the system catalog, set the USECATALOG attribute to TRUE. If the USECATDEFAULT system option is enabled, the default value of the USECATALOG attribute is TRUE.
- If you want the file to have cycle and version information other than the default values of 1 and 0 (zero), respectively, specify the desired values by using the CYCLE and VERSION attributes. If you specify a value of 0 for the CYCLE attribute, the values of the CYCLE and VERSION attributes are changed to their default values.
- If you do not want your program to wait for disk space to become available when a new area is allocated, set the NORESOURCEWAIT attribute to TRUE. The NORESOURCEWAIT value is not stored permanently with the file. You should be aware that the system waits, even if the value is TRUE, in the following situations:
  - If the file is protected
  - If a temporary file has been closed with a disposition of lock that causes sectors to be required for the directory
  - If the file is flexible, and directory sectors are required when increasing the header

Information Storage Tasks

Consider performing the following tasks to store information that can be used by other programs:

- If the data of the file should be processed according to a specific coded character set, or language or cultural rules, assign the CCSVERSION attribute a value. For more information about the CCSVERSION attribute, refer to the *File Attributes Reference Manual*. For information about the rules refer to the *MultiLingual System Administration, Operations, and Programming Guide.* The following values are available:

Mnemonic Value	Meaning
ARABIC20	Arabic ccsversion (previously known as "Version 20")
ASERIESNATIVE	Default ccsversion
BRAZILIAN	Brazilian ccsversion
CANADAEBCDIC	CANALPHA1 ccsversion
CANADAGP	CANASUPPL ccsversion
CZECHOSLOVAKIA	Czechoslovakia ccsversion
FRANCE	French ccsversion
HUNGARIAN	Hungarian ccsversion
LATINGREEK	Latin-Greek ccsversion
NORWAY	Norwegian ccsversion
POLISH	Polish ccsversion
ROMANIAN	Romanian ccsversion
RUSSIAN	Russian ccsversion
SPANISH	Spanish ccsversion
SWEDISH1	Swedish ccsversion
SWISS	Swiss ccsversion
TURKISH	Turkish ccsversion

- If you want to store site- or application-specific information about the file in the file header, use the USERINFO attribute.
- If you want a certain amount of information to be printed out when the file you are creating is printed out, use the NOTE attribute. You can store up to 250 characters.
- If you want to store any software-release-specific information with the file, use the RELEASEID attribute. You can store up to 250 characters.

Creating the File

After you have finished the preceding tasks that assign the various file attributes, you can open the file. When you open the file, the system creates a disk file header for the file and stores the values of the various permanent 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 the FILELENGTH attribute, and places the header in the flat directory of the disk if the file is closed with a disposition of lock or crunch. Refer to Section 2, "Understanding Programming for Files" for information about closing files and their associated dispositions.

Accessing an Existing Disk File

Perform all or some of the following tasks, depending on the needs of your program:

- Assign the KIND attribute a value of DISK.
- Determine on which family the existing file resides.
- If the physical file name is different from the internal file name of the file, specify the physical file name by using the FILENAME or TITLE attribute.
- If you did not identify the family name of the disk where the file resides in the TITLE attribute, identify the name of the family where the file resides by using the FAMILYNAME attribute. If you did specify the family name by using the ON option, the KIND file attribute value is now PACK.
- Assign the NEWFILE attribute a value of FALSE.
- If you want your logical file to share buffers with another logical file, refer to "Sharing Buffers with Other Files" later in this section for instructions.
- If you want to process the data in the character set of the physical file rather than requesting translation to the character set of the logical file, assign the DEPENDENTINTMODE attribute a value of TRUE. The default value is FALSE.
- It is recommended that you set the DEPENDENTSPECS attribute value to TRUE, unless you have a special reason to manipulate the records differently than the creation program intended.

  If you do not set DEPENDENTSPECS to TRUE and you specify a BLOCKSIZE, MAXRECSIZE, or MINRECSIZE value for a file with any FILESTRUCTURE value that is inconsistent with the values the file was created with, an OPEN error results.
- If you want to access a specific cycle and version of the file, specify the appropriate value in the CYCLE and VERSION attribute. 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 values is online, the system displays an "UNMATCHED GENEALOGY" message on the ODT.

If you specify CYCLE but not VERSION, the system locates the generation with that exact CYCLE value and the VERSION value 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.

- If you want the file to be used by your program exclusively, set the EXCLUSIVE attribute to TRUE. If another program is currently using the desired file, the I/O subsystem does not assign the file to your program until the other program closes that file.
- If you want your program to force a physical WRITE operation to happen every time a WRITE statement is invoked, and then wait until the completion of the WRITE operation to the physical file, assign the SYNCHRONIZE attribute a value of OUT. Usually the data is written to disk in an asynchronous manner. Writing to a disk in a synchronous manner ensures that data is written, but adds overhead. The FILEORGANIZATION value of the file must be NOTRESTRICTED to use the SYNCHRONIZE attribute.

  If you want your program to occasionally ensure that a WRITE operation has completed before going on to the next instruction in the program, use a WRITE statement with the SYNCHRONIZE option each time you want this behavior.
- If the FILESTRUCTURE value of the file is STREAM and the BLOCKSTRUCTURE value is FIXED, you might also want to indicate that the transfer of any number of frames in a single I/O operation is not constrained by the MAXRECSIZE attribute. To indicate this, set the ANYSIZEIO attribute to TRUE.
- If you want a serial WRITE operation to rewrite the changes made to the record that was just serially read, set the UPDATEFILE attribute to TRUE. If you set UPDATEFILE to TRUE when the FILESTRUCTURE value is STREAM or BLOCKED, you prevent synchronization. If UPDATEFILE is not set to TRUE, the next serial WRITE operation writes the changes to the next sequential record.
- If your system uses cataloging, you can perform one or both of the following tasks. Refer to the *System Administration Guide* for more information about cataloging.
  - If you want the system catalog searched when the system is seeking a permanent disk file, set the USECATALOG attribute to TRUE. If the USECATDEFAULT system option is enabled, the value of USECATALOG defaults to TRUE.
  - If you want to select a copy of the file that does not have the latest time and date, identify the copy by using the GENERATION attribute.

- If you want to specify whether POSIX or native MCP environment rules are to be used in searching for an existing file or creating a new file, use the SEARCHRULE file attribute. Refer to the *File Attributes Reference Manual* for details on the SEARCHRULE file attribute.
- If you do not want your program to wait for disk space to become available when a new area is allocated or an old area is activated, set the NORESOURCEWAIT attribute to TRUE. Your program must be prepared to handle the no space error. You should be aware that the system waits, even if the value is TRUE, in the following situations:
  - The file is protected.
  - The file is flexible, and directory sectors are required when the header is increased.

Obtaining Information about a Disk File

When your file is open, you can interrogate several attributes to obtain information about the file. Tables 3–1 through 3–3 identify these attributes.

Table 3–1 identifies some attributes that might be of interest to you that have information that remains constant while the file is open. Most attributes can be interrogated.

Certain file attributes have values that are expressed in terms of blocks or provide some information about blocks. These attributes include BLOCK, BLOCKSIZE, and CURRENTBLOCK. For files with variable-length records, the attributes include AREASIZE and LASTRECORD.

Interrogating these attributes for a file with a FILESTRUCTURE value of STREAM produces an attribute error, because such files are not composed of blocks. Because it is possible for a file of any FILESTRUCTURE value to be opened when DEPENDENTSPECS equals TRUE or when FILETYPE equals 7 or 8, it is possible that an attribute inquiry might function successfully for some files that the program might open. However, the attribute inquiry might also produce an attribute error for other files.

A program that interrogates block-related attributes should interrogate the FILESTRUCTURE attribute after the file is assigned, and then the program should avoid interrogating block-related attributes if the FILESTRUCTURE value is STREAM.

Table 3–1. Constant Information Attributes

Attribute	Information Received
ALTERDATE	The date when the CLOSE operation was performed on the file following some alteration to the data in the file.
ALTERTIME	The time of day, in microseconds since midnight, when the CLOSE operation was performed on the file following some alteration to the data in the file.
AREALENGTH	The number of FRAMESIZE units in each area.
AREAS	The number of areas the physical file can have if the FLEXIBLE value is FALSE. This value can change while the file is open if the FLEXIBLE attribute value is TRUE.
AREASECTORS	The number of physical disk sectors necessary to accommodate one area of the file.
BUFFERSIZE	The number of words in memory that each buffer area occupies.
CREATIONDATE	The date when the file was created.
CREATIONTIME	The time, in microseconds since midnight, when the file was created.
CRUNCHED	TRUE indicates that the disk file was closed with the CRUNCH option. If the file was crunched to conserve space, the last allocated area for the file was truncated.
FAMILYNAME	The name of the family on which the file resides.
FILEKIND	The internal structure and purpose of the records of the file.
FILELENGTH	The length of the file in FRAMESIZE units at the time the file was opened.
LOCKEDFILE	TRUE indicates that the file cannot be removed or replaced, and that its name cannot be changed.
RESTRICTED	TRUE indicates that one of the following restrictions has been placed on the file: The unit on which the file resides was restricted by the RESTRICT (Set Restrictions) system command. The file itself was restricted by the RESTRICT system command. The file is an existing file on a remote host, and the HOSTSRESTRICTED option of the SECOPT system command is set on the local host. The file was created using Host Services logical I/O at a host with the HOSTSRESTRICTED option of the SECOPT system command set. The file was copied from a restricted unit or volume. The file was copied to a host that had SECOPT HOSTSRESTRICTED set.

Table 3–1. Constant Information Attributes

Attribute	Information Received
SECTORSIZE	The size in bytes of the physical disk sector on the disk family where the file resides.
SERIALNO	The serial number of the base member of the disk family where the file resides.
USEDATE	The date when the file was last read from or written to by a user program or, in the case of a code file, when the file was last executed.
USETIME	The time of day, in microseconds since midnight, when the file was last read from or written to by a user program or, in the case of a code file, when the file was last executed.

Table 3–2 identifies some of the attributes you can interrogate that contain values that can change while the file is open.

Table 3–2. Changing Information Attributes

Attribute	Information Received
AREAALLOCATED	TRUE indicates that the specified area has been allocated.
LASTRECORD	In most cases, the record number of the last record in the file, calculated in terms of the blocking of the logical file. In the following cases, the number of the last block is returned: FILESTRUCTURE has a value of either ALIGNED180 or BLOCKED, and the BLOCKSTRUCTURE attribute value is not FIXED. FILESTRUCTURE has a value of either ALIGNED180 or BLOCKED, and the FILETYPE attribute value is not 0 (zero).
POPULATION	The number of functions currently using the disk file header of the file.
AREAADDRESS	The physical disk address of an area of the file.
AREASINUSE	The number of areas allocated for the file.
TIMESTAMP	The header timestamp.
TOTALSECTORS	The number of physical disk sectors currently assigned to the file.

Table 3–3 identifies some of the attributes that you can interrogate after you invoke a READ or WRITE statement to help you make decisions in your program.

Table 3–3. Attributes That Contain I/O Information

Attribute	Information Received
BLOCK	If the FILESTRUCTURE value is not STREAM, the number of the block referenced in the last I/O statement.
RESULTLIST	A list of results caused by the most recent logical I/O function performed on the file.
IOINERROR	TRUE indicates that a physical I/O error has occurred on the file.

In some instances, you might want a logical file in your program to share buffers with a logical file or files in another program or programs that are accessing the same physical file. This allows a change made by one member of the set of programs sharing buffers for a given physical file to be immediately visible to other members of that set of programs.

Notes:

- *Direct files cannot share buffers.*
- *A checkpoint cannot be taken by a program that has an open file that is sharing buffers.*

The following steps enable you to accomplish this task:

- Specify a BUFFERSHARING file attribute value of SHARED or EXCLUSIVELYSHARED. By specifying either of these values, you are indicating that when this file is opened, you want the file to share a single set of buffers with other logical files that are accessing the same physical file and have BUFFERSHARING values of SHARED or EXCLUSIVELYSHARED.

  If you specify SHARED, files with a BUFFERSHARING value of NONE can open the same physical file, but those files each have unique buffers for their I/O operations.

  If you specify EXCLUSIVELYSHARED, only files with a BUFFERSHARING value of SHARED or EXCLUSIVELYSHARED can open the physical file when your program has the file opened. If a file with a BUFFERSHARING value of NONE has the file open before your program opens the file, the OPEN operation waits until that file is closed before completing the OPEN operation in your program for the file. If you do not want your program to wait to open the file, use the AVAILABLE option of the OPEN statement or interrogate the AVAILABLE file attribute. If the file can be opened, the OPEN operation proceeds. If the file cannot be opened, an OPEN result OPENWITHOUTBUFFERSHARINGRSLT (252) is returned.
- Specify a BLOCKSTRUCTURE file attribute value of FIXED. Other values are not valid when sharing buffers.
- Specify a FILEORGANIZATION file attribute value of NOTRESTRICTED. Other values are not valid when sharing buffers.

- Ensure that the BLOCKSIZE file attribute value is equal to the value stored in the disk file header of the physical file.

  Note: *This condition is automatically true if a new file is being created or if the DEPENDENTSPECS file attribute value is set to TRUE when an existing file is opened.*

- Ensure that the MAXRECSIZE file attribute value is equal to the value stored in the disk file header of the physical file if the FILESTRUCTURE file attribute is BLOCKED.

  Note: *This condition is automatically true if a new file is being created or if the DEPENDENTSPECS file attribute value is set to TRUE when an existing file is opened.*

- Set the APPEND file attribute to TRUE, if you want all WRITE operations to be unconditionally appended to the end of the file. When APPEND is TRUE, the MCP implicitly provides record locking on WRITE operations. This record locking ensures that the WRITE operations are atomic with respect to WRITE operations done to the same file by another task that has a file with the APPEND value set to TRUE, and the BUFFERSHARING value set to SHARED or EXCLUSIVELYSHARED.

When a file is buffer sharing, unwritten file space added by writing beyond the end of the file from the subject logical file is unconditionally scrubbed with zeros, as long as the PROTECTION file attribute value is other than PROTECTED. As a result, setting the CLEARAREAS file attribute to TRUE is not needed when buffer sharing, unless it is required that unwritten file space added by, or in unwritten areas touched by, logical files with a BUFFERSHARING value of NONE needs to be scrubbed with zeros.

Refer to the description of the BUFFERSHARING file attribute in the *File Attributes Reference Manual* for more information.

Locking a Disk File on a Record-by-Record Basis

If more than one logical file is accessing and updating a single physical file, there is a need to lock the file on a record basis. COBOL85 provides statement constructs to invoke disk file locking, but ALGOL and other languages do not.

To lock a file on a record-by-record basis using ALGOL or other languages, perform the following steps:

- Specify SHARED or EXCLUSIVELYSHARED as the BUFFERSHARING file attribute value.
- Write code that uses the RECORDLOCKER procedure, which is a library entry point exported from the MCP.

  For information about the RECORDLOCKER procedure, refer to the *MCP System Interfaces Programming Reference Manual*.

Securing Disk Files

This section addresses how to manage disk file security using either the POSIX or traditional security models of the MCP environment.

In both security models, only a privileged user (or process) or the owner can change the security of an existing file.

File Ownership

In most of the MCP environment file system, ownership of files is determined by the name of the file. If the name of a file contains a usercode node, the file is owned by that usercode; otherwise, it is (in effect) owned by non-usercoded ("*") processes.

The permanent directory namespace can be enabled on NX systems. This namespace consists of the files named *DIR/=. Within this namespace, files can have an owner, which is not part of the filename. You can determine the owner of a file by interrogating the OWNER file attribute.

A non-usercoded process has substantially the same rights over non-usercoded ("*") files as a process running under a usercode has over files owned by that usercode.

Securing Files Using the Traditional Security Model of the MCP Environment

In the traditional security model, file security is based on the owner of the file and on specific security attributes of the file.

Traditional Security Attributes

In the traditional security model, files are secured from nonprivileged, non-owner users by assigning values to the following file attributes:

- SECURITYTYPE
- SECURITYGUARD
- SECURITYUSE

SECURITYTYPE

To specify the way nonprivileged users can access a file, assign one of the following values to the SECURITYTYPE attribute:

Mnemonic	Meaning
PRIVATE	Only the owner can access the file.
PUBLIC	All nonowner users can read, write, or execute the file, or do all three, based on the value of the SECURITYUSE file attribute.
GUARDED	Access by specific nonowner users is controlled by a guard file.
CONTROLLED	Access by all users, including the owner, is controlled by a guard file. (This value is invalid for Host Services logical I/O.)

SECURITYUSE

To specify how a file protected with a SECURITYTYPE value of PUBLIC can be accessed by nonprivileged users, assign the SECURITYUSE attribute one of the following values:

Mnemonic	Meaning
IN	Users and programs have read-only access to the source file, data file, or code file. Code files can also be executed.
IO	Users and programs have read and write access to the source file, data file, and code file. Code files can also be executed.
OUT	Users and programs have write-only access to the source file, data file, and code file. Codes file can also be executed.
SECURED	The code file can be executed only.

SECURITYGUARD

An alternative to granting read, write, or execute rights to all nonowner users is to grant or deny access to individual users or programs. The SECURITYGUARD attribute specifies the guard file to be used to limit access. If you set SECURITYTYPE to CONTROLLED or GUARDED, you must specify a guard file.

For more information about the use of guard files, see the *Security Features Guide*.

Additional Information about Traditional File Security

The traditional security model is a subset of the POSIX security model. See the description of the SECURITYMODE attribute in the *File Attributes Reference Manual* for more information.

For more information about the interaction of secured files and processes, see the *Security Administration Guide*. For more information about a specific security attribute, see the description of the attribute in the *File Attributes Reference Manual*.

Securing Files Using the POSIX Security Model

The POSIX security model differs significantly from the traditional security model of the MCP environment. The MCP environment provides security capabilities which are based on the POSIX standards, but which extend it in various ways.

The traditional security model based file security on the file name and the SECURITYTYPE, SECURITYUSE, and SECURITYGUARD attributes. POSIX file security is based on the owner of the file and a series of file permission bits that are permanently associated with the file.

The features of the POSIX security model apply to all files on the system; traditional file security is implemented as a subset of the POSIX security model.

Note: *Prior to the 44.2 release, any attempt to set the GROUP or SECURITYMODE attributes while the ENABLEPOSIX system option is RESET resulted in a READONLY attribute value. It is no longer necessary to set or reset the ENABLEPOSIX option to control any functionality. The ENABLEPOSIX option will be deimplemented in a future release.*

A file can be associated with a group of users identified by a groupcode, as groups in POSIX can also be granted or denied privileges to files. Thus, the ability of the user to access a file depends not only on his or her own privileges, but the privileges granted to any security groups to which the user belongs.

Determining Access to a File

When a file is opened, access to the file is determined by the owner, group, or other security attribute:

- When the file is opened, the I/O subsystem first compares the value of the USERCODE attribute of the process opening the file with the usercode of the owner of the file. If they match (or if the file is a "*" file and the process is non-usercoded) the OWNERRWX security subattribute is used.
- If the usercodes do not match, the GROUP and ALTERNATEGROUPS file attributes of the file (if any) are compared to the GROUPCODE task attribute of the process and to the list of groupcodes in the SUPPLEMENTARYGROUPS task attribute of the process. If any of the groupcodes of the file matches any of the groupcodes of the task, the GROUPRWX and ALTERNATEGROUPS permissions for matching groupcodes are merged to determine the access permissions.
- If neither the usercodes nor the groupcodes match, the value of the OTHERRX subattribute is used.

File Permission Bits

The file permission bits in a file are stored in the SECURITYMODE attribute of the file. The attribute is broken into subattributes so values can be assigned to the subattribute, or so individual bits can be set directly.

The following table lists the name and description of the owner, group, and other subattributes:

Name	Description
OWNERRWX	The read, write and execute permissions that are granted to the owner of the file.
GROUPRWX	The read, write, and execute permissions that are granted to the group associated with the file.
OTHERRWX	The read, write and execute permissions that are granted to all other users of the system.

Four additional bits are defined for implementing features analogous to the traditional security attributes SECURITYUSE and SECURITYGUARD:

Name	Description
GUARDOWNER	If TRUE when USEGUARDFILE is TRUE, the guard file also applies to the owner of the file.
USEGUARDFILE	If TRUE, the guard file also defines access privileges to the file.
SETUSERCODE	If TRUE for a code file, the code file executes with a USERCODE of the owner of the file.
SETGROUPCODE	If TRUE for a code file, the code file executes with a GROUPCODE of the group of the file.

Note: *The SETUSERCODE and SETGROUPCODE flags can cause programs to behave unexpectedly since they execute under a USERCODE and/or GROUPCODE that might be different from that of the initiator.*

The subattributes listed in the preceding two tables, along with additional bit masking subattributes, are explained fully in the description of the SECURITYMODE file attribute in the *File Attributes Reference Manual*.

The ALTERNATEGROUPS file attribute can be used to extend the capabilities of the SECURITYMODE attribute by associating multiple groups with the file.

> **Caution**
>
> Mixing the use of the SECURITYTYPE and SECURITYUSE attributes with the SECURITYMODE or ALTERNATEGROUPS attributes can produce unexpected results and should be avoided.

Setting the SECURITYTYPE or SECURITYUSE attributes might cause the value of SECURITYMODE to change.

Certain values of SECURITYMODE also do not map exactly back to SECURITYTYPE. For example, the SECURITYMODE value of a file might have GROUPR and GROUPW set (granting read and write permissions to the group), but no permissions set in OTHERRWX. Interrogating SECURITYTYPE for this file would return a value of PUBLIC even though the file is not accessible to all users of the system. In addition, the value of SECURITYUSE that is returned would be IO even though execute permission is not granted.

For detailed information about the interaction of the POSIX and traditional security models, see the description of SECURITYMODE in the *File Attributes Reference Manual*. For more information about the interaction of secured files and processes, see the *Security Administration Guide*.

When to Use the POSIX Security Model

Use the capabilities of the POSIX security model for specific files that need to be accessible by a specific group of users or otherwise controlled using the specific capabilities of the model.

If you have programs that propagate the security of other files, consider using the POSIX security model so that security is propagated accurately.

File Security Propagation

In order to facilitate conversion of programs that propagate security from one file to another, an MCPSUPPORT interface is provided that returns security information about a specified file, including an indication as to whether the SECURITYMODE attribute or the SECURITYTYPE and SECURITYUSE attributes should be used to propagate security.

Disk Files

The FILE_SECURITY procedure is declared as follows:

```
LIBRARY MCP (LIBACCESS=BYFUNCTION, FUNCTIONNAME="MCPSUPPORT");

INTEGER PROCEDURE FILE_SECURITY_INFO (F, MODE, TYPE, USE, GUARDED);
        FILE    F;
        INTEGER MODE,
                TYPE,
                USE,
                GUARDED;
        LIBRARY MCP;
```

The parameter F is a file that must be assigned to a disk file.

The procedure returns the following information:

Information	Meaning
Procedure Result	Indicates SECURITYMODE and related attributes are fully supported by the MCP and can be set. Releases HMP 3.0 and SSR 44.2 and later always return this result. Indicates SECURITYMODE and related attributes are READONLY A negative value indicates an error condition.
MODE	The F.SECURITYMODE value (returned even if the MCP does not support the SECURITYMODE file attribute)
TYPE	The F.SECURITYTYPE value
USE	The F.SECURITYUSE value
GUARDED	Indicates that the SECURITYGUARD attribute is set Indicates that the SECURITYGUARD attribute is not set

When the procedure returns a value of 1, security should be propagated by setting the SECURITYMODE attribute of the file to which security is being propagated to the value returned in MODE; otherwise SECURITYTYPE and SECURITYUSE should be set.

In some cases where SECURITYMODE is to be propagated, it might also be necessary to propagate the value of the GROUP attribute.

Direct I/O Files

The FILE_SECURITY_INFO_DIRECT procedure is provided for Direct I/O files. The procedure is identical to FILE_SECURITY_INFO except that it takes a DIRECT FILE rather than a disk file as a parameter.

Refer to the *MCP System Interfaces Programming Reference Manual* for more information on these procedures.

Securing Allocated and Deallocated Disk Space

The following paragraphs describe the security of allocated and deallocated disk space:

Initializing Allocated Disk Space

If the disk area of a file is to be set to all zeros when the area is allocated to the file, set the CLEARAREAS file attribute to TRUE.

If you are using the Security Accountability Facility, instead of setting the CLEARAREAS attribute to TRUE, you can set the DISKSCRUB option of the SECOPT system command to ensure that any data remaining in a disk area is "scrubbed" (removed) before the area is reused.

Returning Deallocated Disk Space

If the current data in a disk space is to be removed when disk space is returned to the system, set the SENSITIVEDATA file attribute to TRUE. Doing so causes the file to be entered into the directory when the file is opened as if the PROTECTION attribute were set to SAVE or PROTECTED, and the disk space to be overwritten with an arbitrary pattern before the space is returned to the system for reallocation.

Files with a KIND Value of CD

Files that reside on a CD-ROM device have a KIND value of CD. The operating system identifies the disk on a CD-ROM device by using the Volume Identifier of the disk that was assigned by the creator of the disk.

All files that reside on a CD-ROM disk must be identified with a FILESTRUCTURE file attribute value of STREAM.

If you are not familiar with basic programming methods, review those methods in Section 2, "Understanding Programming for Files," before reading the following procedures. You can identify all the file attributes that can be used with a disk file by reviewing Table A–1. For more information about any of the file attributes, refer to the *File Attributes Reference Manual*.

Accessing a CD-ROM File

Perform all or some of the following tasks, depending on the needs of your program:

- If the physical file name is different from the internal file name of the file, specify the physical file name by using the FILENAME attribute.
- Specify where the file resides by using the FAMILYNAME attribute. If the file is on a CD-ROM, use the Volume Identifier.
- Assign the KIND attribute a value of CD.
- Assign the NEWFILE attribute a value of FALSE.
- Because CD-ROM files always have an EXTMODE value of OCTETSTRING, which requires the same value of INTMODE, assign the INTMODE attribute a value of OCTETSTRING or assign the DEPENDENTINTMODE attribute a value of TRUE. The default value is FALSE.
- Set the DEPENDENTSPECS attribute value to TRUE or assign the FILESTRUCTURE attribute a value of STREAM.

  Attempting to open a file with a FILESTRUCTURE value of STREAM with inconsistent MAXRECSIZE values results in an OPEN error.
- Since most CD-ROM files have a MAXRECSIZE value of 1, set the ANYSIZEIO attribute value to TRUE.

Obtaining Information about a CD-ROM File

When your file is open, you can interrogate several attributes to obtain information about the file. Tables 3–4 and 3–5 identify these attributes.

Table 3–4 identifies some attributes that might be of interest to you that have information while the file is open. Most attributes can be interrogated.

Table 3–4. Information Attributes for CD-ROM

Attribute	Information Received
BUFFERSIZE	The number of words in memory that each buffer area occupies.
CREATIONDATE	The date when the file was created.
CREATIONTIME	The time, in microseconds since midnight, when the file was created.
FAMILYNAME	The name of the family on which the file resides.
FILELENGTH	The length of the file in number of bytes.
LASTRECORD	The record number of the last record in the file, calculated in terms of the blocking of the logical file.

Table 3–5 identifies some of the attributes that you can interrogate after you invoke a READ statement to help you make decisions in your program.

Table 3–5. CD-ROM Attributes That Contain I/O Information

Attribute	Information Received
RESULTLIST	A list of results caused by the most recent logical I/O function performed on the file.
IOINERROR	TRUE indicates that a physical I/O error has occurred on the file.

I/O Timer Handling

The I/O timer handling feature enables you to improve control over the timing of disk I/O operations. By making use of the interfaces for this feature, you can improve the I/O response time of your applications when, for example, a disk malfunctions or excessive I/O traffic induces unusually long queuing delays within a subsystem.

You might need to use I/O timer handling because of the characteristics of the high-speed storage devices used on the system and the need for quick response time in a time-critical, transaction-based environment. This feature is available only on channel-based IOM systems, such as the ClearPath NX4600 and NX4800 systems and the A 11, A 14, A 16, A 18, and A 19 systems.

The I/O timer handling feature provides your application with two interfaces to improve control over the timing of disk READ and WRITE operations. The first interface specifies the I/O time limit that is used for disk READ and WRITE operations. The second interface directs the MCP to return direct disk I/Os to their requesters as soon as possible. The combination of these interfaces enables your application to have more control over the maximum amount of time an I/O request takes. By using these interfaces, you can improve the application response time for I/O operations that take an unusually long time to complete.

This section contains information on

- How I/O requests are processed
- Why the elapsed time in processing an I/O request varies and can exceed the value set with the IOTIMER system command
- How to determine the length of time an application should wait for an I/O request

A variety of system software is involved in I/O timer handling, including

- IOCANCEL, IOMASK, and IORESULT attributes
- IOTIMER system command, which sets or queries time limit values
- SETSTATUS interface to set time limit values
- GETSTATUS and SYSTEMSTATUS interfaces, which query time limit values
- "OL" and "PER PK" system commands displays, which provide time limit information
- "MOVE PK" and "MIRROR CREATE" system commands, which copy the time limit from the source pack to the destination pack
- Logging, which provide the time limit in effect for a unit
- Status Change Message, which informs applications of a time limit change
- PTD I/O time limit, which does not exceed a user-specified value

Other aspects of I/O timer handling are explained in the manuals listed in the following table:

For information on	Refer to the
File attributes	*File Attributes Reference Manual*
I/O TIMER and other system commands	*System Commands Reference Manual*
GETSTATUS and SETSTATUS I/O timer interfaces	*GETSTATUS/SETSTATUS Reference Manual*
SYSTEMSTATUS requests	*SYSTEMSTATUS Programming Reference Manual*
System log and LOGANALYZER reports	*System Log Programming Reference Manual* and the *System Software Utilities Manual*
Status change message in the STATUS_CHANGE_REPORT entry point	*MCP System Interfaces Programming Reference Manual*

Understanding Time Limit Values

The following background information explains how the system uses the time limit value. The explanation includes references to various components of the system. If you are interested in more information on these system components, refer to the appropriate Capabilities Overview manual.

The I/O time limit value controls the amount of time that passes before the host decides that an I/O operation will not complete. The I/O time limit value is specified in the input/output control block (IOCB) that is associated with the operation. The I/O time limit value determines how long the input/output unit (IOU) will time an I/O operation from when it is removed from the IOU unit queue until the result is received from the channel.

The following steps provide a conceptual overview of the logic used when the IOM system processes an I/O request:

1. An application issues a disk Read or Write operation. Then it waits for the completion of an event associated with the direct I/O buffer.
2. The MCP builds an IOCB for the operation and passes the IOCB to the IOU for normal execution. The IOU places the IOCB at the tail of the FIFO unit queue that is maintained for the device. From this queue, IOCBs are executed one at a time for the target device.
3. Once the IOCB reaches the head of its queue, the IOU promotes it to active status and gives it to the appropriate channel manager unit (CMU) for outboard execution.
4. The CMU passes the active I/O request to the channel.
5. The channel passes the I/O request to the disk subsystem.
6. The disk subsystem processes the I/O request and informs the channel when processing finishes.

7. The channel informs the IOU when I/O processing finishes.
8. If the I/O request completes successfully without exception (including various attention exceptions for MCP special action), the IOU

   a. Promotes the next IOCB in the unit queue to active status

   b. Applies a hardware finish action to the successful IOCB

   Then processing continues.

9. If the I/O request completes as an exception, the IOU suspends processing of the unit queue pending software resolution and returns the IOCB to the MCP for appropriate exception handling. Exception handling can include retries of I/O requests or special operations to condition the subsystem.
10. If an I/O is to be retried, the IOU inserts the IOCB at the head of the queue before the device is restarted or before a special forced execution of the IOCB is requested. In either case, steps 3 through 8 are repeated until a resolution is reached. Examples of resolution are

    - Successful I/O completion
    - The occurrence of an irrecoverable exception
    - Exceeding the retry limit for the I/O request

    When a resolution is reached, the device is restarted and the IOCB is returned to the IOU for hardware finish. Note that an irrecoverable error can result.
11. So that the application can detect that the operation completed, the task control unit (TCU) causes the I/O completion event associated with the IOCB as part of hardware finish action.
12. The application resumes when the completion event associated with the direct I/O buffer occurs. The application program then looks at the logical result associated with the I/O to determine if the I/O operation was successful.
13. The MCP sets a time limit value in the IOCB before passing the IOCB to the IOU. The IOU uses this value to control the length of time the IOCB can be outstanding to the I/O subsystem. The IOU begins timing the IOCB when the IOCB is passed to the CMU. If a result is not returned to the IOU within the specified time limit, the IOU forces the IOCB to complete, then returns the IOCB to the MCP with a Timelimit Exceeded exception result.

In the I/O flow described in the previous list, the time limit specified in the IOCB controls the time permitted to complete steps 3 through 7. The time limit does not include the MCP initiation time or the IOU unit queue time. Each time the IOU passes an IOCB to the CMU, it applies the time limit specified in the IOCB. This is the case whether the IOCB time limit is applied to the original I/O request, an MCP retry of the I/O request, or an MCP special operation to condition the subsystem.

The time limit specified in the IOCB does not correspond to I/O elapsed time measured by your application. In the I/O flow described previously, elapsed time refers to the time needed to complete all 11 steps. Elapsed time for an I/O request begins when the application issues the I/O request and ends when the application is notified that the I/O has completed.

Elapsed time includes

- The time needed for the original I/O
- The sum of all retries
- The sum of all processing overhead. Overhead might include initiation delays or delays because of MCP handling of specific errors that required operator intervention. Examples of errors that might include operator intervention include not ready, write lockout, or no path errors.

The time limit value that the MCP stores in an IOCB for disk READ and WRITE operations applies to all read and write operations of the affected units, such as user I/O requests and MCP I/O requests. The value specified is a global timer for the unit. No interface exists that enables you to specify a time limit on an I/O request by I/O request basis. This limit is used for each retry performed by the MCP when errors occur.

Default Time Limit Value

By default, the MCP specifies an 80-second time limit for all disk read and write operations. If you do not use I/O timer handling, you cannot change this value. If you use I/O timer handling, you can specify one of the following values:

5 seconds
10 seconds
20 seconds
40 seconds
80 seconds

If you select a time limit less than 80 seconds, you override the system default. You should choose a lower limit only if the default value is unacceptably long.

The system assigns an 80-second time limit because 80 seconds was determined to be the best available value that ensures that the I/O subsystem has done all it can to issue the I/O request. If the 80-second time limit is exceeded, a part of the I/O path between the IOU and the subsystem is experiencing a hardware problem. If you reduce the time limit too much, the I/O request might have completed successfully if you had allowed more time or you might prevent the system from detecting and handling the problem.

Effective use of the interface can shorten the elapsed time that an I/O, which will eventually time out, takes to complete because the amount of time the IOU allows the subsystem to process the operation is shorter. Depending on the configuration of the I/O subsystem, your site might be able to determine with certainty that a hardware problem exists if an I/O exceeds the shortened time limit. Therefore, shortening the time limit can recognize a hardware problem early, thus reducing the maximum length of time the system takes to handle the condition.

Range of Time Limit Value

The IOU uses the time limit value in the IOCB to determine the length of time an active I/O can be outstanding to the CMU. The IOU uses an internal clock, which checks for IOCB timeouts every 1.2 seconds. If the time limit specified in one of the active IOCBs will expire during the current 1.2 second interval, the IOU takes the necessary action to time out the IOCB.

Because the IOU checks for timeouts every 1.2 seconds, the time limit value in the IOCB represents a 1.2-second range, not an absolute value. The amount of time the IOU gives an individual IOCB depends on when during the 1.2 second interval the IOCB becomes active. If the IOCB becomes active at the start of a new time-out checking interval, the IOCB is given almost 1.2 seconds longer than an IOCB that becomes active just before the end of the same interval.

The following table shows the range of time-out times of an IOCB based on the time limit specified in the IOCB:

IOCB Time Limit	Minimum IOU Timeout Range	Maximum IOU Timeout Range
5	4.8	6.0
10	9.6	10.8
20	19.2	20.4
40	39.6	40.8
80	79.2	80.4

For example, if your site changes the disk read and write timeout to 10 seconds, the IOU allows the active IOCB to be outstanding to the CMU from 9.6 to 10.8 seconds before initiating a Timelimit Exceeded action.

Limited Effect of Time Limit Value

Changing the time limit in the IOCB does not ensure that an I/O completes within a predetermined length of time. The following list includes some of the factors that affect the amount of time an I/O request takes, independent of the I/O time limit specified in an IOCB:

- MCP initiation time
- IOU unit queue time
- Cumulative effect of time elapsed for MCP retries
- Delays due to MCP handling of errors that indicate that operator intervention is required (for example, not ready, write lockout, no path errors)
- Delays caused by mirrored writes because of faulty set members
- System overhead, which occurs during the processing of a Timelimit Exceeded exception

Returning an I/O Request As Soon As Possible

The I/O timer handling feature provides a direct I/O interface to instruct the MCP to return disk I/O requests to their requesters as soon as possible. The return occurs even when an I/O request would be successful if it was allowed to complete normally. Because architectural factors can delay the return of the I/O request, the direct I/O interface does not always return requests immediately.

The direct I/O programmatic interfaces include changes to the IOCANCEL, IOMASK, and IORESULT attributes, which are described next.

Direct I/O Buffer Program Interface

The following direct I/O buffer attributes instruct the MCP to complete disk I/O requests as soon as possible:

IOCANCEL Attribute

This attribute is enabled for disk files. If the IOCANCEL attribute is set to TRUE following an I/O request to a disk unit and the I/O request is in progress, the MCP is instructed to return the I/O request with a User Cancel Result (see IORESULT). Other I/O requests can also be returned to their initiators as a result of setting IOCANCEL attribute to TRUE. See the following description of the IOMASK attribute for details on the return of other I/O requests.

IOMASK Attribute

Mask bit 13 is defined for use by disk files. If a direct I/O application sets this bit and another direct I/O buffer uses the IOCANCEL attribute, the MCP might return the I/O request as user canceled. An I/O request that meets all the conditions required for returning the I/O request is called a qualifying user-cancel-masked I/O request. The following conditions must apply for the MCP to return the masked I/O request as user canceled.

- The direct I/O buffer for the IOCANCEL target I/O request and the direct I/O buffer for the masked I/O request must be declared by the same stack.
- Both I/O requests must be directed to the same disk unit.
- The masked I/O request must not be completed previously.

When an I/O request is user canceled, its requester receives a User Cancel Result. See the following description of the IORESULT attribute for an explanation of User Cancel Result.

IORESULT Attribute

A User Cancel logical result bit (13) is defined for disk files. If an I/O request is returned because a direct I/O buffer set its IOCANCEL attribute, the IORESULT attribute has the User Cancel (bit 13), Cancel (bit 2), and Exception (bit 0) bits set. This result, referred to as the User Cancel Result, applies to the IOCANCEL target-I/O request and any qualifying user-cancel-masked I/O request.

Effects of the Direct I/O Attribute Program Interface

The following text describes the effects of the IOCANCEL, IOMASK, and IORESULT direct I/O buffer attributes when the IOCANCEL attribute is set to TRUE for a disk file buffer.

Nonmirrored Units

When the IOCANCEL attribute is set to TRUE for a direct I/O request issued to a nonmirrored unit, the following process occurs:

- The unit queue affected by the I/O request is suspended and recalled.
- The recalled queue is searched for qualifying user-cancel-masked I/O requests. By definition, the IOCANCEL target-I/O request always qualifies.
- Each qualifying I/O request is returned and its IORESULT attribute is set to the User Cancel Result.
- The unit queue is restarted.

Mirrored Units

When the IOCANCEL attribute is set to TRUE for a direct I/O request issued to a mirrored unit, the following process occurs:

- For each member of the mirrored set, the unit queue is suspended and recalled.
- Each recalled queue is searched for qualifying user-cancel-masked I/O requests. By definition, the IOCANCEL target-I/O request always qualifies.

  If an online set member has no qualifying IOCB queued, online members with a qualifying IOCB queued are decommitted to audit.

  If each of the online members has a qualifying IOCB queued, only the copy determined to be best remains online; the remaining copies are decommitted to audit. A copy is determined to be best if its topmost, qualifying IOCB has been queued for the shortest amount of time.

  Among the qualifying user operations, a mirrored WRITE operation is successfully returned if completion occurs without exception to any set member remaining online. Incomplete qualifying mirror READ operations, and qualifying mirror WRITE operations that do not complete successfully to any member left online, are returned with a User Cancel Result.

  To restore set integrity as soon as possible, the MCP immediately begins to apply audited write operations to the decommitted members. The MCP uses a visible MIRROR_CREATE process for each member being restored.

- For each member of the mirrored set, the unit queue is restarted.

  Once processing of the attribute completes, I/O activity resumes on the affected logical unit, with the exception of any mirror that was placed in the offline mirror state. I/O activity to these mirrors is automatically restarted after the audit is applied by the MCP.

Factors Affecting "As Soon As Possible"

The following text discusses the factors that determine the length of time the system needs to process the IOCANCEL request. As previously noted, the IOCANCEL interface does not always return the requested I/O requests immediately; it returns it "as soon as possible."

The MCP must have control over an I/O request before it can return the I/O request with a User Cancel Result. When the IOCANCEL attribute for an active I/O request is set, either the MCP or the IOU has control over the I/O request. The MCP does not have control over the I/O request from the time that the MCP passes the IOCB to the IOU until the IOU returns the IOCB. If the IOU has control over the I/O request, the MCP issues these instructions to regain control of the I/O request from the IOU:

1. The MCP instructs the IOU to stop processing I/O requests to the target device when the active I/O request completes.
2. The MCP instructs the IOU to return all the I/O requests for the device that are currently on the unit queue.

The active I/O request from Step 1 is always given time to complete. Upon completion, the I/O request has either completed successfully or by exception. The amount of time given to the I/O request is determined by the time limit contained in the IOCB. Since the IOU starts timing when the I/O request is passed to the CMU, the timer either could expire immediately (if the I/O request is almost ready to time-out when Step 1 is processed) or take the entire time limit specified in the IOCB (if the I/O request is passed to the CMU just before Step 1 is processed).

If the IOU times out the active I/O request, a maximum of 2.4 additional seconds might be needed for the hardware to cleanly return the IOCB to the MCP with a Timelimit Exceeded exception (IORESULT value of Timelimit Exceeded (bit 15) and Exception (bit 1)). Additionally, the MCP requires 4 to 12 seconds to recover from the time-out condition. Therefore, if the default 80-second time limit is specified and the IOCANCEL attribute is set to TRUE, the system might need a maximum of 94.8 seconds to return the I/O request.

You can shorten the length of time it takes the system to return a qualifying IOCANCEL I/O request by reducing the time limit applied to the operation. You cannot affect the time required by the IOU (up to 2.4 seconds) or the MCP (up to 12 seconds). Therefore, under normal operating conditions, the maximum amount of time the system needs to return the IOCANCEL target I/O request is equal to the sum of the time limit specified in the IOCB for the active I/O request plus 14.4 seconds.

The following table shows the relationship between the IOCB time limit value and the maximum wait time for a qualifying IOCANCEL I/O request.

IOCB Time Limit	Maximum IOU Timer	Possible IOU Overhead	Possible MCP Overhead	Maximum Wait Time
5	6	2.4	12	20.4
10	10.8	2.4	12	25.2
20	20.4	2.4	12	34.8
40	40.8	2.4	12	55.2
80	80.4	2.4	12	94.8

The Maximum Wait Time column represents the longest elapsed time an IOCANCEL assignment statement can take to synchronously complete and return control to an application. The value assumes a worst case for each of the timing windows applied to the active I/O request. The actual time that an application waits could be much shorter.

Programming Considerations

The I/O timer handling interface enables a direct I/O application to user-cancel a disk READ or WRITE operation. The decision to user-cancel the operation often occurs after a specified period of time elapses. After the I/O request is initiated, a WAIT statement can specify this time period and an event associated with the I/O request.

Your application must specify the length of time to wait before user-canceling the I/O request. Several factors discussed earlier affect the amount of time an operation needs. You must consider these four factors in deciding how long the application is to wait.

1. The maximum elapsed time that your application can tolerate.
2. The time limit contained in the IOCB of the active I/O request.
3. The IOU requirement of 0 to 2.4 seconds, and the MCP requirement of 4 to 12 seconds to recover from a time-out condition.
4. Application wait time before deciding to user-cancel the I/O request.

The relationship of these four timer values is outlined by the following equation:

```
<maximum I/O elapsed time> - <IOCB time limit> - 14.4 = <wait time>;
        (1)                        (2)          (3)       (4)
```

Items 1, 2, and 4 are site specific. Item 3 is a constant value. You must set the site-specific timer values to suit your needs. Consider the following examples:

Example 1

If your site requires that an I/O request is outstanding for 3 minutes or less (180 seconds), you can use the default IOCB time limit value and wait 85 seconds before user-canceling the I/O request:

```
<maximum I/O elapsed time> - <IOCB time limit> - 14.4 = <wait time>
         180               -       80.4        - 14.4 = 85.2
```

Example 2

If your site requires that an I/O request is outstanding for 30 seconds or less, you must reduce the IOCB time limit to either 5 or 10 seconds, which results in a program wait time of no more than 9.6 and 4.8 seconds, respectively:

```
<maximum I/O elapsed time> - <IOCB time limit> - 14.4 = <wait time>
          30               -       6           - 14.4 =  9.6
          30               -      10.8         - 14.4 =  4.8
```

Notice that the greater the value permitted for maximum I/O elapsed time the more flexible wait time and IOCB time limit can be (as in example 1). Whereas, the smaller the value, the less flexible the other timers become (as in example 2).

If the WAIT statement completes because the time expires, the application can make use of the IOCANCEL attribute to user cancel the I/O request. The actions taken as a result of setting the IOCANCEL attribute are outlined in "Effects of the Direct I/O Attribute Program Interface" earlier in this section.

The following sample pseudocode ensures that an I/O operation is returned within 30 seconds when the IOCB time limit is 10 seconds:

```
WRITE (<file>,<length>,<buffer>) [<event>];
RSLT := WAIT ((4.8),<event>);
IF RSLT EQL 1 THEN
     BEGIN
     <buffer>.IOCANCEL := TRUE;
     WAIT (<event>);
     IF BOOLEAN(<buffer>.IORESULT) THEN
          IF <buffer>.IORESULT.[13:1] THEN
               % I/O has been user-canceled.
          ELSE
               % Some other exception occurred.
     ELSE
          % The I/O completed successfully before
          % being user-canceled by the system.
     END;
```

The pseudocode in the previous example works well if your application serially processes one I/O request at a time. It cannot be used, however, to ensure that more than one I/O request is returned within the same time period. Consider the following:

```
WRITE (<file>,<length>,<buffer_1>) [<event_1>];
WRITE (<file>,<length>,<buffer_2>) [<event_2>];
RSLT := WAIT ((4.8),<event_1>,<event_2>);
IF RSLT EQL 1 THEN
     BEGIN
     <buffer_1>.IOCANCEL := TRUE;
     WAIT (<buffer_1>);
     IF BOOLEAN(<buffer_1>.IORESULT) THEN
          IF <buffer_1>.IORESULT.[13:1] THEN
               % I/O has been user-canceled.
          ELSE
               % Some other exception occurred.
     ELSE
          % The I/O completed successfully before
          % being user-canceled by the system.
     <buffer_2>.IOCANCEL := TRUE;
     WAIT (<buffer_2>);
     IF BOOLEAN(<buffer_2>.IORESULT) THEN
          IF <buffer_2>.IORESULT.[13:1] THEN
               % I/O has been user-canceled.
          ELSE
               % Some other exception occurred.

ELSE
          % The I/O completed successfully before
          % being user-canceled by the system.
END;
```

In the previous program example, neither buffer has the User Cancel IOMASK bit set, so each IOCANCEL assignment statement affects only its own buffer. Since an IOCANCEL assignment statement is handled synchronously by the system, the maximum elapsed time of the I/O for buffer_2 can exceed 30 seconds. The maximum possible elapsed time for buffer_2 is calculated as follows:

```
<maximum elapsed time> = <wait time> + <buffer_1 IOCB time limit> + 14.4
                                     + <buffer_2 IOCB time limit> + 14.4;
```

If you use the values from the example in the preceding text, the maximum elapsed time is calculated as follows:

```
50.4 = 9.6 + 6 + 14.4 + 6 + 14.4;         % <IOCB time limit> of 5 seconds
55.2 = 4.8 + 10.8 + 14.4 + 10.8 + 14.4;   % <IOCB time limit> of 10 seconds
```

The maximum elapsed time for buffer_2 exceeds 30 seconds because of the actions taken for each IOCANCEL assignment statement. Each IOCANCEL assignment statement results in the MCP recalling all I/O requests for this device from the IOU, user-canceling the IOCANCEL target I/O request from the list of recalled I/O requests, and reissuing the remaining I/O requests to the IOU. For each IOCANCEL statement processed, the IOU does not return the outstanding I/O requests until after the active I/O request completes.

If your application issues more than one I/O request and requires the I/O requests to be completed in a specified length of time, the I/O requests must be issued with the User Cancel IOMASK bit set (13). Setting the User Cancel IOMASK bit causes any qualifying user cancel masked I/O request to be returned with the IOCANCEL target I/O request.

```
<buffer_1>.IOMASK := * & 1 [13:1];
<buffer_2>.IOMASK := * & 1 [13:1];
WRITE (<file>,<length>,<buffer_1>) [<event_1>];
WRITE (<file>,<length>,<buffer_2>) [<event_2>];
RSLT := WAIT ((4.8),<event_1>,<event_2>);
IF RSLT EQL 1 THEN
     BEGIN
     <buffer_2>.IOCANCEL := TRUE;
     WAIT (<buffer_1>);
     IF BOOLEAN(<buffer_1>.IORESULT) THEN
          IF <buffer_1>.IORESULT.[13:1] THEN
               % I/O request has been user-canceled.
          ELSE
               % Some other exception occurred.
     ELSE
          % The I/O completed successfully before being
          % user-canceled by the system.

     WAIT (<event_2>)
     IF BOOLEAN(<buffer_2>.IORESULT) THEN
          IF <buffer_2>.IORESULT.[13:1] THEN
               % I/O request has been user-canceled.
          ELSE
               % Some other exception occurred.
     ELSE
          % The I/O completed successfully before being
          % user-canceled by the system.
     END;
```

Because the User Cancel mask bit is set for both I/O requests before they are issued, the IOCANCEL assignment statement can be made to either buffer. After the IOCANCEL statement completes, the completion events associated with both I/O requests occur, enabling both I/O requests to be processed within the 30 seconds required by the application.

Your application must wait on each completion event following the IOCANCEL assignment statement. In general, this WAIT statement completes immediately, but you must take precautions in case the event has not yet occurred. Completion is delayed only if some external portion of the system is not functioning properly. For example, an asynchronous finish action to log a User Cancel, or other exception result, is sometimes necessary when an IOCANCEL assignment is made. If the finish action is delayed because the MCP is unable to write to the LOG because of LOG unit errors, then causing the I/O completion event is also delayed.

Mirrored Disk Considerations

In general, all members of a mirrored set use the same time limit. If you change the time limit of a unit that is also a member of a mirrored set, the change applies to all the other members of the set.

MIRROR CREATE System Command

The disk added into the mirrored set is given the same time limit value as the current set members. The time limit of the set applies to the new member as it is being created. If an error prevents the disk from becoming an online member of the set, the original time limit value of the disk is restored.

MIRROR RELEASE System Command

A MIRROR RELEASE system command has no effect on the time limit assigned to the remaining or released members of a mirrored set.

Logging Considerations

The system logs every request for a disk time-limit change. These requests appear in the SUMLOG. The following examples show the format of log entries:

```
08:44:29      ->5993      OPERATOR ENTERED: IOTIMER PK 241-243 VALUE 10
08:44:30      ->5994      OPERATOR ENTERED: IOTIMER PK ALL VALUE DEFAULT
08:44:31      ->5995      OPERATOR ENTERED: IOTIMER PK SUBTYPE 50 VALUE 10
```

The Hardware Configuration report displays the read and write time-out value for each disk contained in the report. The time-out values appear in the DISK R/W TIME-OUT column. If the time-out value in effect is the default value, a lowercase d appears after the time-out value. The following example show the format of the Hardware Configuration report.

```
"d" AFTER DISK READ/WRITE TIME-OUT VALUE INDICATES MCP DEFAULT

UNIT   TYPE (SUBTYPE & DENSITY)                    TIME-OUT
----   -----------------------                    --------
100    419-1 SCSI PACK                            10
200    805-1 SCSI PACK                            80    d
```

The analysis provided for a Timelimit Exceeded exception includes the time limit used by the IOU to time the I/O request. The analysis appears in maintenance log reports and in I/O summary reports. The format of the report is as follows:

```
IO TIMED OUT (<value>)
```

where the value either can be the specified number of seconds (for example, 20 SECONDS) or the word UNTIMED, which indicates the I/O operation was issued as an untimed I/O request.

In the following IOSUMMARY report example, pack 800 had two I/O requests that timed out while the time limit was 10 seconds. Then it had one I/O request that timed out while the time limit was 80 seconds.

```
ERROR   UNIT                    RESULT                              . . .
COUNT   TYPE   UNIT #   R/W     ANALYSIS                            . . .
-------------------------------------------------------------------------
    2   PACK   800         W    IO TIMED OUT (10 SECONDS)           . . .
    1   PACK   800         W    IO TIMED OUT (80 SECONDS)           . . .
```

System Interface Considerations

A program can monitor when a change in a time limit for a disk occurs by making use of the STATUS_CHANGE_REQUEST procedure in the MCPSUPPORT library. Message number 34 applies to time-out information. The change occurs for any of the following conditions:

- The IOTIMER system command or the corresponding SETSTATUS call changes the time-limit value
- A disk is added to a mirrored set and the time limit in effect for the mirrored set differs from that of the disk being added
- The disk is the destination pack of a MOVE PK command

Peripheral Test Driver (PTD) Considerations

The PACKSCAN PTD can be executed against an online pack. The I/O operations issued by the PACKSCAN PTD are interleaved with the read and write operations issued by other applications. In general, the time limit assigned to PACKSCAN I/O operations are greater than the MCP default value for disk read and write operations.

The MCP checks whether a user-specified time limit is in effect for a disk before assigning a time limit for a PTD I/O operation. The MCP checks only the PTD I/O operations that do not require exclusive use of the disk (for example, the disk does not have to be reserved.) If a user-specified time limit is in effect, the lower value between the user-specified and the PTD-specified time limit is used.

Section 4
Using Tape Files in a Program

The MCP systems support a wide range of tape devices such as reel-to-reel tapes and half-inch cartridge tapes, 8mm and 4mm cartridge tapes. The I/O subsystem enables you to read tapes of many different types, and to create both unlabeled and labeled tapes.

A single reel or cartridge is referred to as a volume. A volume can contain more than one file, and a file can occupy more than one volume. A *single-file* single volume tape contains one file on one volume, but the contents of a *multivolume* file are spread across two or more volumes. As a programmer, you do not have to be concerned with controlling when the file goes to another volume. The physical I/O subsystem and logical I/O subsystem take care of the necessary tasks. Continuation volumes are assigned to a medium that is compatible with the previous volume.

When two or more files reside on one volume, the tape is known as a *multifile* tape. A *multifile multivolume* tape contains more than one file and part of a file that began on another volume or begins on the current volume.

Although you can give a tape a file name of up to 12 nodes, ANSI standards allow only two nodes to be placed in the tape label. If your file name has more than two nodes, the I/O subsystem uses only the first and the last nodes. Thus, a tape file name of A/B/C/D is stored in the tape label as A/D. When more than one file is on one volume, the first node must be the same for all files on the volume, even if a file spans a number of physical volumes. Additionally, no two files on a volume can have the same second node.

Any tape created has at least one tape mark to delimit the logical entities on the tape, and the last valid data on a tape is followed by two tape marks.

A tape can be either labeled or unlabeled. A *labeled* tape has label records that contain information needed to locate a specific file on the tape and a serial number that is initially assigned by an operator using the SN (Serial Number) system command. Refer to the *System Operations Guide* for information about using the SN command.

Each file on a labeled tape is preceded and followed by a set of label records. A tape mark is used to separate the label records from the records of a file on the tape. Refer to Appendix E, "Standard Tape Label Formats," for information about the exact contents of tape labels.

As a multifile multivolume tape is created, the creation date in HDR1 and EOF1 is updated for each file on the volume. For example, FILE/1 is created on July 10, and it goes to a second volume on July 11 where the file ends. FILE/2 is created and follows FILE/1 on the second volume. FILE/1 (on reels one and two) has a creation date of July 10, and FILE/2 has a creation date of July 11.

An *unlabeled* volume has no label records and no serial number on the tape.

A file on the following tape drives cannot be accessed in the reverse direction, nor can a block be rewritten in place:

- CLU9710-36T, CTS5136, CTS5236, and OST5136
- CLU9710-DLT4, CLU9710-DLT7, and CLU9710-DLT8
- CTS9840
- HS4400
- HS8500, HS8500C, HS8505
- ALP430
- ALP920
- FIPS 5073, USR5073 (only when compression is on)

Note: *Extending a file whose last block is not full would require rewriting the last block, and so cannot be done.*

The steps needed to accomplish the following tasks are presented in this section:

- Creating a tape file
- Reading a tape file
- Reading a tape file in reverse
- Creating an unlabeled tape file
- Creating a tape with more than one file
- Accessing an unlabeled tape
- Treating labeled tapes as unlabeled tapes

If you are not familiar with basic programming methods, review those methods in Section 2, "Understanding Programming for Files," before reading the following information. You can identify all the file attributes that can be used with a tape file by reviewing Table A–1. For more information about any of the file attributes, refer to the *File Attributes Reference Manual*.

Creating a Tape File

The task of creating a tape file has been broken down into the following groupings:

- Tasks that are required
- Tasks the ensure file security
- Tasks that define complex record structures
- Tasks that allow for special handling of the file

Required Tasks

You should perform all the following tasks:

- Specify the KIND value as TAPE. The default value is DONTCARE.
- To specify that this file is new, set the NEWFILE attribute to TRUE.

  If you want to read the file after you create it, be sure to set the NEWFILE attribute to FALSE before you reopen the file. Header labels overwrite the existing file on the tape if the file is reopened without first changing the NEWFILE attribute to FALSE.
- If you want the physical file name to be different from the internal file name, specify the physical file name by using the FILENAME attribute.
- Identify how the data is going to be transferred by using the FRAMESIZE attribute. Refer to "Identifying How Data Is Transferred" in Section 2.
- If you want an INTMODE attribute value other than the default value assumed by your language, assign that value to INTMODE. Refer to "Identifying How Data Is Transferred" in Section 2 for language default information. The INTMODE value is assumed by the EXTMODE value. If the physical file must have a character encoding set that is different from the INTMODE value, assign EXTMODE the appropriate value. Refer to Table 2–14 for possible EXTMODE and INTMODE combinations.
- Identify the maximum size of any record by using the MAXRECSIZE attribute. You should express the MAXRECSIZE value in terms of FRAMESIZE units. A tape file defaults to 10 words.
- To indicate that the file has more than one record in a block, specify the length of the block by using the BLOCKSIZE attribute. You should express the BLOCKSIZE value in terms of FRAMESIZE units. Because BLOCKSIZE defaults to the MAXRECSIZE value, you must specify a value larger than MAXRECSIZE. In the case of a file with fixed-length records, the value must be a multiple of MAXRECSIZE.

  The minimum block size—that is, the physical record size—for tape files is six words or 36 EBCDIC bytes. Blocks that are shorter than the block size are padded with zeros, if necessary. You should be careful to ensure that each block written to the tape—including the last block—is at least the minimum block size in length.

For instance, you might have a problem in the following situation:

– You write a file made up of fixed-length records that have a MAXRECSIZE of less than 36 bytes long and have more than one record for each block, and the BLOCKSIZE of the file is less than 36 bytes.

– You read the file, which can cause the I/O subsystem to pass the program one or more records that are all zeros.

When you close such an input file, the I/O subsystem sometimes displays a record count error message. Such an error does not occur if all the records are at least six words or 36 EBCDIC bytes long—for example, if MAXRECSIZE is at least this long and the file has fixed-length records.

The I/O subsystem pads blocks when the block size of the data is an odd number of bytes. The I/O subsystem writes the tape in an even-numbered block size. If padding occurs, the tape header contains the odd number for the block size of the data, and the I/O subsystem writes the physical tape with a block size that is one greater than the odd number. You can avoid these problems by specifying records that contain an even number of bytes when you write the file.

Be aware that systems other than MCP systems might not be able to read a tape that has been padded in this way.

Some tape drive models have maximum block-size restrictions. These restrictions are described in the following table:

Tape Drive Model	Maximum Block Size
2145 GCR/PE/NRZ, ALP430	65535 bytes
CLU9710-36T, CTS5136, CTS5236, CTS9840, OST5136	262144 bytes
FIPS 5073 half-inch cartridge	131068 bytes
HS8500, HS8505, HS8500C	245760 bytes
USR5073 half-inch cartridge	262139 bytes
HS4400	262139 bytes
All others	393210 Bytes

When connected to certain system types, such as the NX4200 and LX systems, and CS servers that are delivered with Virtual Machine for ClearPath MCP software, the maximum block size can be restricted to 65,535 bytes. You can check the block size supported by a tape drive/system combination by checking the OL MT display for the drive.

Security Tasks

Consider performing the following tasks to secure your file if you use the Security Accountability Facility on your system and if the SECOPT TAPECHECK form of the SECOPT (Security Options) system command is set to AUTOMATIC.

If the SECURITYLABELS tape volume security option is TRUE, the values for the FAMILYOWNER, SECURITYGUARD, SECURITYTYPE, and SECURITYUSE attributes are stored in the tape label and the tape volume directory. Refer to *WFL Reference Manual* for information about using SECURITYLABELS.

- Specify the owner of the tape volume by using the FAMILYOWNER attribute. Refer to the *Security Administration Guide* for more information about security.
- If you want to restrict who can use the file, assign a mnemonic value to the SECURITYTYPE attribute. All privileged users have access to all files regardless of the SECURITYTYPE value. The following are the possible values you can specify:

Mnemonic Values	Meaning for a Nonprivileged User
PRIVATE	The owner can access the file.
PUBLIC	Access by a nonowner is allowed.
GUARDED	Access by nonowner users is controlled by a guard file.
CONTROLLED	A guard file controls access by all users including the owner. This value is not supported by Host Services logical I/O.

- If you chose the CONTROLLED or GUARDED values, specify the name of the guard file by using the SECURITYGUARD attribute.
- To specify how a physical file protected with a SECURITYTYPE value of PUBLIC can be accessed by nonprivileged users using nonprivileged programs, assign the SECURITYUSE attribute one of the following values:

Mnemonic Values	Meaning
IN	Specifies read-only access to source files, data files, and code files. Also, a code file can be executed.
IO	Specifies read and write access to source files, data files, and code files. Also, code files can be executed. The default value of SECURITYUSE is IO for all disk files.
OUT	Specifies write-only access to source files, data files, and code files. Also, code files can be executed.
SECURED	Specifies that nonprivileged users do not have access to source files, code files, or data files, but they can execute a code file. For example, a nonprivileged user cannot copy a secured code file, but he or she can still execute it.
DLT3	Digital linear tape (DLTIII)

Mnemonic Values	Meaning
DLT6	Digital linear tape (DLTIII)
DLT10	Digital linear tape (DLTIII or DLTIIIxt)
DLT20	Digital linear tape (DLTIV)
DLT35	Digital linear tape (DLTIV)
DLT40	Digital linear tape (DLTIV)

Complex Record Tasks

If you want to define variable-length records, refer to "Establishing a Record Format" in Section 2.

Special Requirement Tasks

Consider performing the following tasks if you have special requirements:

- If you want to be able to read any data that was written to the tape, even if the file creation process is interrupted by a halt/load, assign a value of PROTECTED to the PROTECTION attribute.
- If you want to ensure that the tape file cannot be accidentally purged, set the LOCKEDFILE file attribute to TRUE. Once you have set the value to TRUE, any programmatic attempt to close and purge the file results in the file being closed, but not purged. To purge the tape with a system command requires operator confirmation.
- If you want tape reels unloaded after they are used, so that they can be put away or made inaccessible to other files, your program can invoke a CLOSE operation that results in a disposition of lock. Refer to Section 2, "Understanding Programming for Files," for information about closing a file.

  Another method of controlling the unloading of tapes is to assign the AUTOUNLOAD file attribute a value of ON. When the value is ON, any CLOSE operation that has a disposition of purge or an association of release and a disposition of rewind or a disposition of block exit causes the tape to unload, regardless of the automatic-unload mode of the unit. Refer to Section 2, "Understanding Programming for Files," for information about CLOSE operations.
- If you do not want the tape to be identified as a scratch tape as soon as the tape is released by your program, set the SAVEFACTOR attribute to the number of days that you want the tape saved.

  The operating system handles an expired tape based on the setting of the system option TAPEEXPIRATION, which is controlled by using the *SYSOPS TAPEEXPIRATION* system command. If the system option is set to TRUE, an expired tape is handled as a scratch tape. If the system option is set to FALSE, the expiration date is ignored.

Note: *A tape that is past its expiration date is identified as a scratch tape by the operating system only if it is write-enabled. The data on the tape is not actually purged. Thus, a tape that is past its expiration date can still be read if it is not write-enabled. On reel-to-reel tape drives, a tape is write-enabled if it has a write ring.*

- If you have any of the following types of tape drives on your system, and the file you are creating must be written to one of these tapes drives, use the appropriate DENSITY attribute value to designate the particular tape drive. For the following tape types, if the I/O subsystem cannot find a tape unit that supports that density, it places the user task in the waiting state and issues a request for a tape unit that does support that density.

Mnemonic Value	Integer Value	Media Type	Tape Subsystems
BPI800	0	9-track NRZ reel-to-reel tape	2145, 4125
BPI1600	3	9-track PE reel-to-reel tape	2145, 4125
BPI6250	4	9-track GCR reel-to-reel tape	2145, 4125
BPI38000	5	18-track half-inch cartridge tape and 4mm cartridge tape	RM5073, HS4400 Notes: *The HS4400 4mm emulates a half-inch cartridge tape, thereby inheriting the density of the RM5073 tape.* *18-track tapes are read-only on subsystems supporting 36-track HIC media.*
BPI1250	6	Quarter-inch cartridge tape	QIC, QIC1000
BPI11000	7	8mm cartridge tape	HS8500 (Exabyte)
FMT36TRK	8	36-track half-inch cartridge tape	CTS5136, OST5136, CTS5236, and CLU9710-36T
FMTDDS2	9	DDS–2 cartridge tape	HS4400 and ALP430
FMTQIC1000	10	Quarter-inch cartridge tape	QIC1000
FMTDDS3	11	DDS–3 cartridge tape	ALP430
FMTDLT3	13	Digital linear tape–DLTIII	CLU9710-DLT4 and DLT7 ***Note:*** *Tapes are read-only on these subsystems.*

Mnemonic Value	Integer Value	Media Type	Tape Subsystems
FMTDLT6	14	Digital linear tape–DLTIII	CLU9710-DLT4 and DLT7 ***Note:*** *Tapes are read-only on these subsystems.*
FMTDLT10	15	Digital linear tapes–DLTIII or DLTIIIxt	CLU9710-DLT4, DLT7 and DLT8
FMTDLT20	16	Digital linear tape–DLTIV	CLU9710-DLT4, DLT7, and DLT8
FMTDLT35	17	Digital linear tape–DLTIV	CLU9710-DLT7 and DLT8
FMTST9840	18	High-capacity cartridge tape	CTS9840
FMTDDS	19	DDS–1 cartridge tape	HS4400 and ALP430
FMTAIT	21	Advanced intelligent tape	ALP920
FMTAIT2	22	Advanced intelligent tape	ALP920
FMTDLT40	23	Digital linear tape–DLTIV	CLU9710-DLT8

- If you have a tape drive that allows compression and you want the data to be compressed as it is written to tape, perform one of the following tasks:
  - If you want compression to occur based on the compression flag maintained in the tape label, specify SYSTEM as the COMPRESSIONCONTROL file attribute value. The compression flag in the tape label can be controlled by an operator by using the SN (Serial Number) or PG (Purge) system commands.
  - If you want the program to control whether compression occurs or not, specify USER as the COMPRESSIONCONTROL attribute value and set the COMPRESSIONREQUESTED file attribute to TRUE if you want compression to occur.

  Note: *Compression cannot be used for tasks that need to predict ahead of time how much data can be written to the tape, nor can it be used for tape files that need to be read in reverse.*

- If you are not using direct I/O and you want to ensure that every WRITE operation has completed before going on to the next instruction in the program, set the SYNCHRONIZE attribute to the value OUT. This causes a separate block to be written for each record even when the BLOCKSIZE value allows for more than one record. That is, the I/O subsystem ignores the BLOCKSIZE and uses the record size for WRITE operations.

You should be aware that, for certain styles of tape drives having a long repositioning time (such as the HS8500), setting the SYNCHRONIZE attribute to NO can avoid significant performance degradation.

If you want your program to occasionally ensure that a WRITE operation has completed before going on to the next instruction in the program (and you are not using direct I/O), use a WRITE statement with the SYNCHRONIZE operation each time you want this behavior.

Refer to Section 10, "Using Direct I/O Files," for information about enabling buffering mode when using direct I/O.

- If your site uses cataloging and the USECATDEFAULT system command is not set, and you want the file to be entered into the system catalog, set the USECATALOG attribute to TRUE. The physical tape that is to be used must have been added to the tape volume directory by using the Work Flow Language (WFL) VOLUME ADD statement.
- If you want to differentiate the file from a file that has the same file name, use the CYCLE attribute. The value of the CYCLE attribute can be changed by the operator each time the program is run by specifying a value for the CYCLE attribute in a FILE statement that is included with the RUN statement of the program. Additionally, you can use the VERSION attribute to differentiate the file from other versions of the file within the cycle.

  To request that the tape file be placed on a tape with a specific serial number, use the SERIALNO attribute. If your file requires more than one physical volume, you can specify the serial number of each volume by using the optional FILESECTION number parameter of the SERIALNO attribute. The following ALGOL statements would specify the serial numbers of the volumes of a three-volume file:

```
F(1).SERIALNO :="FIRST ";
F(2).SERIALNO :="SECOND";
F(3).SERIALNO :="THIRD ";
```

  The system option SERIALNUMBER (option 27) controls the assignment of scratch tapes when the SERIALNO and SCRATCHPOOL file attribute values are not specified. The assignment of scratch tapes is controlled in the following ways:

  – If the option is set, no file assignment automatically takes place and the "<file name> REQUIRES" message is displayed. The operator can respond with a DS (Discontinue) or OU (Output Unit) system command, or can specify a serial number or scratch pool name with the FA (File Attribute) system command.
  – If the option is reset, the system uses any available scratch tape that does not have a scratch pool name.

  For further information about using the OP (Options), DS, OU, and FA system commands, refer to the *System Operations Guide*.

  If you choose to specify a serial number and that serial number matches the serial number of the tape, the system assigns the file whether or not the file has expired or the tape is a scratch tape, as long as the tape is write-enabled. Non-scratch volumes are not assigned to output tapes if the TAPEOVERWRITE option of the SYSOPS system command is set to FALSE.

To indicate that the serial number is not to be considered during file assignment, set the SERIALNO attribute to all null characters; that is, a value in which all bits are equal to 0 (zero).

- If you want the created file to be stored on a physical tape that is from a specific pool of tapes, use the SCRATCHPOOL attribute. The name you use must be an identifier of 1 to 17 EBCDIC characters, left-justified in a field of blanks (a hyphen or an underscore is not permitted as the first character).

  You can place a scratch tape in a specific pool by naming that pool when you use the PG (Purge) or SN (System Number) system command.

  If you request a printer backup tape with a specification of SCRATCHPOOL, the file will be added on to any existing, mounted backup tape from that pool.

  If the SERIALNO file attribute has been specified for a particular member of a single volume or a multivolume set, the SCRATCHPOOL attribute is not taken into consideration when that volume is assigned.

  If a tape volume is closed with the PURGE option, and the SCRATCHPOOL value is not a null string, the SCRATCHPOOL value is used as the pool name for the volume in the resulting purge operation.

Reading a Tape File

Perform all or some of the following tasks, depending on the purpose of your program:

- If the tape has a physical file name that does not match the internal file name, specify the physical tape name of the file by using the FILENAME attribute.
- Specify the KIND value as TAPE.
- Specify the NEWFILE attribute value as FALSE.
- You should set the DEPENDENTSPECS attribute value to TRUE, unless you have a special reason to manipulate the records differently than the creation program intended.

  If you do not set the DEPENDENTSPECS attribute to TRUE and if you specify BLOCKSIZE, MAXRECSIZE, and MINRECSIZE values that are inconsistent with the physical file values, your program might not behave the way you expected it to. For example, the program might not read all the data originally written to the file. If such an inconsistency exists, a run-time warning is issued during the process of opening the file.

  If your program receives an inconsistent blocking warning, you should examine the program to determine if the blocking of the logical file should be different from the blocking of the physical file and if your program can function correctly with the values you specified. If your program functions correctly, you can suppress the warning by using the SUPPRESSWARNING task attribute. If the difference is not necessary, set the DEPENDENTSPECS value to TRUE or change the three values of the logical file to match the physical file values. For more information about the conditions that cause warnings, refer to the BLOCKSIZE, MAXRECSIZE, and MINRECSIZE descriptions in the *File Attributes Reference Manual*.

  Certain tape drive models have block size restrictions as shown in the following list. In addition, the type of connection to a tape unit might also impose a limit on the

maximum BLOCKSIZE that can be used. An emulated SCSI DLP or an emulated Native SCSI Channel limits the maximum BLOCKSIZE of any tape connected to it to 65,535 bytes.

- The 2145 GCR/PE/NRZ and ALP430 tape drives have a 65,535-byte maximum.
- The CTS5136, CTS5236, CLU9710-36T, CTS9840 half-inch cartridge tape drives have a 262,144-byte maximum.
- The OTS5136 half-inch cartridge tape drive has a 262,144-byte maximum.
- The FIPS 5073 half-inch cartridge tape drive has a 131,068-byte maximum.
- The HS8500, HS8505, and HS8500C tape drives have a 245,760-byte maximum.
- The USR5073 half-inch cartridge tape drive and the HS4400 4mm cartridge tape drive have a 262,139-byte maximum.
- All other tape drives have a 393,210-byte maximum.

Caution

The NX4200 and LX systems, and CS servers that are delivered with Virtual Machine for ClearPath MCP software, use Windows NT as the underlying operating system and cannot read or write a block of data from tape greater than 65,535 bytes. This is a Microsoft limitation on I/O drivers. As a result, any tape written on another system with a BLOCKSIZE greater than 65,535 bytes cannot be read by the NX4200, LX, and CS servers that are delivered with Virtual Machine for ClearPath MCP software.

- If you want tape reels unloaded after they are used, so that they can be put away or made inaccessible to other files, your program can invoke a CLOSE operation that results in a disposition of lock. Refer to Section 2, "Understanding Programming for Files" for information about closing a file.

  Another method of controlling the unloading of tapes is to assign the AUTOUNLOAD file attribute a value of ON. When the value is ON, any CLOSE operation that has a disposition of purge or an association of release and a disposition of rewind or a disposition of block exit causes the tape to unload, regardless of the automatic-unload mode of the unit. Refer to Section 2, "Understanding Programming for Files," for information about CLOSE operations.
- If you want a particular cycle and version of the file, specify the particular cycle number in the CYCLE attribute and the particular version number in the VERSION attribute.
- If you are using cataloging at your site, you can do the following:
  - If the USECATDEFAULT system option is not set and you want the system catalog searched when the system assigns the permanent file, set the USECATALOG attribute to TRUE.
  - If you want a copy of the file that does not have the latest time and date, specify the desired copy in the GENERATION attribute.

- If you want to read a tape file that resides on a tape that has a specific serial number, use the SERIALNO attribute to identify the correct serial number. If you do so, the I/O subsystem assigns the physical file only if the serial number of the tape on which it resides matches the value you specified for the SERIALNO attribute and the file meets the other selection criteria.
- If the file is split across volumes and you want information from a specific volume, you can specify that volume by using the FILESECTION attribute.
- After you invoke READ statements, you can interrogate the following attributes to obtain certain information. This is not a complete list. Remember that almost all attributes for a tape file can be interrogated.

  The following table lists some attributes that can be interrogated and identifies what information can be obtained:

Attribute	Information Received
CREATIONDATE	The creation date of the tape file.
EOF	TRUE indicates that the end-of-file condition has been reached.
IOINERROR	TRUE indicates that a physical I/O error has occurred on the file.
RECORDINERROR	The record number or block number of the information in the currently used buffer.
TAPEREELRECORD	The logical record number relative to the beginning of the current volume.

Reading a File in Reverse

You can read the file backwards if the tape is positioned at the end of the file and has been closed if reverse READ operations are supported on the drive. Among those drives that do not support reverse reading are the HS8500, HS4400, FIPS 5073, CTS5136, and OST5136 tape drives. In addition, half-inch cartridge tapes that have been written in EDRC format cannot read in reverse. You can verify whether a tape drive supports reverse reading by checking the OL MT display for the drive.

To indicate that you want to read the file from back to front, assign the DIRECTION attribute a value of REVERSE.

To indicate that you want to change the direction of the READ operation back to the forward direction, assign the FORWARD value to DIRECTION. FORWARD is the default value.

To specify that you need a read-reverse capable unit, use the READREVERSECAPABLE attribute.

When a tape is read in the reverse direction, the buffer is filled from the end to the beginning so that the data are in the order in which they were written to the tape—the forward direction. With this method, the buffer image is the same no matter in which direction the tape is read, unless the length of the block on the tape is shorter or longer than the length or size of the buffer. If the block is short, the last characters or words of the buffer are filled with blanks, and they are the image of the tape in the forward direction. The rest of the buffer is undisturbed. If the block is long, the buffer contains the first characters or words of the tape block that are encountered as the block is read backwards. Thus, the first data are missing from the buffer.

Direct I/O files allow you to change the direction of a READ operation at anytime.

Creating an Unlabeled Tape

To create an unlabeled tape, perform those tasks that are appropriate from the identified tasks in "Creating a Tape File" in this section. In addition, assign the OMITTED value to the LABEL attribute.

Creating a Tape with More Than One File

It is sometimes useful to put more than one file on a physical tape. To accomplish this task, close the logical file, do not rewind the tape, and leave the unit assigned to the program. The ALGOL statement CLOSE(F,*) and the COBOL statement CLOSE <file-ID> WITH NO REWIND leave the logical file assigned to the physical file.

Naming Conventions

When you write more than one file to a physical tape, each file must have a two-node tape file name. The first node should be the same for all files on the physical tape, even if the physical tape includes a number of physical reels or volumes. Additionally, no two files on the tape should have the same second node.

Searching Conventions

If two files have the same first and second nodes, the second file can be located only if it is on a different physical volume of the logical tape file and if the different volume does not begin with the continuation of the first file of the same name. You cannot locate two files with the same name on the same tape volume by assigning different values to the CYCLE and VERSION attributes.

The file search routines make extensive use of these name restrictions in order to reduce the time taken to find a tape file. Files on tapes that do not meet these criteria might not be found without operator intervention.

The following method is used to search for files on multifile tapes.

First, all units associated with the process attempting to open the file are searched. If the file is on one of those units, the unit is assigned and the search terminates. If the file is not on one of those units, all other tapes whose first-node names match that of the required file, except those that are continuations of the required file, are searched. When a unit is searched unsuccessfully, it is not locked; the serial number of the searched tape is kept by the search routines to indicate that the tape has already been searched. If an operator wants the system to search a tape with a serial number identical to a tape that has already been searched, the IL (Ignore Label) system command should be used.

Logical I/O does not rewind tapes to search for files if it can determine that the file is not on the tape. This determination is based on the first node of the file name or the file section number it is seeking. Specifically, tapes that are online are rewound only if the first node name of the first file on the tape and the file being sought match, and if the file section number of the first file on the tape matches the one being sought, or if the one being sought is 1. This means that searching for a continuation reel rarely causes the tape to rewind.

Note: *In some instances, when more than one task is trying to open tape files with the same multifile ID, the right tape might not be found on the first pass if another task is already searching the tape.*

In ALGOL, you can space past the last file on a labeled tape by invoking the OPEN(F);CLOSE(F,*) statement pair a sufficient number of times. When this has been done, attributes that return actual values from the current physical file, such as CREATIONDATE or LABELKIND, return information appropriate to an unlabeled tape. Normally, the returned value is the value declared by the user. If the tape is positioned past the last file on the tape, some file attributes such as CREATIONDATE return attribute errors. The LABELKIND attribute returns a value of 1, which indicates that the tape is unlabeled.

Accessing an Unlabeled Tape

You can access the data on an unlabeled tape by performing the following tasks:

1. Identify the file as a tape file by assigning the KIND attribute a value of TAPE.
2. Assign the NEWFILE attribute a value of FALSE.
3. Specify values for the MAXRECSIZE, BLOCKSIZE, INTMODE, EXTMODE attributes and any other attributes you would normally assign a value to, such as BLOCKSTRUCTURE. Do not assign DEPENDENTSPECS a value of TRUE.
4. Indicate that the I/O subsystem should treat the file as an unlabeled file by assigning to the LABEL attribute one of the following values:

Mnemonic Value	Behavior When a Tape Mark Is Encountered
OMITTEDEOF	An end-of-file action occurs.
OMITTED	A volume switch is attempted. An operator can use the FR (Final Reel) system command to indicate that the end of the file has been reached. Refer to the *System Operations Guide* for information about using the FR command.

5. When the file is opened by the program, the operating system suspends the program and displays the following message:

   ```
   <mix number> NO FILE <file name> UL (UNLABELED MT)
   ```

   The operator should identify the location of the unlabeled tape by entering the following response:

   ```
   <mix number> UL MT <unit number>
   ```

6. To position the tape at the desired file on a multifile volume, you will need to know the position of the file on the volume. Is it, for example, the first file or the third file on the tape? For each file that must be bypassed, you must open and close the file one time, and then you must open the file again to allow the data to be read.

   The following are appropriate open and close statements to bypass a file:

ALGOL	`OPEN(F);`
	`CLOSE(F,*);`
	`OPEN(F);`

COBOL74 and COBOL85	`OPEN INPUT IN-FILE.`
	`CLOSE IN-FILE WITH NO REWIND.`
	`OPEN INPUT IN-FILE WITH NO REWIND.`

Treating Labeled Tapes as Unlabeled Tapes

The operating system permits most tapes to be treated as if they were unlabeled. With the exception of tape marks, no interpretation is placed on any data found on the tape. The data contained on the tape is assumed to comprise one or more files. File boundaries are delimited by tape marks. All such groupings of data can be read.

For example, to access a labeled tape as an unlabeled tape, perform the steps 1 through 5 in the procedure for "Accessing an Unlabeled Tape" earlier in this section.

To position the tape at the desired file, you must know the position of the file on the tape. Is it the first file or the third file on the tape? For each file that must be bypassed, you must open and close the file three times; once for the header label, once for the file data itself, and once for the trailer label. Thus, if your file is the third file on the tape, you must open and close the file seven times to position the tape at the data portion of the third file—3 times for each of the first two files being bypassed and 1 more time to bypass the header label of the third file—and then you must open the file one more time to allow the data to be read. If your file is the first file on the tape, you must open and close the file one time to position the tape at the data portion of the file, and then you must open the file one more time to allow the data to be read.

The following are appropriate open and close statements to bypass a file:

ALGOL

```
OPEN(F);
CLOSE(F,*);
OPEN(F);
```

COBOL74 and COBOL85

```
OPEN INPUT IN-FILE.
CLOSE IN-FILE WITH NO REWIND.
OPEN INPUT IN-FILE WITH NO REWIND.
```

Section 5
Using Printer Files in a Program

In the MCP environment, you can choose to print directly to a printer, or you can choose to have a printer backup file created. Choosing to create a printer backup file saves your program execution time, because the printer backup file is not dependent on the speed or immediate availability of the printer. You can print the backup file at a convenient time. Using printer backup files also allows you to take advantage of the flexibility of the print subsystem. Following are some of the tasks that can be accomplished by the print subsystem:

- Controlling when the print request is issued
- Aligning the appropriate forms
- Attaching a customized banner to the beginning of the print job
- Controlling where the print request is printed and how many copies are printed
- Identifying who is charged for the printing
- Controlling the name of the backup file
- Requesting that a checkpoint be taken while the file is printed
- Controlling the format of printed output

Note: *You can create printer backup files with the EXTMODE file attribute set to HEX, 8-bit, 16-bit, or mixed multi-byte character. However, the Print System can only print files with 8-bit characters.*

If you are not familiar with basic programming methods, review those methods in Section 2, "Understanding Programming for Files," before reading the following information.

You can identify all the file attributes that can be used with a printer file by reviewing Table A–1. For more information about any of the mentioned attributes, refer to the *File Attributes Reference Manual*.

For information about the capabilities of the print subsystem, refer to the *Print System Guide*.

Defining the Characteristics of a Printer File

Perform all or some of the following tasks, depending on the purpose of your program.

- Specify the KIND attribute value as PRINTER.
- Identify the structure of the file.

  Printer files can be declared as blocked files. The most extreme case of blocking, where the MAXRECSIZE value is 1, the BLOCKSIZE value is 132, and the FRAMESIZE value is 8, can be useful in printing graphs. When a printer file is blocked, a WRITE statement of zero length can be used to terminate the block.
- Decide if you want to print directly to the printer or create a printer backup file and whether you want to control all aspects of printing or allow the Print System to control printing.

  To print directly to the printer and control all aspects of printing, set the PRINTDISPOSITION file attribute to one of the following values:

  – DIRECTDLP

  – DIRECT, when the value of the DIRECTPRINTER option of the SYSOPS system command is DIRECTDLP.

  Notes:

  - *Setting the DIRECTDLP value works only on systems with printers directly attached.*
  - *If you set PRINTDISPOSITION to the value DIRECT, the system option SYSOPS DIRECTPRINTER changes the PRINTDISPOSITION to DIRECTPS or DIRECTDLP, depending on the value of DIRECTPRINTER.*

  To print directly to the printer but allow the Print System to control printing, set PRINTDISPOSITION to one of the following values:

  – DIRECTPS

  – DIRECT, when the value of the DIRECTPRINTER option of the SYSOPS system command is DIRECTPS.

  – NOW

  Notes:

  - *The DIRECTPS value is valid for both direct and non-direct I/O files. The NOW value provides additional functionality from the Print System, but is valid only for non-direct I/O files.*
  - *If you set PRINTDISPOSITION to the value DIRECT, the system option SYSOPS DIRECTPRINTER changes the PRINTDISPOSITION to DIRECTPS or DIRECTDLP, depending on the value of DIRECTPRINTER.*

To create a printer backup file without printing the file, set PRINTDISPOSITION to DONTPRINT.

To create a printer backup file and allow the Print System to print the file, set PRINTDISPOSITION to one of the following values:

  – EOJ
  – EOT
  – FILECLOSE
  – FILEOPEN

- If you need to specify a specific print train, identify that print train by using the TRAINID attribute. Refer to the TRAINID attribute in the *File Attributes Reference Manual* for a list of possible values.
- If your job requires a specific type of paper or form, use the FORMID attribute to notify the operator of this requirement. If no printer matches the value of the FORMID attribute, the operator can assign a printer form request to the printer. Refer to the *System Commands Reference Manual* for information about performing this task.
- If your program is not creating a printer backup file, the system prints a standard banner at the beginning and end of the file. If you do not want this standard banner to print, set the LABEL attribute value to OMITTED or OMITTEDEOF. Be aware that a top-of-page operation is performed when printing completes, regardless of the value of the LABEL attribute.

  Note: *If you use the FORMID attribute and set the LABEL attribute to OMITTED or OMITTEDEOF, no header and trailer pages print unless the HEADER and TRAILER print modifiers of the PRINTDEFAULTS of your task are set to UNCONDITIONAL. PRINTDEFAULTS is associated with your task, not the device configuration HEADER and TRAILER characteristics.*

- If you are creating a printer backup file and the INTMODE and EXTMODE attribute values have a value of EBCDIC, you can save space on the backup medium by letting the TRIMBLANKS value default to TRUE. This action causes the trailing blank characters to be removed from the file as the file is written to the backup medium. However, the processor time required to perform each WRITE operation is increased.

  Note: *If you are creating a printer backup file with an EXTMODE value greater than ASCII, the EXTMODE is no longer changed to EBCDIC. Therefore, EXTMODE values other than EBCDIC, such as LATIN1EBCDIC, can be specified for printer back up files.*

  When a logical file is assigned to an existing printer backup file, the value of INTMODE takes on the EXTMODE value. Translation occurs if the TRANSLATE attribute has a value of FORCESOFT. When the logical file is no longer assigned to the physical file, INTMODE is restored to its original, pre-assignment value.

- If you are creating a printer backup file, you can specify the specific backup device by using the BACKUPKIND attribute. You can choose among DISK, PACK, or TAPE. This value can also be changed by using the SB (Substitute Backup) system command or the LPBDONLY option of the OP (Options) system command. Refer to the *Print System Guide* for information about using these commands.

  Note: *If you use the TAPE option and an OPEN or CLOSE operation causes a volume switch to occur, the open or close errors could have values that are not normally associated with an OPEN or CLOSE operation.*

- You can also create a printer backup file as a delimited character-stream disk file. Backup files in delimited form can be transferred to other operating systems with a mechanism such as FTP.

  To create a delimited backup file, set the FILESTRUCTURE attribute to STREAM. The resulting disk file is created with the following attributes and values:

  – FILECLASS = CHARACTERSTREAM

  – EXTDELIMITER = CRCC

  – FILEKIND = PRINTFILE

  Although the printer backup disk file is a character-stream file, there is no change to the structure of the logical printer file. The file continues to be record-oriented and each block represents one print record. The MCP inserts the appropriate delimiters at the end of each block based upon the carriage control specified when writing the file:

  – Double spacing is represented in the printer backup file by an additional carriage return-line feed pair after the delimiters at the end of the first print line.

  – A channel skip is represented by a carriage return-form feed pair, regardless of the destination channel.

  Unlike a file with a FILEKIND attribute value of BACKUPPRINTER, there is no embedded binary control information. Software translation is allowed and the delimiters are inserted in the EXTMODE attribute character set. The EXTMODE value must correspond to a character set that is EBCDIC-based, ASCII-based, UCS2, or UCSNT. Refer to Appendix H, "Structure of Backup Files," for more information about printer backup files.

- If you want to ensure that your backup file cannot be removed or replaced and that the name of the file cannot be changed, set the LOCKEDFILE file attribute to TRUE. For a disk backup file, if the value is set to TRUE, the permanent file cannot be purged unless you or a privileged user changes the LOCKEDFILE value to FALSE. For a tape backup file, if the value is set to TRUE, the file cannot be purged programmatically, but can be purged if an operator confirms that the purge request is appropriate.
- If you are creating a printer backup file, you can use the file attributes AREAS, AREASIZE, AREALENGTH, and FLEXIBLE. The values specified affect the printer backup file created on disk and are interpreted in the context of the attributes of that disk file. When calculating the size of an area, the MAXRECSIZE and BLOCKSIZE of the resulting printer backup disk file are used. The default value of AREAS for a printer backup file is 15 and the default value of AREASIZE is 150. Additional areas can be added unless the value of the FLEXIBLE attribute is FALSE. Refer to

Appendix H, "Structure of Backup Files," for more information about printer backup files.

- You can reduce the necessary I/O time to write rules by specifying that FILESTRUCTURE = BLOCKED. Then the structure file is not different, but buffers are greater than the 300-word block size. You can modify the default value of the BUFFERSIZE file attribute if you set the BUFFERGOAL factor. Otherwise, the system default value is used.
- If you assigned a value of CLOSE, EOJ, EOT, or FILEOPEN to the PRINTDISPOSITION attribute and assigned a value of DISK or PACK to the BACKUPKIND attribute, you can indicate to the print subsystem that you want to print only a portion of the file by assigning a value to the PRINTPARTIAL attribute. If the value of PRINTDISPOSITION is FILEOPEN, then PRINTPARTIAL is restricted to only COLUMN selection phrases. The following example shows how to request that only the text located in columns 1 through 72 on lines 100 through 900 be printed instead of the entire file:

```
PRINT F1 (PRINTPARTIAL="COLUMN 1-72 SEQUENCE 100-900")
```

- If you are using certain data-comm-connected printers configured to use standard device transforms supplied by Unisys, you can exercise considerable control over the appearance of printed output by using the PAGECOMP attribute. Refer to the *Print System Guide* for information about the many options of the PAGECOMP attribute.
- If you want the file to be printed on a printer with certain characteristics, assign the appropriate mnemonic to the PRINTERKIND file attribute. Refer to the *File Attributes Reference Manual* for the possible mnemonics that you can assign.
- If you want to secure the backup file, perform the following tasks for any disk backup file. These tasks can also be performed for any tape backup file if you use the Security Accountability Facility on your system and the SECOPT TAPECHECK form of the SECOPT (Security Options) system command is set to AUTOMATIC.
  - Specify the owner of the tape volume by using the FAMILYOWNER attribute. Refer to the *Security Administration Guide* for more information about security.
  - If you want to restrict access to the file, assign one of the following values to the SECURITYTYPE attribute. All privileged users have access to all files regardless of the SECURITYTYPE value.

Mnemonic Values	Meaning for a Nonprivileged User
PRIVATE	The owner can access the file.
PUBLIC	Access by a nonowner is controlled by the SECURITYUSE attribute.
GUARDED	Access by nonowner users is controlled by a guard file.
CONTROLLED	Access by all users including the owner is controlled by a guard file. This value is not supported by Host Services logical I/O.

  - If you chose the CONTROLLED or GUARDED values, specify the name of the guard file by using the SECURITYGUARD attribute.
  - To specify how a physical file that is protected with a SECURITYTYPE value of PUBLIC can be accessed by nonprivileged users using nonprivileged programs, assign one of the following values to the SECURITYUSE attribute.

Mnemonic Values	Meaning
IN	Specifies read-only access to source files, data files, and code files. Also, a code file can be executed.
IO	Specifies read and write access to source files, data files, and code files. Also, code files can be executed. The default value of SECURITYUSE is IO for all disk files.
OUT	Specifies write-only access to source files, data files, and code files. Also, code files can be executed.
SECURED	Specifies that nonprivileged users do not have access to source or data files, but a code file can be executed. For example, a nonprivileged user cannot copy a secured code file, but can still execute it.

- If the backup file is to go to tape, and if you want to be able to read any data that is written to the tape, even if the file creation process is interrupted by a halt/load, assign the PROTECTION attribute a value of PROTECTED.
- If the backup file is to go to a disk, it is entered into the disk directory immediately after the file is opened. You can also protect that file with the PROTECTION attribute set to PROTECTED.

  When the PROTECTION attribute is set to PROTECTED, the file is entered into the disk directory immediately after the file is opened, and special action is taken to ensure that the correct end-of-file pointer is maintained across a system failure. If the FILESTRUCTURE attribute value of the file is STREAM, the end-of-file marker is placed at the end of the disk sector that was last written, even though the end of that sector might not be the end of a record.

  If the PRINTDISPOSITION attribute value of the file is FILEOPEN, the file is always PROTECTED.
- If you chose TAPE as the backup medium, consider the following tasks:
  - If you want the backup file to go to a tape with a specific serial number, use the SERIALNO attribute.
  - If you want the backup file to go to a tape with a specific scratchpool assignment, use the SCRATCHPOOL file attribute. Refer to the *File Attributes Reference Manual* for more information on the SCRATCHPOOL file attribute.

If you have any of the following types of tape drives on your system (Table 5–1), and the file you are creating must be written to one of these tapes drives, use the appropriate DENSITY attribute value to designate the particular tape drive. If the I/O subsystem cannot find a tape unit that supports that density, it places the user task in the waiting state and issues a request for a tape unit that does support that density.

Table 5–1. Tape Drive Density Values

Mnemonic Value	Integer Value	Media Type	Tape Subsystems
BPI800	0	9-track NRZ reel-to-reel tape	2145, 4125
BPI1600	3	9-track PE reel-to-reel tape	2145, 4125
BPI6250	4	9-track GCR reel-to-reel tape	2145, 4125
BPI38000	5	18-track half-inch cartridge tape and 4mm cartridge tape	RM5073, HS4400 ***Note:*** *The HS4400 4mm tape emulates a half-inch cartridge tape, thereby inheriting the density of the RM5073 tape.* *18-track tapes are read-only on subsystems supporting 36-track HIC media.*
BPI1250	6	Quarter-inch cartridge tape	QIC, QIC1000
BPI11000	7	8mm cartridge tape	HS8500 (Exabyte)
FMT36TRK	8	36-track half-inch cartridge tape	CTS5136, OST5136, CTS5236, and CLU9710-36T
FMTDDS2	9	DDS–2 cartridge tape	ALP430 tape
FMTQIC1000	10	Quarter-inch cartridge tape	QIC1000
FMTDDS3	11	DDS–3 cartridge tape	ALP430 tape
FMTDLT3	13	Digital linear tape–DLTIII	CLU9710-DLT4 and DLT7 ***Note:*** *Tapes are read-only on these subsystems.*
FMTDLT6	14	Digital linear tape–DLTIII	CLU9710-DLT4 and DLT7 ***Note:*** *Tapes are read-only on these subsystems.*
FMTDLT10	15	Digital linear tapes–DLTIII or DLTIIIxt	CLU9710-DLT4, DLT7, and DLT8

Table 5–1. Tape Drive Density Values

Mnemonic Value	Integer Value	Media Type	Tape Subsystems
FMTDLT20	16	Digital linear tape–DLTIV	CLU9710-DLT4, DLT7, and DLT8
FMTDLT35	17	Digital linear tape–DLTIV	CLU9710-DLT7 and DLT8
FMTST9840	18	High-capacity cartridge tape	CTS9840
FMTDDS	19	DDS–1 cartridge tape	ALP430 tape
FMTAIT	21	Advanced intelligent tape	ALP920 tape
FMTAIT2	22	Advanced intelligent tape	ALP920 tape
FMTDLT40	23	Digital linear tape–DLTIV	CLU9710-DLT8

Controlling the Printing of Lines and Pages

You can handle carriage control in two ways. The most common method is to use the syntax of the programming language. Another method is to use the information in the first character of the record. The CARRIAGECONTROL attribute allows you to specify what information is contained in the first character of the record.

- The CTLASA option indicates that the first character contains a value that specifies a given action. That action takes place before the line is printed.
- The CTL360 option indicates that the first character controls the paper motion by using the various fields in the character itself.

Refer to the CARRIAGECONTROL attribute in the *File Attributes Reference Manual* for the possible values.

If you are using COBOL, Report Writer gives you a great amount of printing flexibility without requiring a great amount of programming. However, if you need more control or are programming in ALGOL, the I/O subsystem allows you to set the page size and then keeps track of the lines that have been printed and the number of the current page. After the last line on the page is printed, the end-of-page indicator is returned, and if you check for that condition, you can code end-of-page routines to handle new page headings. To use this capability, do the following:

- Set the PAGESIZE attribute to the number of lines that you want on a page.
- When you use a WRITE statement to print a line, check the WRITE result for the end-of-page condition. Each time the end-of-page result is returned, the value of the LINENUM attribute is returned to 1, and the PAGE attribute value is incremented by 1. If the end-of-page result has not been reached, LINENUM is incremented by 1.

The following table gives you information about how the LINE, SKIP, and SPACE options of the WRITE statement affect the values of the LINENUM and PAGESIZE attributes.

Option	Effect
LINE	The action taken depends on the values of the arithmetic expression, the PAGESIZE and LINENUM values, and the language being used. When ALGOL is used and the WRITEAFTER compiler control option is FALSE, and the LINE option is used, the usual action of spacing after printing is temporarily suspended, and the printer is spaced forward before printing to the logical line specified by the arithmetic expression. An end-of-page exception is returned when the PAGESIZE attribute has a value of 0 (zero)—in other words, when logical page accounting is not being done. Otherwise, when the value of the arithmetic expression is less than or equal to the PAGESIZE value and greater than or equal to the LINENUM value, the page is spaced forward to the logical line number of the arithmetic expression, and LINENUM is set to the value of the arithmetic expression. If the value of the arithmetic expression is less than the LINENUM value, printing begins on the next logical page; the PAGE attribute is incremented by 1, but no skip to channel 1 is performed. The page is spaced forward to the logical line number of the arithmetic expression, and the LINENUM value is set to the value of the arithmetic expression. Printing is done either before or after spacing, depending on the value or values of the appropriate parameter or parameters in the language. When the arithmetic expression is less than 1, the LINENUM value is set to 1, the PAGE value is incremented by 1, the line is printed without spacing, and an end-of-page exception is returned to the program. A similar action occurs if the arithmetic expression is greater than the PAGESIZE value.
SKIP	The action is equivalent to a skip-to-channel operation on the carriage control tape. A skip-to-channel operation can affect the LINENUM value. A skip to channel 1 changes the LINENUM value to 1 after the skip. A skip to any other channel does not update the LINENUM value; therefore, after such a skip, the LINENUM value might not indicate the actual position on the page. The PAGE value is not incremented by skip-to-channel actions.

Option	Effect
SPACE	When the number of lines specified is greater than or equal to 0, the page is spaced forward the specified number of lines. If the number of lines specified is less than 0, the page is spaced forward 1 line and an end-of-page result is returned to the program. The LINENUM value is incremented by the number of lines spaced, and if the sum is greater than or equal to PAGESIZE value, the LINENUM value is set to 1, the PAGE value is incremented by 1, and an end-of-page result is returned to the program.

When you want to skip to another page before the page is full, change the LINENUM value to a value equal to or greater than the PAGESIZE attribute. This causes the next WRITE operation to return an end-of-page result, sets the LINENUM value to 1, and increments the PAGE value by 1.

If you want to add more lines to a page, subtract the number of extra lines wanted from the LINENUM value. Do not change the value of PAGESIZE.

Direct Printing through a Transparent Printer (XLP) DLP

Direct printing is used when you want data from your program delivered to the printing device rather than being spooled (usually to disk) by the MCP. The Print System normally is responsible for details of device handling. However, when the print system is bypassed, the programmer must take into account any idiosyncrasies of the device and its connection.

Printers generally expect ASCII data, and MCP based programs generally emit EBCDIC data. Because transparent printer (XLP) DLPs do not provide character translation, you must ensure that the program translates all data to ASCII characters before sending the data to the printer.

If your system has an OS/2 or UNIX environment and if the SPOOLER option is enabled on the printer, the program must send a data block of ESC FF at the completion of printing. The ESC FF command releases the print file and makes it possible for the OS/2 or UNIX environment to print the file on an actual printer.

Section 6
Using Remote Files in a Program

The I/O subsystem supports communication between a remote device and a program as long as the remote device is attached to the system through an data communications processor in the MCP environment.

Such communication is possible because your program regards the data being passed between the remote device and your program as a file. Remote file communication is comparable to the communication between a local peripheral device and your program. Remote files have essentially the same degree of device independence as other peripheral files.

Remote files work in conjunction with the message control system (MCS) that controls the remote device station. Possible MCSs in the MCP environment are

- Command and Edit (CANDE)
- Unisys e-@ction Transaction Server
- Other MCS programs that have been installed at your site

The MCS is responsible for logging the user on, handling error situations, assigning the station to logical files, and performing other functions required or desired by the MCS designers.

When a program opens a remote file, the MCS might participate in I/O. By participating, the MCS processes each I/O message and can selectively route each message to or from particular programs, files in programs, or particular stations. The MCS and the programs must use the same protocol so that the MCS can perform message switching and the program can identify source and destination stations. Without MCS participation, the I/O subsystem uses remote files to route the messages directly between the data communications subsystem and a program.

To communicate with the remote devices, a list of valid stations must be maintained. This is done through a DATACOMINFO file or through an MCS. This list of stations can be dynamically changed by using the STATIONLIST attribute in your program, or by using the Interactive Datacomm Configurator (IDC) or the commands of your MCS. Refer to the *IDC Operations Guide* for information about using the Interactive Datacomm Configurator.

When a family of stations is identified and the remote file is opened, a station list is created. This station list has all the necessary information to distinguish the different stations from each other.

When the station list is created, each station in the list is assigned a relative station number (RSN), which serves as an index to the station list. If a station is removed from this list, its RSN becomes invalid and points to an empty area in the station list, and the STATIONCOUNT attribute value is decreased by 1. As a consequence, a valid RSN can be larger than the value of the STATIONCOUNT attribute. When a station is added, it is added to the end of the list and given an RSN one higher than the last station in the list.

Your program does not need to deal with an RSN if you always want to direct the output to the terminal that sent the preceding input. However, if you want to change where the output is sent, you can modify the value of the LASTSUBFILE attribute. Doing so automatically modifies the WRITE statement to point to the station identified in the LASTSUBFILE attribute. For file attributes that require an RSN parameter, you can use a specific RSN number to point to the desired station.

If you are not familiar with basic programming methods, review those methods in Section 2, "Understanding Programming for Files." You can identify all the file attributes that can be used with a remote file by reviewing Table A–1. You can also find more information about any of the mentioned attributes in *File Attributes Reference Manual*.

Identifying the Characteristics of a Remote File

The following tasks should be performed before opening the file:

- Specify the KIND value as REMOTE.
- Identify the structure of the file.
- Specify whether you are going to use the file for input only, output only, or both input and output by setting the FILEUSE attribute to IN, OUT, or IO. If your remote file is used for I/O and is opened with a WRITE statement, language permitting, set the FILEUSE value to IO prior to the first I/O operation. You can set the value in the program or by compile-time or run-time file equation.
- If you do not want an I/O operation to wait indefinitely, use the TIMELIMIT attribute to specify how long, in seconds, the I/O subsystem should wait for an I/O operation to occur. This value is used differently, depending on the I/O operation.
  - When a READ operation is invoked and no input is received within the specified number of seconds, the READ operation is terminated with a TIMELIMIT error.
  - When a WRITE operation is invoked and no data can be buffered for output within the number of seconds specified, the WRITE operation is terminated with a TIMELIMIT error.

  You can also modify the TIMELIMIT value at the time a READ or WRITE statement is invoked if you are using ALGOL. The *[TIMELIMIT <arithmetic expression>]* form of the *[<record number or carriage control>]* part of the I/O statement allows this modification.

- If you want to reduce the data communications line time, and the INTMODE and EXTMODE attribute values have a value of EBCDIC, set the TRIMBLANKS attribute to TRUE. The trailing blank characters or words are stripped from the outgoing messages, but the processor time required to perform each WRITE operation increases.
- If your program is writing to a station, and you either want the output to be held or do not want the MCS to allow the output to be held at all, set the TANKING attribute to one of the following values:

Mnemonic Value	Meaning
ASYNC	The output is tanked. When the file closes, the task can proceed and even go to the end of the task. The output continues to be transmitted until there is no more output.
NONE	The output is not tanked.
SYNC	The output is tanked. When the file closes, the task cannot proceed until all tanked output has been transmitted.
UNSPECIFIED	The MCS controls whether or not the output is tanked.

- Your program can open any number of remote files, and can specify the stations with which these files can communicate by specifying a FILENAME file attribute value for each remote file declared in your program. If a DATACOMINFO file name with the same name exists in the DATACOMINFO file, the station or list of stations is used to communicate with the specific remote file. If no DATACOMINFO file name with the same name exists in the DATACOMINFO file, the system uses the specified FILENAME value as the initial station name in the station list for the remote file. Stations can be added or deleted from a station list by using the STATIONLIST file attribute.

  Note: *If your program is run from CANDE or MARC, all the remote files automatically communicate with the station that initiated the program, because these MCSs set the STATION task attribute to the logical station number of the station that initiated the program. If you prefer to let the FILENAME value select the stations to be used, set the STATION task attribute to 0 (zero) before the remote files are opened.*

Opening Remote Files

When a program opens a remote file, a "FILE OPEN" message for each station in the file is sent to the controlling MCS for the station. This message notifies the MCS that the logical file is ready to communicate with the station. The MCS can allow, postpone, or deny assignment of the station to the file.

- If the MCS allows assignment, the program can proceed to use the station and file, unless the data communications subsystem overrules this assignment. A denial is forced if the MCS did not participate, and either the station has no line assignment or the assignment would result in more than one concurrent input-capable file for the station.
- If the MCS postpones assignment, the file remains open and the station is not assigned. A subsequent READ or WRITE operation causes an end-of-file action. The field [26:10] in the result returned by the I/O operation contains the value 2 (ASSIGNMENTPOSTPONED).
- If the MCS denies assignment, which it might also do subsequent to having allowed or postponed assignment, further READ operations — if the file is input-capable—or WRITE operations to the station cause an end-of-file action. The field [26:10] in the result returned by the I/O operation contains the value 1 (ASSIGNMENTDENIED).

When a program executes a READ or WRITE operation and one of the following situations is true, an end-of-file action occurs, and the field [26:10] in the result returned by the I/O operation contains the value 3 (ILLEGALFILEUSE).

- The program already has assigned a remote file that is input capable to the station, and an MCS for the station is not participating in the I/O operations.
- The participating MCS indicated that an unacceptable FILEUSE value for the remote file was used.
- The program assigned an input-capable remote file to a station that is not input capable.
- The program assigned an output-capable remote file to a station that is not output capable.

After the file is opened you might want to determine the following information about the station or stations that are available for communication:

- If you are expecting more than one station to be assigned as the remote file opens, you might want to know which stations are available and which stations are not. Interrogate the STATIONSALLOWED attribute to determine the number of logical stations assigned to the remote file. By comparing that value with the value you can obtain from the STATIONCOUNT attribute, you can determine whether all the stations are available. You can also interrogate the STATIONSDENIED attribute to determine the number of the stations in a family that have been denied assignment to the file by their controlling MCS.
- If, under certain conditions, your program requires a station or a set of stations to be added to or subtracted from the remote file, use the STATIONLIST attribute. Then interrogate the LASTSUBFILE attribute to determine the RSN of the added station.

- If you need to know the specific name of a station or stations, you can index through the station list by using the STATIONNAME attribute.
- If you want to know if a specific station is available for use, interrogate the DISPOSITION attribute for the specific station.
- If the time the station was assigned is important to you, interrogate the ASSIGNTIME value for the specific station.
- If your program needs to determine whether the station is a screen device, interrogate the SCREEN attribute value for the specific station. A value of TRUE indicates that the device is a screen device. Having determined that the device is a screen device, you might want to know the number of lines for each page on the screen and the number of characters in a logical line. Interrogate the SCREENSIZE attribute value to determine the number of lines for each page on the screen and the WIDTH attribute value to determine the number of characters in a logical line for the specific station.

Reading Information from a Station

If your program reads information from a station or stations, it receives input from the remote file in a first-in, first-out order. After a READ statement, the LASTSUBFILE attribute contains the RSN of the station from which the input came. If you do not change this value by modifying the LASTSUBFILE value or by using the STATION option of the WRITE statement, the next output message is written to the same station the last input was received from. Thus, there is no need to determine where the input came from. A SEEK statement only changes the value of LASTSUBFILE; no information is moved into the buffer.

Interrogate the STATE attribute to determine whether an end-of-file condition has been encountered. A multistation remote file receives an end-of-file notification for each station in the file. To determine which station is no longer assigned to the file after an end-of-file notification, interrogate the LASTSUBFILE attribute. To determine the reason for the end-of-file notification, such as the file was closed or access was denied, interrogate the FILESTATE attribute for the specific station.

In a multistation situation, you might want to determine whether a specific station or any station is enabled for input. If you interrogate the ENABLEINPUT attribute for the entire remote file or for a specific subfile, you receive a value of TRUE when any file of the remote file is capable of sending input or when a specific file is ready to send input.

To determine if a message has been queued, interrogate the INPUTEVENT attribute. When the happened state of the INPUTEVENT value is TRUE, messages are queued. The INPUTEVENT attribute can be useful as a parameter to a WAIT statement that waits for more than one condition to occur.

To determine how many messages are queued, interrogate the CENSUS attribute.

In some instances, the transmission number of the last input received from a specific station is important to your program. To determine the current number for the specific station, use the TRANSMISSIONNO attribute. The DATACOMINFO file information for a specific station can indicate that no transmission numbers are to be assigned by the data communications subsystem. In such a case, a value of –1 is returned.

Writing Information to a Station

Your program can control the destination of the message in one of two ways:

- To broadcast the output to every station in the file, assign a 0 (zero) to the LASTSUBFILE attribute. If there is only one station in the file, the result is the same as for a WRITE operation directed to that station.
- To direct the output to a specific station in the file, identify an RSN by using the LASTSUBFILE attribute.

In ALGOL, the LASTSUBFILE attribute can be set using the *STATION <arithmetic expression>* form of the *[<record number or carriage control>]* part of the WRITE statement.

Closing a Remote File

Before your program closes the remote file, it can gather some statistics about the stations that were communicated with during the session. The following file attributes can be interrogated to obtain statistic information:

- Interrogate the RECEPTIONS attribute to determine the number of messages received from a specific station or from all the stations.
- Interrogate the TRANSMISSIONS attribute to determine the total number of output messages sent to a specific station or to all the stations.

If you want to retain the current station list and its associated RSNs for use the next time you open the remote file, close the file with an ALGOL *CLOSE(<file name>,REWIND)* statement or the regular COBOL *CLOSE* statement.

If you want to use the station list of the system at the time you open the remote file, close the file with a normal ALGOL *CLOSE* statement or a COBOL *CLOSE WITH RELEASE* statement.

Section 7
Using Card Files in a Program

Data Specifications

Purpose

You can use a file whose KIND value is READER (KIND = READER) to read card images from a data specification in a WFL job.

Explanation

A data specification in a WFL job supplies input data in the form of card images to a particular task. The task reads from the data specification as if it were a file whose KIND value is READER (KIND = READER). A task that attempts to read from a card reader file will automatically read from a data specification. If there is no data specification then the program receives a NO FILE condition. Tasks that read from other kinds of files can be file-equated to cause them to read from a data specification instead.

Note: *The default MAXRECSIZE value of a READER file is 14 words (84 characters), but a record contains only 80 characters of valid data. Take this into consideration when using a data specification, since a record in a file generated by CANDE with a FILEKIND value of JOBSYMBOL (FILEKIND = JOBSYMBOL) contains data in columns 1 through 80, spaces in columns 81 and 82, and the sequence number in columns 83 through 90. To avoid getting unwanted information, equate UNITS to CHARACTERS and set the MAXRECSIZE value to 80 when reading the data specification.*

The data images are records of EBCDIC data.

Direct I/O is not allowed for files whose KIND value is READER (KIND = READER).

Refer to "Global Data Specifications" in Section 4 or "Local Data Specifications" in Section 5 of the *WFL Reference Manual* for more information.

When a task tries to open a card reader file, it searches among the data specifications associated with that task for the first unread data specification with the correct file name or no file name.

If the local data specification cannot be located, the task searches for a global data specification with the correct name. If no global data specification with the correct name is located, the task receives a NO FILE condition. When a NO FILE condition occurs

- If the program called OPEN, logical I/O issues the RSVP, NO FILE <filename>(CR).
- If the program called AVAILABLE or RESIDENT, logical I/O returns an error to the program.

Examples

The following WFL example shows the simplest use of a data specification. The program (WALLY)OBJECT/COUNTUP reads data from a single card reader file. In this situation, the data specification does not need to be named, and no file equations are required.

```
RUN (WALLY)OBJECT/COUNTUP;
DATA
  6
? % End of data
```

It is a good idea to give each data specification a title if more than one data specification is being used by the task. This makes it obvious which data specification is being substituted for which input file. The data specification should have the same title as the input file it is replacing in the program, unless the input file has been file-equated to a different title.

In the following WFL example, the program reads the data specification titled TERMIN1 and reads the data specification titled READDAT instead of the input file titled TERMIN2:

```
RUN (WALLY)OBJECT/COUNTTWO;
  FILE TERMIN1(KIND=READER);
  FILE TERMIN2(TITLE=READDAT,KIND=READER);
DATA TERMIN1
  3
  128
? % End of TERMIN1 data
DATA READDAT
  5
? % End of READDAT data
```

The following example shows a task that has more than one data specification. The data specification that OPEN assigns to a file whose KIND value is READER (KIND = READER) depends on several factors. Each time a task attempts to open a file whose KIND value is READER (KIND = READER) the OPEN procedure

1. Searches through the data specifications for the task in the WFL job in the order they are declared

2. Ignores any data specifications that already have been opened or read by the task

3. Ignores any data specification with a file name that does not match the file name of the file to be opened

4. Selects the first unused data specification it encounters that does not have a file name specified, or has a file name that matches the file name of the file to be opened.

```
RUN PROG;
DATA ONE
   <card images>
?
DATA
   <card images>
?
DATA THREE
   <card images>
?
```

If PROG attempts to open a file that has KIND = READER and FILENAME = THREE specified, the system assigns the second data specification (the one with no file name) to the file.

If PROG then attempts to open a file that has KIND = READER and FILENAME = ONE specified, the system would assign the first data specification to the file.

If PROG then attempts to open a file that has KIND = READER and FILENAME = ONE specified, it receives a NO FILE ONE (CR) condition because the first two data specifications have already been used and the third data specification has a file name (THREE) which does not match the file name requested (ONE).

Section 8
Using Operator Display Terminal (ODT) Files

When you want a program to communicate directly with the ODT without using data communications, define an ODT file. When the ODT file is opened, all input from the ODT to the file must be preceded by the group separator (<GS> or <delta>) character represented by the hexadecimal value 1D. To indicate that input is complete, enter *<GS>?END*. Be aware that the automatic display mode (ADM) feature is suppressed on an ODT when your program is communicating with it.

If you are not familiar with basic programming methods, review those methods in Section 2, "Understanding Programming for Files." You can identify all the file attributes that can be used with an ODT file by reviewing Table A-1. You will also find more information about any of the mentioned attributes in the *File Attributes Reference Manual*.

To use an ODT file, perform the following tasks:

1. Assign the KIND attribute a value of ODT.
2. Indicate the external recording mode by setting the EXTMODE attribute value to EBCDIC.
3. Indicate the purpose of the file by using the FILEUSE attribute. If you choose the OUT option, the file is assigned to any scratch ODT unit, unless the program was initiated at an ODT. If the program was initiated at an ODT, the output is directed to the initiating ODT if that unit is a scratch ODT unit. An ODT is considered a scratch unit if it is not labeled. To control where the output is directed, you can set the UNITNO attribute to the desired ODT unit number.

   An ODT becomes a scratch unit when a labeled file assigned to the unit is closed, or an operator uses the CL (Clear) system command. Refer to the *System Operations Guide* for information about using the CL command.

   If you choose the IN or IO option, the file is assigned to a labeled ODT. Set the FILENAME attribute to the file name attached to the ODT. The file name is attached to the ODT by using the LABEL (Label ODT) system command. Refer to the *System Commands Reference Manual* for detailed information on the LABEL command.
4. For most uses, set the FRAMESIZE attribute value to 8 and the BLOCKSTRUCTURE attribute value to EXTERNAL. These values define a character-oriented, variable-length ODT file.

   When a READ operation is requested, the READ operation uses the MAXRECSIZE attribute value to determine the maximum number of characters that should be read.

When the number of input characters transmitted is less than the MAXRECSIZE value, the remaining character spaces are filled with blanks. The I/O result descriptor size field ([47:20]) and the CURRENTRECORDLENGTH attribute value contain the exact number of characters that were read into the buffer.

When a WRITE operation is requested, only the number of characters identified in the WRITE statement are transferred to the ODT.

Section 9
Accessing and Creating Files Using Distributed File Services

You can access and create files on a remote host by using

- Host Services logical I/O
- File Transfer, Access, and Management (FTAM)

If your system is running BNA, Host Services logical I/O enables programs to access and create files on another ClearPath NX server, A Series system, or V Series system that is running BNA. Host Services logical I/O can also be used with ClearPath NX servers and A Series systems across an OSI network.

If your system is running OSI, FTAM allows a disk file to be accessed or created on another OSI host.

When you access or create a file on a remote host, DSS Management determines which service to use. When more than one service can be used, Host Services logical I/O is always given priority over the other possible services, BNA is given priority over OSI and FTAM, and OSI is given priority over FTAM.

WARNING

Both sending and destination hosts must have their SYSOPS LONGFILENAMES option set (see *System Commands Reference Manual*, SYSOPS command) when file transfer can involve file(s) with node names exceeding 17 characters.

Using Host Services Logical I/O

Host Services logical I/O enables you to access and create the following files on a remote host:

- A disk file
- A printer file
- A card reader file
- A remote file
- A tape file

To speed transmission of data between hosts, the data is compressed for all of the preceding peripheral files, except disk files. Files that are disk files are compressed only if one of the following FILEKIND values is specified:

ALGOLSYMBOL	ESPOLSYMBOL	PLISYMBOL
BASICSYMBOL	FORTRANSYMBOL	RPGSYMBOL
BINDERSYMBOL	FORTRAN77SYMBOL	SANSSYMBOL
COBOL74SYMBOL	JOBSYMBOL	SEQDATA
COBOL85SYMBOL	JOVIALSYMBOL	SFORTRANSYMBOL
CSEQDATA	LCOBOLSYMBOL	SORTSYMBOL
DASDLSYMBOL	NDLIISYMBOL	TEXTDATA
DATA	NDLSYMBOL	VFORTRANSYMBOL
DCALGOLSYMBOL	NEWPSYMBOL	XALGOLSYMBOL
DCPSYMBOL	OHNESYMBOL	XFORTRANSYMBOL
DMALGOLSYMBOL	PASCALSYMBOL	

Opening a File Using Host Services Logical I/O

Perform the following tasks, depending on the needs of your program:

1. Identify the name of the remote host where the file resides or is to be created by using the HOSTNAME attribute. The identifier can be 1 to 17 characters long and can contain uppercase alphanumeric characters.
2. Identify the device type of the file by using the KIND attribute.
3. If you want to determine if an open error occurred, check the results of the OPEN statement. If the value BADHSFILERSLT (28) is returned, interrogate the ATTERR and ATTYPE attributes to determine which attribute, if any, caused the open error. An attribute error message is displayed whenever a value of 28 is returned. The following message is displayed if an invalid INTMODE value was specified for a file named F:

```
ATTRIBUTE ERROR: F.INTMODE  DSS DOES NOT SUPPORT THE VALUE OF
THIS ATTRIBUTE
DSS ABORT: FILE F AT BLUE OPEN ERROR:  DSS CANT HANDLE THIS FILE
```

   If a check of the results was not done, the following line of text would be attached to the preceding message:

```
FILE F AT BLUE OPEN ERROR:  DSS CANT HANDLE THIS FILE
```

Examples

The following ALGOL code identifies E as the remote host where a file named THE/FILE is stored on a disk:

```
BEGIN
  FILE F(KIND=DISK,DEPENDENTSPECS=TRUE,FILENAME="THE/FILE.",
         HOSTNAME="E.");
  ARRAY A[0:12];
  LABEL EOF;
  WHILE TRUE DO
    BEGIN
      READ(F,12,A)[EOF];
      .
      .
    END;
  EOF:
    CLOSE(F);
END.
```

The following ALGOL code identifies E as the remote host where a file named OVER/THERE will be created on a disk:

```
BEGIN
  FILE F(KIND=PACK,MAXRECSIZE=14,FILENAME="OVER/THERE.",
         NEWFILE=TRUE,
         HOSTNAME="E.");
  ARRAY A[0:12];
  BOOLEAN DONE;
  DO
    BEGIN
      .
      .
      WRITE(F,12,A);
      .
      .
    END
  UNTIL DONE;
  LOCK(F);
END.
```

For COBOL74 programs, you can identify the HOSTNAME attribute value by using the VALUE OF clause of the file-description entry. To dynamically change the name of the host, use the CHANGE statement.

The following WFL job identifies remote host D as the location of the file name S/PROG that resides on a disk:

```
?BEGIN JOB FOREIGN/COMPILE;
  COMPILE PROG COBOL;
  COBOL FILE CARD(KIND=DISK,FILENAME=S/PROG,HOSTNAME=D);
?END JOB
```

Performing I/O Using Host Services Logical I/O

Programming for any given device type is the same as programming for that device on a local MCP environment system. However, there are some limitations because some logical I/O features are not supported by Host Services logical I/O. Before programming for a given device, review Table 9–25 for information about which file attributes are supported when you are using Host Services logical I/O.

As a programmer using Host Services logical I/O, be aware that a handler task named FILE/HANDLER/<process-hostname> is initiated by Host Services logical I/O on the file host, the host where the file resides. This task performs all I/O subsystem functions on the file. Messages and responses pertaining to the file include the file host name and the mix number of the handler task when displayed at the process host, the host where the program is running. This handler mix number is used in the AT (AT Remote Host) system command. Refer to the *System Commands Reference Manual* for a description of this command.

When both the process host and the file host are ClearPath NX servers or A Series systems, the following restrictions exist:

- The file must use appropriate values for the KIND, INTMODE, EXTMODE, and BLOCKSIZE attributes. Refer to the *File Attributes Reference Manual* for Host Services restrictions for these file attributes.
- A program accessing or creating a file at a remote host must be running under a usercode.
- Direct I/O is not supported.
- Double-byte (16-bit) and mixed multi-byte (mixed 8-bit and 16-bit with FRAMESIZE = 8*)* character sets are not supported.
- For disk files, the FILESTRUCTURE value must be ALIGNED180 or STREAM.
- The ANYSIZEIO file attribute is not supported.
- KEYEDIO is not supported. KEYEDIOII, however, is supported if the KIIOIIHSSUPPORT library is properly installed. Refer to the *KEYEDIOII Reference Manual* for details.
- Relative I/O is not supported.
- Compilers cannot create code files on remote hosts.
- USE routines for tape labels are not supported.
- The ALGOL ERASE statement is not supported.
- Update I/O action is not supported. COBOL programs cannot specify the I-O phrase in an OPEN statement. The value of the UPDATEFILE attribute must be FALSE.
- Buffer sharing is not supported.
- When the AREASIZE file attribute of a disk file being accessed using Host Services Logical I/O is interrogated, the value of AREALENGTH is returned.
- Error results for WRITE statements are reported one buffer later than normal.

- A privileged status, whether from a privileged usercode or from a privileged program, is not carried across the network. As a result, actions that are allowed on the process host—such as creating and removing disk files stored under another usercode, reading and copying files of another user, and invoking certain operating system control interfaces—are not allowed on the file host.
- A BLOCKSTRUCTURE value of VARIABLE is not supported for files on a remote Host Services-capable host. Because of this, any file you declare using the COBOL74 phrase *integer-1 TO* cannot be opened if the file resides on a remote Host Services-capable host. An attempt to open such a file results in an open error. A BLOCKSTRUCTURE value of VARIABLE2 or VARIABLEOFFSET is not supported.
- If the FILESTRUCTURE value is ALIGNED180, a BLOCKSTRUCTURE value of EXTERNAL is supported only for unblocked files.
- Binary I/O is not supported. For example, the ALGOL statement *WRITE (FILE1, *,X)* is not supported.
- Files with partial last records and files created by PLISUPPORT ISAM intrinsics are not supported.
- Host Services logical I/O rejects requests to create nondata files such as system files, compilers, or code files if the host where the file is to be created has the SECOPT HOSTSRESTRICTED attribute set and the user requesting the file creation
  - Is not a privileged user or security administrator
  - Is not assigned an alias to a privileged user or security administrator on the receiving host

  If the user is privileged, the file created is marked as a restricted file. This designation means that the file cannot be executed and can be copied only by a privileged user.

  For information about restrictions on files, units, volumes, and hosts, refer to the *Security Administration Guide*.

To create a file on a host that is not a ClearPath NX server or A Series system, you must make sure the value of the NEWFILE attribute is TRUE. If the value of the NEWFILE attribute is modified to FALSE, Host Services logical I/O searches for an existing file. Note that you cannot use the FA (File Attribute) system command to change the NEWFILE value to TRUE if the program is suspended when no file can be found.

For COBOL74 or COBOL85 programs, the compiler modifies the value of the NEWFILE attribute appropriately on an OPEN statement, so the programmer does not have to consider this situation.

Using FTAM

If your system is running OSI, FTAM enables a disk file to be accessed or created on another system running OSI. The information that is presented here is concerned only with the topic of creating or accessing a file through a user-created program. For information about copying a file, refer to the *Distributed Systems Services Operations Guide*. For information about renaming, inquiring about, or removing a file, refer to the *System Commands Reference Manual*.

FTAM uses document types to specify the following:

- The coded character set that is used
- The maximum string length or maximum record length
- Whether strings or records are of fixed or variable length
- Whether boundaries of strings are to be maintained after data transfer
- The actions that are allowed

The document type you select must be supported on both host systems.

Table 9–1 identifies the four FTAM document types supported by ClearPath NX and A Series hosts for file access and creation. Additionally, the table identifies the types of accesses that are possible.

Table 9–1. FTAM Document Types

Document Type	Description	Possible Access
FTAM-1	A file that contains alphanumeric data and control characters	Sequential
FTAM-2	A file that contains alphanumeric data and control characters	Random or sequential
FTAM-3	A file that contains binary data	Sequential
INTAP-1	A file that contains binary data	Sequential

Before programming your FTAM application, you might want to review FTAM implementation information for the MCP environment found in "FTAM Features in the MCP Environment" later in this section.

The following restrictions should be noted before using FTAM:

- Random I/O for FTAM-1, FTAM-3, or INTAP-1 document types is not supported.
- Direct I/O is not supported.
- Binary I/O is not supported.
- The CRUNCH option of the CLOSE statement is supported, but it is treated as if a LOCK option is invoked.
- KEYEDIO is not supported.
- Relative I/O is not supported.
- C programs cannot access or create files on remote hosts.
- Compilers cannot create code files on remote hosts.
- Double-byte (16-bit) and mixed multi-byte (mixed 8-bit and 16-bit with FRAMESIZE = 8*)* character sets are not supported.

Additionally, you should scan Table 9–25 to determine what disk attributes are supported by FTAM.

Creating a New File on a Remote OSI Host

Before you begin writing a program, you should answer the following questions:

- What is the host name of the system where the file resides or is to be created?
- What is your valid usercode on the remote system?
- What document type do you want the file to be?
- What character set do you want the file to have?
- What value should INTMODE have?
- What I/O actions do you want performed on the file?

The following procedure gives step-by-step instructions about how to create a file on another remote OSI host from an MCP environment system using ALGOL.

1. Specify DISK as the KIND attribute value.
2. If you do not want the system to select the file service for you, specify FTAM as the SERVICE attribute value.
3. Specify the name of the host where the file is to be created by using the HOSTNAME attribute.
4. Use the FILEUSE attribute to specify how the file will be used by the program.

5. Specify a file name that is appropriate for the remote host by using the FILENAME attribute. Enclose the file name in apostrophes (' ') if the file name does not conform to rules of the MCP environment.
6. Specify the maximum size of a record by using the MAXRECSIZE attribute. You can optimize FTAM performance by structuring a file to have a small number of large records, rather than a large number of small records.
7. Specify the structure of the file and the format of the records by using the BLOCKSTRUCTURE attribute.

   Note: *Some systems that are not native MCP systems cannot store an FTAM-1 or FTAM-3 file that has a BLOCKSTRUCTURE value of FIXED or VARIABLE because they do not support the String Significance value of Fixed or Variable for FTAM-1 and FTAM-3 files. If you must create a file on such a system, do not assign a value of FIXED or VARIABLE to BLOCKSTRUCTURE.*

   The following table shows the possible values that you can use:

Document Type	Possible BLOCKSTRUCTURE Value
FTAM-1	FIXED, EXTERNAL, and VARIABLE
FTAM-2	FIXED, EXTERNAL, and VARIABLE
FTAM-3	FIXED, EXTERNAL, and VARIABLE
INTAP-1	FIXED and VARIABLE

8. Specify STREAM as the FILESTRUCTURE attribute value.
9. Set the NEWFILE attribute to TRUE.
10. Indicate a valid usercode, password, and account at the remote host by using the USERCODE attribute.
11. Assign appropriate values to the EXTMODE and INTMODE attributes. Refer to Table 9–20 for the possible EXTMODE character set names that can be used. Note that the usage of a character set is limited by the document type of the file.

    Notes:

    - *If you are creating an FTAM-1 or FTAM-2 file and you want to use the ISO 8859-1 coded character set, EXTMODE must have the value ISOGENERALSTRING or ISOGRAPHICSTRING and each record must contain the appropriate escape sequences. This is necessary to satisfy the ISO Presentation Layer standards that allow GeneralStrings and GraphicStrings to contain multiple character sets and to dynamically switch character sets. Refer to Table 9–21 for possible escape sequences.*
    - *If you are creating an FTAM-1 or FTAM-2 file, the Universal Class parameter values supported by the remote system must be considered when you assign INTMODE and EXTMODE values. Some systems that are not native MCP systems support a subset of the allowable Universal Class parameter values.*

- *FTAM-1 and FTAM-2 files with a universal class of IA5 String or GeneralString can contain control characters from the ISO 646 CO set. This category includes all characters that have hexadecimal values 00 through 1F. Most MCP environment systems software treats these control characters as data. Many systems that are not native MCP systems give special significance to some of these characters when they display or print files. For example, some systems use a carriage return (CR)/line feed (LF) pair to indicate the end of a line of text. When a file is sent from an MCP system to a system that is not an MCP system, the file must contain the control characters necessary for the file to be processed correctly on that system. When an MCP system receives control characters in a file, the characters are stored in the file as data. MCP environment OSI FTAM does not insert or delete control characters when it sends or receives file data.*

12. If you want automatic character set translation to occur, the INTMODE and EXTMODE values must have one of the following combinations. Any other situations where INTMODE and EXTMODE do not match result in an open error.

INTMODE Value	EXTMODE Value
EBCDIC or ASCII	IA5STRING or ISOVISIBLESTRING
ISOVISIBLESTRING	IA5STRING
IA5STRING	ISOVISIBLESTRING

13. Specify a value for the PERMITTEDACTIONS attribute if you do not want this value negotiated for you. Refer to Table 9–22 for possible values for this attribute.

    Be aware that this attribute value can never be changed once the file is created. Thus, if you must control this value, specify the appropriate value before the file is opened.

14. Specify a value for the DOCUMENTTYPE attribute if you do not want this value negotiated for you. Refer to Table 9–1 for possible values for this attribute.

    Be aware that this attribute value can never be changed once the file is created. Thus, if you must control this value, specify the appropriate value before the file is opened.

    If you do not specify a DOCUMENTTYPE attribute value, a value is selected based on the FILEKIND and EXTMODE values of the file according to the algorithm shown in Table 9–2, as long as the document type is supported by both hosts.

Table 9–2. Document Type Selection

EXTMODE	FILEKIND	BLOCKSTRUCTURE	Resulting Document Type
EBCDIC, ASCII, IA5STRING, ISOGENERALSTRING, ISOVISIBLESTRING, ISOGRAPHICSTRING	Symbolic values	FIXED, EXTERNAL	FTAM-1
EBCDIC, ASCII, IA5STRING, ISOGENERALSTRING, ISOVISIBLESTRING, ISOGRAPHICSTRING	Symbolic values	VARIABLE	FTAM-2
EBCDIC, ASCII, IA5STRING, ISOGENERALSTRING, ISOVISIBLESTRING, ISOGRAPHICSTRING	Nonsymbolic values	FIXED, EXTERNAL, VARIABLE	FTAM-3
SINGLE, OCTETSTRING	Any value	FIXED, EXTERNAL, VARIABLE	FTAM-3

If you want to create an INTAP-1 file, specify INTAP1 as the DOCUMENTTYPE value.

If a remote host does not support FTAM-2 documents, an FTAM-2 file is handled as though it were an FTAM-1 document. Some file characteristics are not retained during this process of simplification.

Note: *All files with a FILEKIND of SEQDATA, CSEQDATA, TEXTDATA, or xSYMBOL (where xSYMBOL represents a kind of compiler symbol file, such as ALGOLSYMBOL or NDLSYMBOL) are considered symbolic files. Symbolic files normally contain only displayable characters*

Example

The following is an example of ALGOL code that creates a new file on a FTAM remote host:

```
BEGIN
FILE DK(KIND=DISK,
        HOSTNAME="NODE3.",
        SERVICE=FTAM,
        DOCUMENTTYPE=FTAM3,
        EXTMODE=OCTETSTRING,
        INTMODE=OCTETSTRING,
        FILEUSE=OUT,
        BLOCKSTRUCTURE=FIXED,
        FILENAME="'A:ACCOUNTS.PAY'.",
        FILESTRUCTURE=STREAM,
        NEWFILE=TRUE,
        USERCODE="'VALID'/'USER'.",
        MAXRECSIZE=30,
        FRAMESIZE=8,
  .
  .
  .


OPEN_RESULT:=OPEN (DK,AVAILABLE);
  .
  .
  .
    BEGIN
       .
       .
       .
    IO_RESULT :=WRITE (DK,30, DBUFF);
       .
       .
       .
    END;
 LOCK (DK);
 END.
```

FTAM Parameters Used for Communication When You Create an FTAM-1 File

If you created an FTAM-1 file, each record was mapped to an FTAM data element. Table 9–3 identifies the FTAM parameters that were used to transport some of the file attribute information to the remote host.

Table 9–3. FTAM Parameters Used to Communicate Information in FTAM-1 File Creation

FTAM Parameter and Values	File Attribute and Values in the MCP Environment
Maximum String Length	MAXRECSIZE, FRAMESIZE
MAXRECSIZE	If FRAMESIZE = 8
MAXRECSIZE * 6	If FRAMESIZE = 48
String Significance	BLOCKSTRUCTURE
Fixed	FIXED
Variable	VARIABLE
Not Significant	EXTERNAL
Universal Class	EXTMODE
GeneralString	ISOGENERALSTRING
GraphicString	ISOGRAPHICSTRING
VisibleString	ISOVISIBLESTRING
IA5String	IA5STRING

FTAM Parameters Used for Communication When You Create an FTAM-2 File

If you created an FTAM-2 file, each record was mapped to an FTAM File Access Data Unit (FADU) containing one data element. Table 9–4 identifies the FTAM parameters that were used to transport some of the file attribute information to the remote host.

Table 9–4. FTAM Parameters Used to Communicate Information in FTAM-2 File Creation

FTAM Parameter and Values	File Attribute and Values in the MCP Environment
Maximum String Length	MAXRECSIZE, FRAMESIZE
MAXRECSIZE	If FRAMESIZE = 8
MAXRECSIZE * 6	If FRAMESIZE = 48
String Significance is always set to Not Significant	BLOCKSTRUCTURE is always Variable
Universal Class	EXTMODE
GeneralString	ISOGENERALSTRING
GraphicString	ISOGRAPHICSTRING
VisibleString	ISOVISIBLESTRING
IA5String	IA5STRING

FTAM Parameters Used for Communication When You Create an FTAM-3 File

If you created an FTAM-3 file, each record was mapped to an FTAM data element. Table 9–5 identifies the FTAM parameters that were used to transport some of the file attribute information to the remote host.

Table 9–5. FTAM Parameters Used to Communicate Information in FTAM-3 File Creation

FTAM Parameter and Values	File Attribute and Values in the MCP Environment
Maximum String Length	MAXRECSIZE, FRAMESIZE
MAXRECSIZE	If FRAMESIZE = 8
MAXRECSIZE * 6	If FRAMESIZE = 48
String Significance	BLOCKSTRUCTURE
Fixed	FIXED
Variable	VARIABLE

Table 9–5. FTAM Parameters Used to Communicate Information in FTAM-3 File Creation

FTAM Parameter and Values	File Attribute and Values in the MCP Environment
Not Significant	EXTERNAL

FTAM Parameters Used for Communication When You Create an INTAP-1 File

If you created an INTAP-1 file, each record was mapped to an FTAM record. Table 9–6 identifies the FTAM parameters that were used to transport some of the file attribute information to the remote host.

Table 9–6. FTAM Parameters Used to Communicate Information in INTAP-1 File Creation

FTAM Parameter and Values	File Attribute and Values in the MCP Environment
Maximum Record Length	MAXRECSIZE, FRAMESIZE
MAXRECSIZE	If FRAMESIZE = 8
MAXRECSIZE * 6	If FRAMESIZE = 48
Record Significance	BLOCKSTRUCTURE
Fixed	FIXED
Variable	VARIABLE

Accessing a File on a Remote OSI Host

Before you begin writing a program, you must answer the following questions:

- What is the host name of the system where the file resides?
- What is your valid usercode on the remote system?

The following procedure gives step-by-step instructions about how to access a file on another remote OSI host from an MCP system using ALGOL. The remote OSI host could be another MCP system or a system that is not an MCP system.

1. Set the DEPENDENTSPECS attribute value to TRUE.
2. To ensure that the EXTMODE and INTMODE attribute values are the same. This action eliminates the possibility of translation and sets the DEPENDENTINTMODE attribute value to TRUE. The INTMODE value assumes the EXTMODE value.

Notes:

- *If you are accessing an FTAM-1 or FTAM-2 file and the Universal Class value is GeneralString or GraphicString, escape sequences are contained in the data and identify the coded character set that follows. This is done to satisfy the ISO Presentation Layer standards that allow GeneralStrings and GraphicStrings to contain multiple coded character sets and to dynamically switch coded character sets. Refer to Table 9–21 for possible escape sequences.*
- *FTAM-1 and FTAM-2 files with a universal class of IA5 String or GeneralString can contain control characters from the ISO 646 CO set. This category includes all characters that have hexadecimal values 00 through 1F. Most MCP systems software treats these control characters as data. Many systems that are not MCP systems give special significance to some of these characters when they display or print files. For example, some systems use a carriage return (CR)/line feed (LF) pair to indicate the end of a line of text. When a file is sent from an MCP system to a system that is not an MCP system, the file must contain the control characters necessary for the file to be processed correctly on that system. When an MCP system receives control characters in a file, the characters are stored in the file as data. MCP environment OSI FTAM does not insert or delete control characters when it sends or receives file data.*

If you want automatic character set translation to occur, the INTMODE and EXTMODE values must have one of the following combinations. Any other situations where INTMODE and EXTMODE do not match result in an open error.

INTMODE Value	EXTMODE Value
EBCDIC or ASCII	IA5STRING or ISOVISIBLESTRING
ISOVISIBLESTRING	IA5STRING
IA5STRING	ISOVISIBLESTRING

3. Specify DISK as the KIND attribute value.
4. If you do not want the system to select the file service for you, specify FTAM as the SERVICE attribute value.
5. Specify the name of the host where the file is to be accessed by using the HOSTNAME attribute.
6. Use the FILEUSE attribute to specify if the file will be used for input, output, or both.
7. Specify the file name in the format appropriate for the remote host by using the FILENAME attribute. Enclose the name in apostrophes (' ') if the file name does not conform to MCP environment rules.
8. Specify a valid remote host usercode, password, and account by using the USERCODE attribute. If the usercode, password, and account are not valid, the remote host does not allow you to access the file.

9. When an existing file with any of the document types is opened with an OPEN ATEND, OPEN AVAILATEND, or OPEN EXTEND operation, a serial WRITE operation adds data to the file at the end of the file.

   When an existing file with a DOCUMENTTYPE value of FTAM1, FTAM3, or INTAP1 is opened with any other form of the OPEN statement, a serial WRITE operation at the beginning of the file destroys all previous file contents.

   When an existing file with a DOCUMENTTYPE value of FTAM2 is opened with any other form of the OPEN statement, a serial WRITE operation that overwrites an existing record results in an I/O error.

10. If a file has a DOCUMENTTYPE of FTAM1, FTAM3, or INTAP1, switching from a READ operation to a WRITE operation is allowed only at the beginning or the end of the file. If FTAM is unable to determine that the current record position of the file is the first record or is beyond the last record, the program is discontinued if an attempt is made to write to the file.

Example

The following is an example of ALGOL code that accesses a file named TEST/DATA/OUT on an FTAM remote host named NODE3:

```
  BEGIN
  FILE DK(KIND=DISK,
          HOSTNAME="NODE3.",
          SERVICE=FTAM,
          DEPENDENTSPECS=TRUE,
          DEPENDENTINTMODE=TRUE,
          FILEUSE=IN,
          FILENAME ="'A:ACCOUNTS.PAY'.",
          USERCODE="'VALID'/'USER'."),
    .
    .
    .


OPEN_RESULT :=OPEN (DK,AVAILABLE);
    .
    .
    .

IO_RESULT :=READ (DK,REC_SIZE,DBUFF);
    .
    .
    .

END.
```

FTAM Parameters Used for Communication When You Access an FTAM-1 File

When you access an FTAM-1 file, an FTAM data element is used to transport each record to and from the remote OSI host. Table 9–7 identifies FTAM attributes and parameters that are used to transport information about the file to and from the remote OSI host.

Table 9–7. FTAM Parameters Used to Communicate Information in FTAM-1 File Access

FTAM Attribute or Parameter and Values	File Attribute and Values in the MCP Environment
Filename attribute	TITLE
Document Type Name parameter	DOCUMENTTYPE
Permitted Actions attribute	PERMITTEDACTIONS
String Significance parameter	BLOCKSTRUCTURE
Not Significant	EXTERNAL
Variable	VARIABLE
Fixed	FIXED
Universal Class parameter	EXTMODE
GeneralString	ISOGENERALSTRING
GraphicString	ISOGRAPHICSTRING
VisibleString	ISOVISIBLESTRING
IA5String	IA5STRING
Maximum String Length Parameter	MAXRECSIZE
Parameter Available	Maximum string length parameter value
Parameter Not Available	9995, if the String Significance parameter is Variable or the Universal Class parameter is GeneralString or GraphicString 64000, if the String Significance parameter is Not Significant and the Universal Class parameter is IA5String or VisibleString

FTAM Parameters Used for Communication When You Access an FTAM-2 File

If the file is an FTAM-2 file, an FTAM FADU containing one data element is used to transport each record to and from the remote OSI host. Table 9–8 identifies FTAM attributes and parameters that are used to transport information about the file to and from the remote OSI host. The String Significance parameter always has a value of Not Significant.

Table 9–8. FTAM Parameters Used to Communicate Information in FTAM-2 File Access

FTAM Attribute or Parameter and Values	File Attribute and Values in the MCP Environment
Filename attribute	TITLE
Document Type Name parameter	DOCUMENTTYPE
Permitted Actions attribute	PERMITTEDACTIONS
String Significance parameter	BLOCKSTRUCTURE
Not Significant	EXTERNAL
Universal Class parameter	EXTMODE
GeneralString	ISOGENERALSTRING
GraphicString	ISOGRAPHICSTRING
VisibleString	ISOVISIBLESTRING
IA5String	IA5STRING
Maximum String Length Parameter	MAXRECSIZE
Parameter Available	Maximum string length parameter value that does not exceed 9995
Parameter Not Available	9995

FTAM Parameters Used for Communication When You Access an FTAM-3 File

If the file is an FTAM-3 file, an FTAM data element is used to transport each record to and from the remote OSI host. Table 9–9 identifies FTAM attributes and parameters that are used to transport information about the file to and from the remote OSI host.

Table 9–9. FTAM Parameters Used to Communicate Information in FTAM-3 File Access

FTAM Attribute or Parameter and Values	File Attribute and Values in the MCP Environment
Filename attribute	TITLE
Document Type Name parameter	DOCUMENTTYPE
Permitted Actions attribute	PERMITTEDACTIONS
String Significance parameter	BLOCKSTRUCTURE
Fixed	FIXED
Variable	VARIABLE
Not Significant	EXTERNAL
Maximum String length parameter	MAXRECSIZE
Parameter available	Maximum string length parameter value
Parameter not available	9995, if the String Significance parameter is Variable 64000, if the String Significance parameter is Not Significant

FTAM Parameters Used for Communication When You Access an INTAP-1 File

If the file is an INTAP-1 file, an FTAM record is used to transport each record to and from the remote OSI host. Table 9–10 identifies FTAM attributes and parameters that are used to transport information about the file to and from the remote OSI host.

Table 9–10. FTAM Parameters Used to Communicate Information in INTAP-1 File Access

FTAM Attribute or Parameter and Values	File Attribute and Values in the MCP Environment
Filename attribute	TITLE
Document Type Name parameter	DOCUMENTTYPE
Permitted Actions attribute	PERMITTEDACTIONS
Record Significance parameter	BLOCKSTRUCTURE
Fixed	FIXED
Variable	VARIABLE
Maximum Record length parameter	MAXRECSIZE
Parameter available	Maximum record length parameter value
Parameter not available	9995, if the Record Significance parameter is Variable Record

Creating a File on the Local System to Be Accessed through FTAM

To create a file on the local system that will be accessed through FTAM from a remote host, you need to first answer the following questions:

- What document type is needed?
- What character string is expected?

The following procedure identifies the steps that must be taken to create a file that can be accessed by an FTAM remote host.

1. Assign the KIND attribute a value of DISK.
2. Before opening the file, you must assign the EXTMODE attribute one of the following mnemonic values:
   - EBCDIC
   - ASCII
   - IA5STRING

- ISOGENERALSTRING
- ISOGRAPHICSTRING
- ISOVISIBLESTRING
- OCTETSTRING (for FTAM-3 and INTAP-1 files)
- SINGLE (for FTAM-3 and INTAP-1 files)

Notes:

- *If you are creating an FTAM-1 or FTAM-2 file and you want to use the ISO 8859-1 coded character set, EXTMODE must have the value ISOGENERALSTRING or ISOGRAPHICSTRING and each record must contain the appropriate escape sequences. This is necessary to satisfy the ISO Presentation Layer standards that allow GeneralStrings and GraphicStrings to contain multiple character sets and to dynamically switch character sets. Refer to Table 9–21 for possible escape sequences.*
- *Some systems that are not MCP systems cannot access an FTAM-1 or FTAM-2 file that has an EXTMODE value of ASCII, EBCDIC, IA5STRING, ISOGENERALSTRING, ISOGRAPHICSTRING, or ISOVISIBLESTRING because not all the values of the Universal Class parameter are supported. If the EXTMODE value is ASCII, EBCDIC, or IA5STRING, and the remote system does not support the Universal Class parameter value of IA5STRING and does support VisibleString, you can use the DUMPALL utility to change the EXTMODE attribute value to ISOVISIBLESTRING before the file is accessed.*
- *FTAM-1 and FTAM-2 files with a universal class of IA5 String or GeneralString can contain control characters from the ISO 646 CO set. This category includes all characters that have hexadecimal values 00 through 1F. Most MCP environment systems software treats these control characters as data. Many systems that are not MCP systems give special significance to some of these characters when they display or print files. For example, some systems use a carriage return (CR)/line feed (LF) pair to indicate the end of a line of text. When a file is sent from an MCP system to a system that is not an MCP system, the file must contain the control characters necessary for the file to be processed correctly on that system. When an MCP system receives control characters in a file, the characters are stored in the file as data. MCP environment OSI FTAM does not insert or delete control characters when it sends or receives file data.*

3. Set the DOCUMENTTYPE and PERMITTEDACTIONS attribute values for your specific needs. If you do not set these values, the values are set to the defaults shown in Table 9–2 when the file is opened by an FTAM user. The default values are not stored with the file. Refer to Tables 9–1 and 9–22 for possible values for these attributes.

   If you want the file to be accessed as an INTAP-1 file, specify INTAP1 as the DOCUMENTTYPE attribute value.

4. Assign the BLOCKSTRUCTURE attribute an appropriate value from the following table. You can optimize FTAM performance by structuring a file to have a small number of large records, rather than a large number of small records.

   Note: *Some systems that are not MCP systems cannot access an FTAM-1 or FTAM-3 file that has a BLOCKSTRUCTURE value of FIXED or VARIABLE, because they do not support the String Significance value of Fixed or Variable for FTAM-1 and FTAM-3 files. If such a system must access such a file, you can use the DUMPALL utility to change the value of BLOCKSTRUCTURE from FIXED or VARIABLE to EXTERNAL prior to accessing the file.*

Document Type	Possible BLOCKSTRUCTURE Value
FTAM-1	FIXED, EXTERNAL, and VARIABLE
FTAM-2	FIXED, EXTERNAL, and VARIABLE
FTAM-3	FIXED, EXTERNAL, and VARIABLE
INTAP-1	FIXED and VARIABLE

5. Assign the FILEORGANIZATION attribute a value of NOTRESTRICTED.
6. Assign any of the FILESTRUCTURE attribute values; ALIGNED180, BLOCKED, or STREAM.

   Note: *If an FTAM user accesses the created file later and does not specify a usercode with the file name, such as (usercode)<file name>, the MCP environment FTAM software searches for the file in the usercode directory of the accessing user, which might be a LOCALALIAS usercode. If this search is unsuccessful, the FTAM software searches for the file among the files without a usercode.*

How File Attribute Values Are Passed for an FTAM-1 File

If an FTAM-1 file is accessed by a remote OSI host, each record is mapped to an FTAM data element. Table 9–11 identifies how the file attribute values in the MCP environment are mapped to the FTAM parameters.

Table 9–11. File Attribute Values Passed When an FTAM-1 File Is Accessed

File Attribute and Values in the MCP Environment	FTAM Parameter and Values
MAXRECSIZE, FRAMESIZE	Maximum String Length
If FRAMESIZE = 8	MAXRECSIZE
If FRAMESIZE = 48	MAXRECSIZE * 6
BLOCKSTRUCTURE	String Significance
FIXED	Fixed
VARIABLE	Variable
EXTERNAL	Not Significant
EXTMODE	Universal Class
ISOGENERALSTRING	GeneralString
ISOGRAPHICSTRING	GraphicString
ISOVISIBLESTRING	VisibleString
IA5STRING, ASCII, EBCDIC	IA5String

How File Attribute Values are Passed for an FTAM-2 File

If an FTAM-2 file is accessed by a remote OSI host, each record is mapped to an FTAM FADU containing one data element. Table 9–12 identifies how the file attribute values in the MCP environment are mapped to the FTAM parameters.

Table 9–12. File Attribute Values Passed When an FTAM-2 File Is Accessed

File Attribute and Values in the MCP Environment	FTAM Parameter and Values
MAXRECSIZE, FRAMESIZE	Maximum String Length
If FRAMESIZE = 8	MAXRECSIZE
If FRAMESIZE = 48	MAXRECSIZE * 6
BLOCKSTRUCTURE is always variable	String Significance is always Not Significant
EXTMODE	Universal Class
ISOGENERALSTRING	GeneralString
ISOGRAPHICSTRING	GraphicString
ISOVISIBLESTRING	VisibleString
IA5STRING, ASCII, EBCDIC	IA5String

How File Attribute Values are Passed for an FTAM-3 File

If an FTAM-3 file is accessed by a remote OSI host, each record is mapped to an FTAM data element. Table 9–13 identifies how the file attribute values in the MCP environment are mapped to the FTAM parameters.

Table 9–13. File Attribute Values Passed When an FTAM-3 File Is Accessed

File Attribute and Values in the MCP Environment	FTAM Parameter and Values
MAXRECSIZE, FRAMESIZE	Maximum String Length
If FRAMESIZE = 8	MAXRECSIZE
If FRAMESIZE = 48	MAXRECSIZE * 6
BLOCKSTRUCTURE	String Significance
FIXED	Fixed
VARIABLE	Variable
EXTERNAL	Not Significant

How File Attribute Values are Passed for an INTAP-1 File

If an INTAP-1 file is accessed by a remote OSI host, each record is mapped to an FTAM record. Table 9–14 identifies how the file attribute values in the MCP environment are mapped to the FTAM parameters.

Table 9–14. File Attribute Values Passed When an INTAP-1 File Is Accessed

File Attribute and Values in the MCP Environment	FTAM Parameter and Values
MAXRECSIZE, FRAMESIZE	Maximum Record Length
If FRAMESIZE = 8	MAXRECSIZE
If FRAMESIZE = 48	MAXRECSIZE * 6
BLOCKSTRUCTURE	Record Significance
FIXED	Fixed
VARIABLE	Variable

Accessing a File Created through FTAM on the Local System

You can use any programming techniques that are possible for a disk file that has a FILESTRUCTURE attribute value of STREAM if the file was created on the local host by a program on a remote OSI host, or copied from a remote OSI host to the local host.

Note: *When FTAM is transferring a file to an MCP system or is creating a new file on an MCP system, the file is entered in the directory at the time the file is opened, instead of when the file is closed. As a result, if an FTAM session is aborted, that is if an operator terminated the job or the network failed, a partial file might remain on the disk. This behavior is necessary in order to conform to the requirements of the FTAM International Standard.*

File Attribute Values Received for an FTAM-1 File

If the file is an FTAM-1 file, a record was created from each FTAM data element. Table 9–15 shows how the file attributes in the MCP environment obtained their current values from the FTAM attributes and parameters sent by the remote OSI host. FTAM document type parameters are maintained in the disk file header for future use by FTAM, in the event the file is transferred or accessed. Additionally, the FILEORGANIZATION attribute was set to NOTRESTRICTED, the FILESTRUCTURE attribute was set to STREAM, and the FRAMESIZE attribute was set to 8.

Notes:

- *FTAM-1 files with a universal class of IA5 String or GeneralString can contain control characters from the ISO 646 CO set. This category includes all characters that have hexadecimal values 00 through 1F. Most MCP environment system software treats these special characters as data. Many non-A Series systems give special significance to some of these characters when they display or print files. For example, some systems use a carriage return (CR)/line feed (LF) pair to indicate the end of a line of text. When an MCP system receives control characters in a file, they are stored in the file as data. MCP environment OSI FTAM does not insert or delete control characters when it receives file data.*
- *If the Universal Class value is GeneralString or GraphicString, escape sequences are stored as data and identify the character set that follow. This is done to satisfy the ISO Presentation Layer standards that allow GeneralStrings and GraphicStrings to contain multiple character sets and to dynamically switch character sets. Refer to Table 9–21 for possible escape sequences.*

Table 9–15. File Attribute Values for an FTAM-1 File Created by a Remote Host

File Attribute and Values in the MCP Environment	FTAM Attribute or Parameter
TITLE	Filename attribute specified by remote host initiator
DOCUMENTTYPE	Document Type Name parameter specified by remote host initiator
PERMITTEDACTIONS	Permitted Actions attribute
BLOCKSTRUCTURE	Depends on the values of the String Significance and Universal Class parameters
EXTERNAL	If String Significance is Not Significant and Universal Class is IA5String or VisibleString
VARIABLE	If Universal Class is IA5String or VisibleString and String Significance is Variable, or if Universal Class is GeneralString or GraphicString
FIXED	If String Significance is Fixed and Universal class is IA5String or VisibleString
EXTMODE	Universal Class parameter
ISOGENERALSTRING	GeneralString
ISOGRAPHICSTRING	GraphicString
ISOVISIIBLESTRING	VisibleString
IA5STRING	IA5String
MAXRECSIZE	

Table 9–15. File Attribute Values for an FTAM-1 File Created by a Remote Host

File Attribute and Values in the MCP Environment	FTAM Attribute or Parameter
9995	If Universal Class is GeneralString or GraphicString or if the String Significance is Variable and the maximum string length parameter is not available
Maximum String Length parameter value	If Universal Class is IA5String or VisibleString and the maximum string length parameter is available
64000	If Universal Class is IA5String or VisibleString, the String Significance is Not Significant, and the maximum string length parameter is not available

File Attribute Values Received for an FTAM-2 File

If the file is an FTAM-2 file, a record was created from each FTAM FADU and the boundaries between the data elements within the FADU were not maintained. Table 9–16 shows how the file attributes in the MCP environment obtained their current values from the FTAM attributes and parameters sent by the remote OSI host. FTAM document type parameters are maintained in the disk file header for future use by FTAM, in the event the file is transferred or accessed. Additionally, the FILEORGANIZATION attribute was set to NOTRESTRICTED, the FILESTRUCTURE attribute was set to STREAM, the BLOCKSTRUCTURE attribute was set to VARIABLE, and the FRAMESIZE attribute was set to 8.

Notes:

- *FTAM-2 files with a universal class of IA5 String or GeneralString can contain control characters from the ISO 646 CO set. This category includes all characters that have hexadecimal values 00 through 1F. Most MCP environment system software treats these special characters as data. Many non- Series systems give special significance to some of these characters when they display or print files. For example, some systems use a carriage return (CR)/line feed (LF) pair to indicate the end of a line of text. When an MCP system receives control characters in a file, they are stored in the file as data. MCP environment OSI FTAM does not insert or delete control characters when it receives file data.*
- *If the Universal Class value is GeneralString or GraphicString, escape sequences are stored as data and identify the character set that follow. This is done to satisfy the ISO Presentation Layer standards that allow GeneralStrings and GraphicStrings to contain multiple character sets and to dynamically switch character sets. Refer to Table 9–21 for possible escape sequences.*

Table 9–16. File Attribute Values for an FTAM-2 File Created by a Remote Host

File Attribute and Values in the MCP Environment	FTAM Attribute or Parameter
TITLE	Filename attribute specified by the remote host initiator
DOCUMENTTYPE	Document Type Name parameter specified by the remote host initiator
PERMITTEDACTIONS	Permitted Actions attribute
EXTMODE	Universal Class parameter
ISOGENERALSTRING	GeneralString
ISOGRAPHICSTRING	GraphicString
ISOVISIBLESTRING	VisibleString
IA5STRING	IA5String
MAXRECSIZE	
9995	

File Attribute Values Received for an FTAM-3 File

If the file is an FTAM-3 file, a record was created from each FTAM data element. Table 9–17 shows how the MCP environment file attributes obtained their current values from the FTAM attributes and parameters sent by the remote OSI host. FTAM document type parameters are maintained in the disk file header for future use by FTAM, in the event the file is transferred or accessed. Additionally, the FILEORGANIZATION attribute was set to NOTRESTRICTED, the FILESTRUCTURE attribute was set to STREAM, the EXTMODE attribute was set to OCTETSTRING, and the FRAMESIZE attribute was set to 8.

Table 9–17. File Attribute Values for an FTAM-3 File Created by a Remote Host

File Attribute and Values in the MCP Environment	FTAM Attribute or Parameter
TITLE	Filename attribute specified by remote host initiator
DOCUMENTTYPE	Document Type Name parameter specified by the remote host initiator
PERMITTEDACTIONS	Permitted Actions attribute
BLOCKSTRUCTURE	String Significance parameter
FIXED	Fixed

Table 9–17. File Attribute Values for an FTAM-3 File Created by a Remote Host

VARIABLE	Variable
EXTERNAL	Not Significant
MAXRECSIZE	
9995	If the String Significance parameter is Variable and the Maximum String Length parameter is not available
64000	If the String Significance parameter is Not Significant and the Maximum String Length parameter is not available
Maximum String Length parameter value	If the Maximum String Length parameter is available

File Attribute Values Received for an INTAP-1 File

If the file is an INTAP-1 file, a record was created from each FTAM record. Table 9–18 shows how the file attributes in the MCP environment obtained their current values from the FTAM attributes and parameters sent by the remote OSI host. FTAM document type parameters are maintained in the disk file header for future use by FTAM, in the event the file is transferred or accessed. Additionally, the FILEORGANIZATION attribute was set to NOTRESTRICTED, the FILESTRUCTURE attribute was set to STREAM, the EXTMODE attribute was set to OCTETSTRING, and the FRAMESIZE attribute was set to 8.

Table 9–18. File Attribute Values for an INTAP-1 File Created by a Remote Host

File Attribute and Values in the MCP Environment	FTAM Attribute or Parameter
TITLE	Filename attribute specified by remote host initiator
DOCUMENTTYPE	Document Type Name parameter specified by the remote host initiator
PERMITTEDACTIONS	Permitted Actions attribute
BLOCKSTRUCTURE	Record Significance parameter
FIXED	Fixed
VARIABLE	Variable
MAXRECSIZE	
9995	If the Record Significance parameter is Variable and the Maximum Record Length parameter is not available

Table 9–18. File Attribute Values for an INTAP-1 File Created by a Remote Host

File Attribute and Values in the MCP Environment	FTAM Attribute or Parameter
64000	If the Record Significance parameter is Fixed and the Maximum Record Length parameter is not available
Maximum Record Length parameter value	If the Maximum Record Length parameter is available

FTAM Features in the MCP Environment

The following information summarizes some important facts about the implementation of FTAM in the MCP environment.

Mapping File Attributes in the MCP Environment to FTAM File Attributes

Table 9–19 lists the FTAM file attributes and their file attribute equivalents in the MCP environment.

Table 9–19. FTAM File Attribute Equivalents

FTAM File Attribute	File Attribute in the MCP Environment
Filename	FILENAME/TITLE
Permitted Actions	PERMITTEDACTIONS
Document Type Name parameter of Contents Type	DOCUMENTTYPE
Storage Account	Not supported
Date and Time of Creation	CREATIONDATE and CREATIONTIME
Date and Time of Last Modification	ALTERDATE and ALTERTIME
Date and Time of Last Read Access	Not supported
Date and Time of Last Attribute Modification	Not supported
Identity of Creator	Not supported
Identity of Last Modifier	Not supported
Identity of Last Reader	Not supported
Identity of Last Attribute Modifier	Not supported

Table 9–19. FTAM File Attribute Equivalents

FTAM File Attribute	File Attribute in the MCP Environment
File Availability	Supported internally
Filesize	FILELENGTH
Future Filesize	Not supported
Access Control	Not supported
Legal Qualifications	Not supported
Private Use	Not supported

Identifying Coded Character Sets

Table 9–20 identifies the five coded character sets that are available, the ISO coded character sets that can be used, the document type that can use the character set, and whether or not escape sequences are permitted. Double-byte (16-bit) and mixed multi-byte (mixed 8-bit and 16-bit with FRAMESIZE = 8*)* character sets are not supported.

In Tables 9–20 and 9–21, C0 includes the hexadecimal values 4"00" through 4"1F", G0 includes the hexadecimal values 4"21" through 4"7E", and G1 includes the hexadecimal values 4"A0" through 4"FF".

Table 9–20. Possible Character Sets

Character Set Name	ISO Coded Character Set	Document Type	Escape Sequence
IA5STRING Refer to Figure 9--2 for a description of this character set	A string of 8-bit frames containing the ISO646 G0 and C0 sets (Refer to Figure 9--2 for a description of this character set).	FTAM-1 and FTAM-2	Not permitted
ISOGENERALSTRING Refer to Figure 9--2 for a description of this character set	A string of 8-bit frames containing the ISO646 G0 and C0 sets, (Refer to Figure 9--2 for a description of this character set) by default, or the ISO8859-1 G0 and G1 sets (Refer to Figure 9--2 for a description of this character set).	FTAM-1 and FTAM-2	Permitted

Table 9–20. Possible Character Sets

Character Set Name	ISO Coded Character Set	Document Type	Escape Sequence
ISOGRAPHICSTRING	A string of 8-bit frames containing the ISO646 G0 set, (Refer to Figure 9--2 for a description of this character set) by default, or the ISO8859-1 G0 and G1 sets (Refer to Figure 9--2 for a description of this character set).	FTAM-1 and FTAM-2	Permitted
ISOVISIBLESTRING	A string of 8-bit frames containing the ISO646 G0 set (Refer to Figure 9--2 for a description of this character set).	FTAM-1 and FTAM-2	Not permitted
OCTETSTRING	A string of 8-bit frames each containing any binary value from Hex 00 to Hex FF.	FTAM-3 and INTAP-1	Not permitted

Escape sequences allow a program to switch from one ISO coded character set to another in the same record. Table 9–21 identifies the escape sequences that are used to introduce each coded character set.

Table 9–21. Possible Escape Sequences

Coded Character Set	Hex Representation of Escape Sequence
ISO 646 C0	"1B" "21" "40"
ISO 646 G0	"1B" "28" "40"
ISO 8859-1 G0	"1B" "28" "42"
ISO 8859-1 G1	"1B" "2D" "41"

Figure 9–1 presents the ISO 646 coded character set.

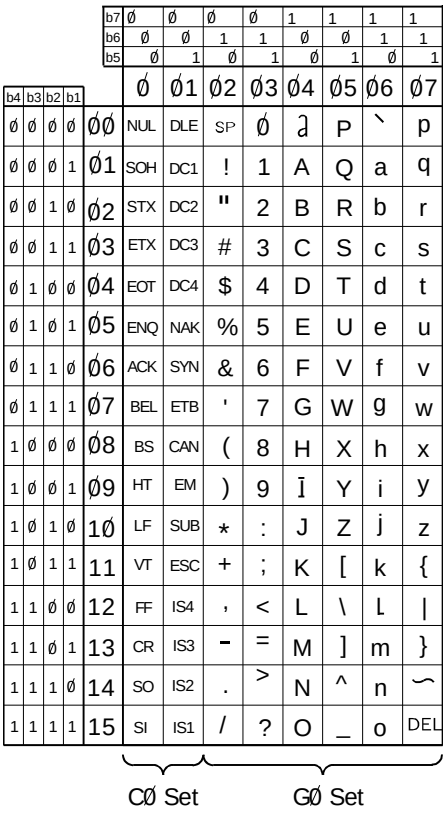


Figure 9–1. ISO 646 Coded Character Set

Figure 9–2 presents the ISO 8859-1 coded character set.

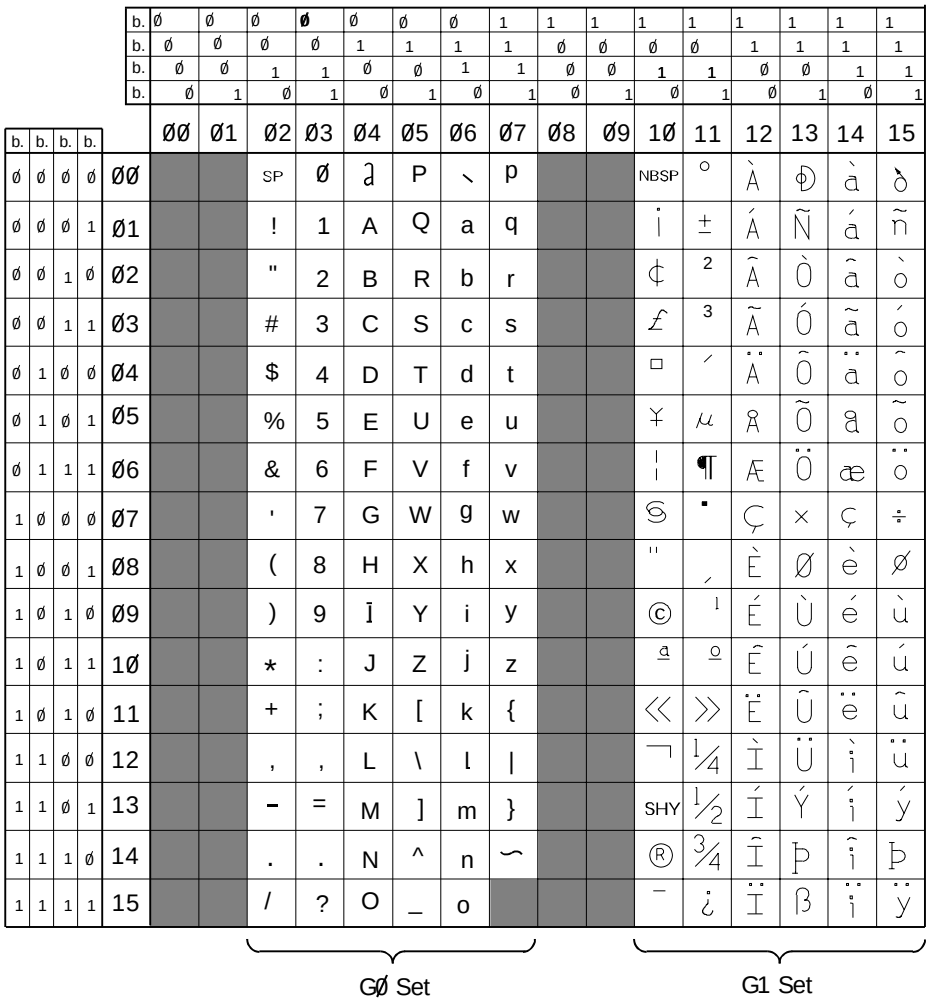


Figure 9–2. ISO 8859-1 Coded Character Set

Specifying I/O Actions with the PERMITTEDACTIONS Attribute

The PERMITTEDACTIONS attribute is used to specify the I/O actions that can be performed in a file through FTAM. The possible I/O actions that can be specified are listed in Table 9–22.

Table 9–22. Possible PERMITTEDACTIONS Values

Action	Valid Document Type	PERMITTEDACTIONS Attribute Field
Read a record.	FTAM-2	[00:01]
Read an entire file.	FTAM-1, FTAM-3, and INTAP-1	[00:01]
Add records at the end of the file.	FTAM-2	[01:01]
Replace the contents of the file.	FTAM-1, FTAM-3, and INTAP-1	[02:01]
Add new data at the end of the file.	FTAM-1, FTAM-3, and INTAP-1	[03:01]
Delete the contents of the file.	All	[04:01]
Interrogate the attributes.	All	[05:01]
Modify the attributes.	All	[06:01]
Delete the file.	All	[07:01]
Traverse the file from beginning to end by using one of the following record identities: • Begin • First • Next • Last • End	FTAM-2	[08:01]
Traverse the file from end to beginning by using one of the following record identities: • Begin • First • Previous • Last • End	FTAM-2	[09:01]
Traverse the file randomly by using one of the following record identities: • Current • Node number	FTAM-2	[10:01]

Controlling the Concurrency-Control Parameter Information

When your program is accessing a file on a remote system, it can control whether the Concurrency-Control parameter of the F-Create and F-Select Application Protocol Data Units (APDUs) are sent by specifying an EXCLUSIVE file attribute value. The Concurrency-Control parameter of the F-Open APDU is never sent. If the program does not specify an EXCLUSIVE value, no information is sent.

If the program specifies an EXCLUSIVE attribute, the values of the DOCUMENTTYPE, EXCLUSIVE, and FILEUSE attributes determine the information sent in the Concurrency-Control parameter. Table 9–23 shows what Concurrency-Control information is sent if the EXCLUSIVE value is FALSE.

Table 9–23. Concurrency-Control Parameter Information Sent When the EXCLUSIVE File Attribute Is FALSE

FILEUSE Value	DOCUMENTTYPE Value	Concurrency-Control Information Sent
Not specified	Not specified	Read Attributes, Change Attributes, and Delete File are set to SHARED.
IN	Not specified	Read is set to SHARED. Insert, Replace, Extend, and Erase are set to NOT REQUIRED.
OUT	Not specified	Erase is set to SHARED. Read is set to NOT REQUIRED.
OUT	FTAM1 or FTAM3	Replace and Extend are set to SHARED. Insert is set to NOT REQUIRED.
OUT	FTAM2	Replace and Extend are set to NOT REQUIRED. Insert is set to SHARED.
IO	Not specified	Read and Erase are set to SHARED.
IO	FTAM1 or FTAM3	Replace and Extend are set to SHARED. Insert is set to NOT REQUIRED.
IO	FTAM2	Replace and Extend are set to NOT REQUIRED. Insert is set to SHARED.

Table 9–24 shows what Concurrency-Control information is sent if the EXCLUSIVE value is TRUE.

Table 9–24. Concurrency-Control Parameter Information Sent When the EXCLUSIVE File Attribute Is TRUE

FILEUSE Value	DOCUMENTTYPE Value	Concurrency-Control Information Sent
Not specified	Not specified	Read Attributes, Change Attributes, and Delete File are set to EXCLUSIVE.
IN	Not specified	Read is set to EXCLUSIVE. Insert, Replace, Extend, and Erase are set to NO ACCESS.
OUT	Not specified	Erase is set to EXCLUSIVE. Read is set to NO ACCESS.
OUT	FTAM1 or FTAM3	Replace and Extend are set to EXCLUSIVE. Insert is set to NO ACCESS.
OUT	FTAM2	Replace and Extend are set to NO ACCESS. Insert is set to EXCLUSIVE.
IO	Not specified	Read and Erase are set to EXCLUSIVE.
IO	FTAM1 or FTAM3	Replace and Extend are set to EXCLUSIVE. Insert is set to NO ACCESS.
IO	FTAM2	Replace and Extend are set to NO ACCESS. Insert is set to EXCLUSIVE.

When MCP environment FTAM is the responding host, all valid values of the Concurrency-Control parameter are supported for the F-Create, F-Select, and F-Open PDUs.

Handling Waiting When No File Is Found

When a file is not present, the initiating host waits rather than the responding host.

Handling String Parameters

The FTAM protocol contains four string parameters that require special handling when they are received by an MCP system. This special handling ensures that the contents of the string parameters are properly mapped to the corresponding MCP environment attributes. The following table names the string parameters and indicates to which attributes they are mapped.

String Parameter	Attribute Used
Initiator-Identify	USERCODE
Account	Not used
Filestore-Password	PASSWORD
Filename	TITLE

Initiator-Identify, Account, and Filename are all encoded in the ISOGRAPHICSTRING character set. Filestore-Password can be encoded by systems other than MCP systems either as ISOGRAPHICSTRING or OCTETSTRING. The actual character set used for ISOGRAPHICSTRING encoding is the ISO 646 coded character set.

MCP environment FTAM translates incoming ISOGRAPHICSTRING-encoded parameters to EBCDIC. All lowercase characters that are not enclosed in quotation marks (") are converted to uppercase characters.

If MCP environment FTAM receives a Filestore-Password string parameter encoded in OCTETSTRING, FTAM passes the data to the MCP without translating the data or converting the characters to uppercase characters. If the characters in the character string are not EBCDIC, a security violation results.

MCP environment FTAM sends all four parameters as ISOGRAPHICSTRING characters. You can prevent FTAM from changing all the characters to uppercase characters by enclosing the characters in apostrophes (' ') or quotation marks ('' '') when you specify the FILENAME and USERCODE attributes. If you use apostrophes, no changes are made, and the apostrophes are stripped from the character string before the data is sent. If you use quotation marks, no changes are made, but the quotation marks are not stripped from the character string before the data is sent.

Identifying Supported File Attributes

Table 9–25 identifies the attributes that can be used by Host Services logical I/O and FTAM.

Table 9–25. Host Services Logical I/O and FTAM File Attributes

Attribute	Host Services	FTAM
ACCESSDATE	Supported	Not supported
ACCESSDATEUT	Supported	Not supported
ACCESSTZ	Supported	Not supported
ADAPTABLE	Supported	Not supported
AFTER	Supported	Not supported
ALIGNFILE	Supported	Not supported
ALIGNMENT	Supported	Not supported
ALTERDATE	Supported	Supported
ALTERDATEUT	Supported	Supported
ALTERNATEGROUPS	Supported	Not supported
ALTERTIME	Supported	Supported
ALTERTIMEUT	Supported	Supported
ALTERTZ	Supported	Not supported
APPEND	Supported	Not supported
AREAADDRESS	Not supported	Not supported
AREAALLOCATED	Supported	Not supported
AREALENGTH	Supported	Not supported
AREAS	Supported	Not supported
AREASECTORS	Supported	Not supported
AREASINUSE	Not supported	Not supported
AREASIZE	Restricted usage	Not supported
ATTERR	Supported	Supported
ATTMODIFYDATE	Supported	Not supported
ATTMODIFYDATEUT	Supported	Not supported
ATTMODIFYTIME	Supported	Not supported
ATTMODIFYTIMEUT	Supported	Not supported
ATTMODIFYTZ	Supported	Not supported
ATTVALUE	Supported	Supported

Table 9–25. Host Services Logical I/O and FTAM File Attributes

Attribute	Host Services	FTAM
ATTYPE	Not supported	Supported
AUTOUNLOAD	Supported	Not supported
AVAILABLE	Supported	Supported
BACKUPDATE	Supported	Not supported
BACKUPDATEUT	Supported	Not supported
BACKUPKIND	Supported	Not supported
BACKUPTIME	Supported	Not supported
BACKUPTIMEUT	Supported	Not supported
BACKUPTZ	Supported	Not supported
BLOCK	Supported	Not supported
BLOCKSIZE	Restricted values	Not supported
BLOCKSTRUCTURE	Restricted values	Restricted values
BUFFERS	Supported	Not supported
BUFFERSIZE	Supported	Not supported
CARRIAGECONTROL	Supported	Not supported
CCSVERSION	Supported	Not supported
CENSUS	Supported	Not supported
CHECKPOINT	Supported	Not supported
CLEARAREAS	Supported	Not supported
COMPRESSING	Supported	Not supported
COMPRESSIONCONTROL	Supported	Not supported
COMPRESSIONREQUESTED	Supported	Not supported
COPYDESTDATE	Supported	Not supported
COPYDESTDATEUT	Supported	Not supported
COPYDESTTIME	Supported	Not supported
COPYDESTTIMEUT	Supported	Not supported
COPYDESTTZ	Supported	Not supported
COPYSOURCEDATE	Supported	Not supported
COPYSOURCEDATEUT	Supported	Not supported
COPYSOURCETIME	Supported	Not supported
COPYSOURCETIMEUT	Supported	Not supported
COPYSOURCETZ	Supported	Not supported

Table 9–25. Host Services Logical I/O and FTAM File Attributes

Attribute	Host Services	FTAM
CREATEPASSWORD	Not supported	Supported
CREATIONDATE	Supported	Supported
CREATIONDATEUT	Supported	Supported
CREATIONTIME	Supported	Supported
CREATIONTIMEUT	Supported	Supported
CREATIONTZ	Supported	Not supported
CRUNCHED	Supported	Not supported
CURRENTBLOCKLENGTH	Supported	Not supported
CURRENTRECORDLENGTH	Supported	Supported
CYCLE	Supported	Not supported
DENSITY	Supported	Not supported
DEPENDENTINTMODE	Supported	Supported
DEPENDENTSPECS	Supported	Supported
DIRECTION	Supported	Not supported
DISPOSITION	Supported	Not supported
DOCUMENTTYPE	Supported	Supported
DUMMYFILE	Supported	Not supported
ESTIMATEDRECORDS	Supported	Not supported
EXCLUSIVE	Not supported	Supported
EXECUTEDATE	Supported	Not supported
EXECUTEDATEUT	Supported	Not supported
EXECUTETIME	Supported	Not supported
EXECUTETIMEUT	Supported	Not supported
EXECUTETZ	Supported	Not supported
EXTDELIMITER	Supported	Not supported
EXTMODE	Restricted values	Restricted values
FAMILYINDEX	Supported	Not supported
FAMILYNAME	Supported	Not supported
FAMILYOWNER	Supported	Not supported
FILECLASS	Supported	Not supported
FILEEQUATED	Supported	Supported
FILEKIND	Restricted values	Restricted values

Table 9–25. Host Services Logical I/O and FTAM File Attributes

Attribute	Host Services	FTAM
FILELENGTH	Not supported	Supported
FILENAME	Supported	Supported
FILEORGANIZATION	Restricted values	Restricted values
FILESECTION	Supported	Not supported
FILESTATE	Supported	Supported
FILESTRUCTURE	Restricted values	Restricted values
FILETYPE	Restricted values	Restricted values
FILEUSE	Supported	Supported
FLEXIBLE	Supported	Not supported
FORMID	Supported	Not supported
FRAMESIZE	Restricted values	Restricted values
GENERATION	Supported	Not supported
GROUP	Supported	Not supported
HOSTNAME	Supported	Supported
INPUTTABLE	Supported	Not supported
INTERACTIVEFILE	Supported	Not supported
INTMODE	Restricted values	Restricted values
INTNAME	Supported	Supported
IOHFUNCTIONNNAME	Not supported	Not supported
IOHINTERFACENAME	Not supported	Not supported
IOHLIBACCESS	Not supported	Not supported
IOHLIBPARAMETER	Not supported	Not supported
IOHPREFIX	Not supported	Not supported
IOHSTRING	Not supported	Not supported
IOHTITLE	Not supported	Not supported
KERBEROSACCESS	Not supported	Not supported
KIND	Restricted values	Restricted values
LABEL	Supported	Not supported
LABELKIND	Supported	Not supported
LASTACCESSIBLEAREA	Supported	Not supported
LASTRECORD	Supported	Not supported
LASTSUBFILE	Supported	Not supported

Table 9–25. Host Services Logical I/O and FTAM File Attributes

Attribute	Host Services	FTAM
LFILENAME	Supported	Supported
LICENSEKEY	Restricted usage/value	Not supported
LINENUM	Supported	Not supported
LOCATECAPABLE	Supported	Not supported
LOCKEDFILE	Supported	Not supported
LTITLE	Supported	Restricted values
MAXRECSIZE	Restricted values	Restricted values
MINRECSIZE	Supported	Supported
MYUSE	Restricted usage	Supported
NEWFILE	Supported	Supported
NEXTRECORD	Supported	Not supported
ODDBLOCKSIZE	Not supported	Not supported
OFFSITE	Not supported	Not supported
OFNOTIFICATION	Supported	Supported
OPEN	Supported	Supported
OPTIONAL	Supported	Supported
OUTPUTTABLE	Supported	Not supported
OVERRIDEEXTMODE	Supported	Not supported
PAGE	Supported	Not supported
PAGECOMP	Supported	Not supported
PAGESIZE	Restricted usage	Not supported
PARITY	Supported	Not supported
PATHNAME	Supported	Not supported
PERMITTEDACTIONS	Supported	Supported
PRINTDISPOSITION	Supported	Not supported
PRINTERBACKUPDATA	Not supported	Not supported
PRINTERKIND	Supported	Not supported
PRINTPARTIAL	Supported	Not supported
PRINTREQUEST	Supported	Not supported
PRODUCT	Supported	Not supported
PROTECTION	Restricted usage	Not supported
READDATE	Supported	Not supported

Table 9–25. Host Services Logical I/O and FTAM File Attributes

Attribute	Host Services	FTAM
READDATEUT	Supported	Not supported
READREVERSECAPABLE	Supported	Not supported
READTIME	Supported	Not supported
READTIMEUT	Supported	Not supported
READTZ	Supported	Not supported
REDIRECTION	Not supported	Not supported
REINITIALIZE	Supported	Supported
RELEASEID	Supported	Not supported
RESTRICTED	Supported	Not supported
RESULTLIST	Supported	Supported
SAVEFACTOR	Supported	Not supported
SAVEPRINTFILE	Not supported	Not supported
SCRATCHPOOL	Supported	Not supported
SCREEN	Supported	Not supported
SCREENSIZE	Supported	Not supported
SEARCHRULE	Supported	Not supported
SECTORSIZE	Supported	Not supported
SECURITYADMIN	Not supported	Not supported
SECURITYGUARD	Supported	Not supported
SECURITYTYPE	Restricted values	Not supported
SECURITYUSE	Supported	Not supported
SENSITIVEDATA	Supported	Not supported
SERIALNO	Restricted usage	Not supported
SERVICE	Supported	Supported
SINGLEUNIT	Supported	Not supported
SIZEVISIBLE	Not supported	Supported
STATE	Restricted values	Supported
STATIONCOUNT	Supported	Not supported
STATIONLIST	Supported	Not supported
STATIONNAME	Supported	Not supported
STATIONSALLOWED	Supported	Not supported
STATIONSDENIED	Supported	Not supported

Table 9–25. Host Services Logical I/O and FTAM File Attributes

Attribute	Host Services	FTAM
TAPEREELRECORD	Supported	Not supported
TIMELIMIT	Restricted values	Not supported
TITLE	Supported	Restricted use
TOTALSECTORS	Supported	Not supported
TRAINID	Supported	Not supported
TRANSFORM	Supported	Not supported
TRANSLATE	Restricted values	Not supported
TRANSLATING	Supported	Not supported
TRANSMISSIONNO	Supported	Not supported
UNIQUETOKEN	Supported	Not supported
UNITS	Restricted usage	Supported
UPDATEFILE	Restricted values	Not supported
USECATALOG	Supported	Not supported
USEDATE	Supported	Not supported
USERCODE	Not supported	Supported
VERSION	Supported	Not supported
WARNINGS	Supported	Not supported
WIDTH	Supported	Not supported
YOURHOST	Supported	Supported

Section 10
Using Direct I/O Files

Direct I/O allows your program to do the following while preserving the integrity of the operating system:

- Control the physical I/O activities, such as when a READ or WRITE operation actually starts and stops.
- Handle I/O operations in an asynchronous manner.
- Handle your own error handling.
- Access all of the disk sectors that are assigned to the file if the DIOFILESTRUCTURE value is SECTORSTREAM.

To accomplish this control, your program provides the I/O buffer in the form of a direct array. Any desired mapping of the data in a buffer into records must be performed by the program. Since the logical I/O function of the operating system provides minimal intervention, you, the programmer, must consider the behavior and limitations of the individual device with which your program is interacting.

For direct I/O operations, as for I/O operations in general, the file variable is protected by the system. However, because you manage the direct arrays used as buffers in direct I/O operations, you should avoid attempting multiple, simultaneous use of the direct array in your program.

The following restrictions apply to direct I/O files:

- You cannot use direct I/O with port files.
- You cannot request translation.
- You cannot perform any WRITE operations if any of the following conditions exist:
  - The file has been crunched and the logical BLOCKSIZE does not equal the permanent BLOCKSIZE.
  - The file is a copy of a duplicated file.
  - The file is a code file and your program does not have a FILEKIND value of COMPILERCODEFILE.

Defining the Characteristics of a Direct I/O File

The following tasks need to be performed before a READ or WRITE statement is invoked and in a language that supports direct I/O:

- Identify the name of the direct file with the FILENAME attribute.
- Ensure that the INTMODE and EXTMODE values are equal because translating is not supported for direct I/O files.
- If you are defining a disk file and you set the BLOCKSTRUCTURE value to FIXED or let it default to that value, you must specify the MAXRECSIZE value. The MAXRECSIZE value is used to determine the logical length of the records.

  If you choose any other BLOCKSTRUCTURE value for a disk file, you do not need to specify a MAXRECSIZE value.
- Specify the FRAMESIZE value of the direct file, unless the file you are defining is a disk file. In that case, the FRAMESIZE value is automatically set to 8 by the I/O subsystem.
- If the file is a disk file, set the DIOFILESTRUCTURE value to indicate the FILESTRUCTURE attribute values the program can deal with and the way the program intends to access the file. Additionally, the value determines the semantics of the READ and WRITE operations on the file. This attribute value cannot be changed by file equation or a FA (File Attribute) system command.

Mnemonic Value	Meaning
ALIGNED180, BLOCKED, and STREAM	The program is capable of handling only files with a matching FILESTRUCTURE value. An open error occurs if an attempt is made to open an existing file with a FILESTRUCTURE value that does not match the DIOFILESTRUCTURE value. The value of the DEPENDENTSPECS attribute has no effect on this check.
DEPENDENT	The program is capable of handling files with any FILESTRUCTURE value and allowed to open an existing file with any FILESTRUCTURE value, subject to the normal rules for opening files. Such a program often specifies the DEPENDENTSPECS value as TRUE, but that specification is not required. In any case, the FILESTRUCTURE value is changed to the FILESTRUCTURE value of the permanent file. The FILESTRUCTURE value governs the semantics of READ and WRITE operations.

Mnemonic Value	Meaning
SECTORSTREAM	The program requests sector-oriented access to the file, regardless of the FILESTRUCTURE value of the physical file. The program can open an existing file with any FILESTRUCTURE value. The DEPENDENTSPECS attribute must have a value of TRUE. Regardless of the FILESTRUCTURE value of the physical file, the semantics of the READ and WRITE operations are the same as those of a direct I/O file with a FILESTRUCTURE value of STREAM, except that the entire last sector of an area is visible to the program when the data is being read.

- Specify the BLOCKSIZE value of the direct file. The following are value restrictions for peripheral types:

Peripheral	Value
Card reader	Value cannot be larger than the card.
Printer	Value cannot exceed the size of a print line.
Tape	Value cannot be less than 6 words.

- Create a direct array that matches the FRAMESIZE value and has enough space to contain a block.
- If you want to allow the direct array to be overlaid, set the OVERLAYABLE direct I/O buffer attribute to TRUE.

Reading to and Writing from a Direct Array Buffer

The following tasks need to be accomplished when you are reading to or writing from a direct array buffer:

- Invoke the I/O operation with an I/O statement that has one of the following syntaxes:

```
READ (<file name>,<arithmetic expression>,<direct array name>)

WRITE (<file name>,<arithmetic expression>,<direct array name>)
```

  You can also associate an event with the I/O operation.

  For files that have a FILESTRUCTURE value of ALIGNED180 and all nondisk files, the arithmetic expression must be constructed so that bits [15:16]contain the number of full words to be transferred and bits [19:3] contain the number of additional characters to be transferred. The character count is actually a physical frame count; the frame size is usually eight bits but can be six bits for some devices on some systems. The physical frame size is set by default by the operating system. In some

cases, you can modify the physical frame size through the IOFRAMESIZE bit, [41:1], in the direct buffer attribute I/O control word (IOCW).

The frame count can range from 0 (zero) through 5 for 8-bit frames, and from 0 through 7 for 6-bit frames; 0 is normally used in word-mode files. The I/O length is always specified in the number of 48-bit words and the number of additional physical frames, regardless of the values of the EXTMODE, FRAMESIZE or UNITS, and INTMODE attributes.

If the file has a FILESTRUCTURE value of STREAM or BLOCKED, the arithmetic expression is the number of FRAMESIZE units of data that are to be transferred.

Some peripheral controls require frames to be transmitted in pairs; attempts to transmit an odd number of frames can cause a descriptor error.

When performing direct I/O along with the SPACE operation, the spacing limitation of the device overrides any user-specified arithmetic expression part of the SPACE operation. In the case of a line printer, the maximum spacing is 2; in the case of a magnetic tape, the maximum is 99.

- Determine whether the I/O operation is complete by using one of the following methods:
  - Use a WAIT statement to wait on the buffer or the event associated with the operation. You can place a WAIT statement directly after the WRITE statement, or you can place the WAIT statement after other instructions that do not access information from the direct array buffer.
  - Interrogate the IOCOMPLETE and IOPENDING buffer attributes. The IOCOMPLETE attribute returns TRUE when the I/O operation is complete, and the IOPENDING attribute returns TRUE if the I/O operation is queued or in process.
- If you want to be responsible for initiating recovery of some or all I/O exceptions, use the IOMASK buffer attribute. When an I/O exception is masked out by using the IOMASK attribute, the MCP bypasses any recovery or logging procedures if that exception occurs for direct I/O files.
- Interrogate direct I/O buffer attributes to determine information about the WRITE operation. The following is an ALGOL example for interrogating the IOERRORTYPE buffer attribute for a direct array buffer named DIRECTARRAYID:

```
ERR:= DIRECTARRAYID.IOERRORTYPE;
```

Buffer Attribute	Information Obtained
IOCANCEL	If TRUE, the I/O operation attempted on this buffer was canceled and the IORESULT attribute value is 1.
IOCHARACTERS	The number of characters read into the current buffer. If you interrogate this attribute after a forward READ operation on a tape file, the value returned indicates the actual size of the tape block.
IOEOF	If TRUE, the I/O operation on this buffer encountered an end-of-file condition.

Buffer Attribute	Information Obtained
IOERRORTYPE	The value identifies the error, if any, that occurred as a result of the I/O operation on this buffer. A value of NOERROR (0) indicates that the I/O operation completed successfully
IORECORDNUM	For disk files, the random address (record or sector number, depending on the FILESTRUCTURE value) in the disk file that the last I/O operation on this buffer took place. For remote files, indicates the RSN associated with the last I/O operation on this buffer.
IORESULT	The logical result for the last I/O operation on this buffer.
IOTIME	The time, in 2.4-microsecond units, elapsed for the I/O operation.
IOWORDS	The number of words read into a buffer. If you interrogate this attribute after a forward READ operation on a tape file, the value returned indicates the actual size of the tape block.

Purging the I/O Queue

You can purge the I/O queue for a device that is not a disk or remote file by setting the IOCANCEL buffer attribute for the direct array to TRUE. This action also cancels all the outstanding I/O operations to the same unit that were initiated using direct arrays declared in the same stack as the canceled direct array.

Understanding Direct I/O Disk Files

The use of direct I/O on disk files permits considerable flexibility but also involves some fine distinctions. Direct I/O on most devices links the programmer very closely to the input/output device. However, this connection is less direct for disk files, which exist on devices that can be shared by many users. The normal disk file management system is active in allocating regions of disk to temporary or permanent files. Direct I/O is a means of accessing file data and can be used on any file, regardless of the method used to create the file.

Physical Frame Size and Odd Frames

For disk files, the physical frame size is always 8 bits; any attempt to change the frame size is ignored. Even in HEX or BCL files, where the unit size is 4 bits and 6 bits, respectively, the direct I/O length for a file with a FILESTRUCTURE value of ALIGNED180 is specified by the number of 48-bit words plus the number of 8-bit frames. Refer to the information about using the WRITE and READ statements with a direct I/O file in this section for information about specifying the length of the data to be transferred. For disk files, an odd number of frames can be requested, and the end-of-file reckoning is done with the number specified. The I/O subsystem allows I/O length values other than in 30-word multiples, but the hardware always writes that many (using zero-filling), and READ operations always begin at a sector boundary.

Only an even number of 4-bit units can be specified. One extra unit must be written in any block containing an odd number of HEX records.

Areas, Blocks, Records, and Sectors

The BLOCKSIZE, FRAMESIZE (or INTMODE and UNITS), and MAXRECSIZE attributes define the logical block and record size for the file. Because these attributes define the way a file is handled with logical (nondirect) I/O, their application to direct I/O files requires some explanation. Direct I/O files deal primarily with disk sectors, and secondarily with blocks if the file has a FILESTRUCTURE value of ALIGNED180 or BLOCKED.

The smallest unit of disk storage you can address is the sector, which holds 30 words (180 8-bit bytes). Each I/O operation begins at a sector boundary and transmits one or more contiguous sectors. On a WRITE operation, if the data runs out before the end of a sector, the disk subsystem pads the last sector with nulls. For files with a FILESTRUCTURE value of ALIGNED180 or BLOCKED, every block begins on a sector boundary and occupies one or more contiguous sectors; if the block size is not a multiple of 180 bytes, some wasted space remains at the end of each block. If the BLOCKSTRUCTURE value is FIXED, the ratio of the BLOCKSIZE value to the MAXRECSIZE value must be a fixed integer equal to the number of records in a block.

In ALGOL READ and WRITE statements, a *[<I/O option or carriage control>]* parameter, containing some text in brackets, can appear immediately after the file name. If this parameter is present, *random* I/O occurs; otherwise, the I/O is *serial*. In COBOL, the KEY IS clause invokes random I/O. For ALIGNED180 and STREAM files, serial I/O begins at the sector of the file just past the last sector read or written, regardless of any record boundaries. For BLOCKED files, serial I/O begins at the next block boundary that follows the block where the previous I/O operation ended.

For a file with a FILESTRUCTURE value of ALIGNED180, random I/O always begins at the beginning of a block, and the random address is given as record number that is then adjusted to (R DIV B) * B, where R is the random address and B is the number of records for each block. If the BLOCKSTRUCTURE value is not FIXED, then for direct I/O for disk files, the *record* is a synonym for *block*, so that records for each block always equals 1.

For a file with a FILESTRUCTURE value of BLOCKED, random I/O begins at the beginning of a block, and the random address is given as a file-relative block number.

For a file with a FILESTRUCTURE value of STREAM, random I/O is sector oriented, and the random address is given as a file-relative sector number.

The *[<I/O option or carriage control>]* parameter specifying random I/O in ALGOL takes several forms, the following of which apply to direct I/O disk files:

Parameters	Meaning
[<arithmetic expression>]	The usual syntax, where the value of <arithmetic expression> denotes the random address
[SPACE <arithmetic expression>]	Specifies the current random address plus the value of <arithmetic expression> as the new random address
[NO]	Specifies the current random address as the new random address, but does not update the file position, so that a subsequent serial I/O occurs at the same place

Any other form of the *[<I/O option or carriage control>]* parameter is ignored, and the current record position is used.

Direct I/O permits transmission to begin at any sector in the file and to continue for any length, bounded only by the buffer size and the end of the current area or the end of the file. An area of a file with a FILESTRUCTURE value of ALIGNED180 or BLOCKED consists of an integral number of blocks. The gaps, if any, between the end of the logical block and the end of the sector are accessible with direct I/O. An area of a file with a FILESTRUCTURE value of STREAM consists of exactly AREALENGTH FRAMESIZE units. Unless the DIOFILESTRUCTURE value is SECTORSTREAM, an attempt to access the unused space between the end of the area and the next sector boundary is truncated at the area boundary, and a short block result is returned.

After a direct READ or WRITE operation, the IORECORDNUM buffer attribute reports the random address (record, block, or sector number, depending on the FILESTRUCTURE value) where the transmission began. The file attributes NEXTRECORD and RECORD indicate the random address from which a subsequent serial transmission would proceed. The RECORD attribute can be used only for a file with a FILESTRUCTURE value of ALIGNED180. Neither of these points is necessarily at the beginning of a record if the FILESTRUCTURE value is STREAM, or if a serial READ or WRITE operation is done with lengths different from whole blocks.

Usually, writing to or beyond the end of the file simply extends the file and updates the end-of-file pointer; no error is reported. The exceptions to this rule include crunched files, files with the maximum number of areas, and files with FLEXIBLE equal to FALSE and the number of allowed areas, as specified by the AREAS attribute, already allocated.

A crunched file cannot be extended past the end-of-file sector, and the other two types of files cannot be extended beyond the last area. Attempts to write outside a nonextendable file are treated just like other attempts to perform READ or WRITE operations outside the file.

If I/O is attempted completely outside the file, end-of-file action is taken: no I/O takes place, bits 9 and 0 are set in the logical result descriptor returned by the WAIT function or by the buffer attribute IORESULT, and the buffer attribute IOERRORTYPE reports a WLOOREOF (6). If I/O begins within the file but extends across the area boundary or across the end of file, special action is taken: data transfer occurs with the length truncated, the logical result descriptor has bits 10 and 0 set, and IOERRORTYPE reports READPASTROW (7). Direct I/O can read from or write to the entire sector in which the end of file is located, but only to the area boundary if the DIOFILESTRUCTURE value is STREAM. When data is written, the end-of-file pointer is adjusted to the end of the WRITE operation.

End-of-File Pointers

The end-of-file pointer in a disk file specifies the last bit that has been written in the file. For direct disk files, the end pointer is set according to the starting position and length of any WRITE operation beyond the previous end position. Because direct I/O length is specified in 8-bit physical frames, the end-of-file pointer cannot always be placed at a record boundary.

A similar situation can arise with 6-bit units, where the logical and physical frame boundaries align only at whole-word and half-word boundaries.

When the end-of-file pointer is used by the logical I/O subsystem to determine the number of records in the file, the following are discarded: any partial logical frames, any partial second or subsequent record in a block, and any data in the last sector of a block past the end of the logical block.

Zero-Length I/O

Because zero-length direct READ or WRITE statements transfers no data, zero-length serial operations have no effect on the record pointer in the file. However, random operations reassign the record pointer, thus affecting subsequent nonzero-length serial operations.

Random READ or WRITE operations generate end-of-file action if the specified record number is less than 0 (zero). No other end-of-file checking is done for zero-length READ operations, but zero-length WRITE operations generate end-of-file action if the record number is past the end of a crunched or otherwise nonextendable file.

If the addressed record is in a new area, a zero-length WRITE operation causes the disk space to be allocated.

In summary, a zero-length random READ operation functions as a SEEK operation, whereas a zero-length random WRITE operation functions as a SEEK operation but also can allocate disk space.

Direct I/O Contrasted with Using Buffered Tape Drives

With direct I/O, there is an assumption that when performing a WRITE, the data has been physically transferred to the medium once the event of the I/O has occurred. Since some tape drives have hardware buffering capability, the default for direct I/O is to have buffering turned off.

However, it should be noted that there is a performance penalty when writing to a buffered tape drive with buffering disabled. In fact, for some tape drives such as the HS8500, which require a significant amount of time to reposition whenever tape motion stops, the overall performance penalty can be severe.

It is possible with a buffered tape drive to toggle buffering off and on programmatically when writing to a direct I/O tape file. You do this by setting the SYNCHRONIZE attribute to NO (to enable buffering mode) and OUT (to disable buffering mode). Thus, if it is not essential for all writes to be synchronized (that is, the I/O completion occurs only when the record has been physically written to tape), you can set SYNCHRONIZE to NO for less critical records, and then set SYNCHRONIZE to OUT just before writing any critical records. This causes the MCP to change between buffering modes based on the SYNCHRONIZE attribute.

Optimizing Direct I/O Operations

In certain cases, performance is improved for direct I/O READ and WRITE operations and WAIT operations when an event or event array element is provided with the READ and WRITE statements and then used in the WAIT statement.

There are several forms of I/O initiation statements that can be used with direct files. For the following example, assume that DF is a direct file, DA is a direct array, and E is an event or event array element.

The I/O initiation statements are as follows:

```
1.  READ(DF, length, DA);
    WRITE(DF, length, DA);

2.  READ(DF, length, DA) [E];
    WRITE(DF, length, DA) [E];
```

The following example places the same code in two positions in code:

```
BEGIN
DIRECT ARRAY DA[0:29];
DIRECT FILE F;
EVENT E;
PROCEDURE X (E); EVENT E;
   BEGIN
   READ (F, 30, DA) [E];     (B)
   END;
READ (F, 30, DA) [E];        (A)
X (E);
END.
```

The code line identified with (A) allows the compiler to ensure that E will never occur at a lexical level higher than that of DA. The code line identified with (B) does not allow the compiler to ensure this relationship.

In case B, the lexical level check is performed at run time by the MCP logical I/O READ or WRITE procedure. Less processor time is used when the event or event array element provided for use at initiation time is the same as the one used for the previous I/O on the direct array. When more than a few I/O operations are done, the processor time used is same as that in subcase 2A.

WAIT operations for completion of an I/O operation can also be divided into two cases:

```
1.  WAIT(DA);     % Used with a case 1 I/O initiation statement


2.  WAIT(E);      % Where E was supplied as a completion event in
                  % an I/O initiation statement (Example A or B)
```

In general, a case 2 WAIT operation uses less processor time than a case 1 WAIT operation.

Section 11
Using HYPERchannel (HY) Files

A HYPERchannel network is a networking system that is used by MCP systems. MCP systems support only the A223 adapter. A HYPERchannel link is a direct hardware connection that can transfer data between independent systems, including mixed vendor systems. The hardware connection between a host system and the HYPERchannel network is through an adapter interface.

Also integrated into the system software is the NETEX software that is used with the HYPERchannel network. This software enables two or more application programs, on different host computers or the same host, to communicate with each other.

Understanding a HYPERchannel Network

A HYPERchannel network is made up of two components: the Network Systems Corporation hardware and Unisys hardware.

A HYPERchannel network is composed of HYPERchannel adapters that are connected by a coaxial trunk or trunks. The maximum speed of communication between adapters is 50 megabits per second.

Each adapter has a trunk side and a host side. The host side of the adapter distinguishes one adapter from another, as the adapter is designed to interface to the specific host system to which it is connected.

A HYPERchannel network can have from one to four trunks. Each adapter can be connected to all trunks. For each transmission, two adapters and one trunk are used. For example, using four trunks and eight adapters, four simultaneous transmissions are possible, and using three trunks and six adapters, three simultaneous transmissions are possible. Trunks are selected on a transmission-by-transmission basis. Two adapters can use different trunks on successive transmissions. Trunk selection can be controlled by the host system that initiates the transmission.

Figure 11–1 illustrates a possible configuration.

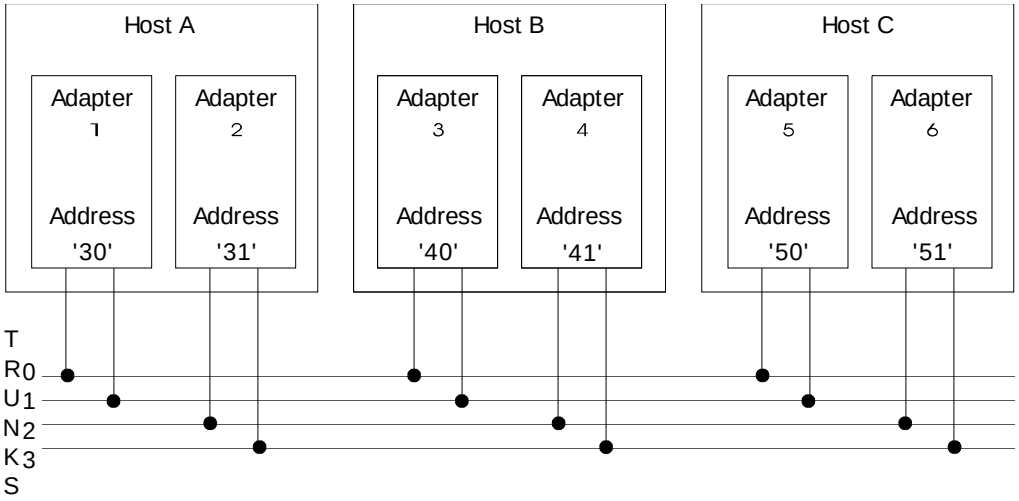


Figure 11–1. Adapter Connections

The preceding figure shows six adapters interconnecting three host systems using four trunks. Adapters 1, 3, and 5 communicate through trunks 0 and 1. Adapters 2, 4, and 6 communicate through trunks 2 and 3. Transmissions through the 1-3-5 set of adapters are logically and physically disjointed from those on the 2-4-6 set.

In the MCP environment, an adapter is connected to the host system by a HYPERchannel data link processor (HYDLP).

Communicating between Systems

To communicate between systems, the initiating host system writes a transmission to the adapter in the form of a control message, referred to as a *message proper*, followed by optional data of an arbitrary length, referred to as *associated data*. The message proper is built by the application on the initiating host system and contains the destination address of the receiving adapter, the address of the initiating adapter, the trunk or trunks that can be used for the transmission, and a presence-of-data indicator. The message proper and associated data are separate entities. Logically, however, a single transmission consists of either a message proper alone or a message proper with associated data.

Once the message proper has arrived at the destination, the application program is responsible for interpreting the message proper and the possible data.

System-to-system communication occurs only when both host systems execute a transmission through their respective adapters. The host system controls the adapter, and through its adapter, can send data to a remote adapter. However, the remote host system must read the data from the remote adapter; otherwise, the data is queued in the adapter.

The responsibility of an adapter that is connected to a host system is to determine if the message proper is being sent to a remote host or being received by the host system. The adapter then uses the information in the message proper to determine where to send the message or what remote host sent the message. Finally, the adapter sends the message proper and any associated data to the appropriate host, or places the message proper and any associated data in the data buffer of the host system when a READ operation is invoked.

The following requests are issued by the host system to the adapter to facilitate the movement of messages:

Request	Action
TRANSMIT MESSAGE	Indicates that the message proper should be sent to another adapter in the network.
TRANSMIT DATA	Indicates that the associated data should be sent to another adapter in the network.
INPUT MESSAGE	Indicates that the message proper should be placed in the data buffer.
INPUT DATA	Indicates that the associated data should be placed in the data buffer.

The command codes described in this documentation are not presented in their entirety. Users intending to use the HYPERchannel network should refer to the Network Systems Corporation documents identified in the bibliography of this guide.

Constructing a Message Proper

It is the responsibility of the software on the initiating system to build the message proper, and the responsibility of the software on the receiving system to use this information to perform a task and transmit a message proper back to the initiating system, if needed.

One piece of information the message proper contains is the destination address, referred to as the *TOADDRESS*. The TOADDRESS is unique within the network and is made up of the following parts:

Physical TO address
: The trunk address of the receiving adapter. This trunk address indicates a physical address on the coaxial cable and is unique to the adapter. The address can be a value of 1 through 255.

Logical TO address
: A logical device address within the receiving adapter. This feature allows the support of multiple dialogs with a single adapter. The adapter does not verify that this value is valid. It is the responsibility of the host software to use a valid logical TO address.

Another piece of information the message proper contains is the source address, referred to as the *FROMADDRESS*. The FROMADDRESS is unique within the network and is made up of the following parts:

Physical FROM address	The trunk address of the sending adapter. This trunk address indicates a physical address on the coaxial cable and is unique to the adapter. The address can be a value of 1 through 255.
Logical FROM address	A logical device address within the sending adapter.

Additionally, the message proper must include the following information:

Field Name	Purpose
SENDTRUNKS	Specifies the trunk or trunks that can be used for transmitting the message and any associated data.
RESPONSETRUNKS	Specifies the trunk or trunks that can be used for transmitting a response to the message.
ASSOCIATEDDATABIT	Indicates the existence of a separate data transmission associated with the message proper.

Programming for a HYPERchannel Network

Two methods of sending messages through the HYPERchannel network are available. If you have purchased NETEX, you can use its capabilities or you can program with a direct I/O HY file.

Both methods enable you to send messages through a HYPERchannel network that has been defined in a HYPERchannel map. The HYPERchannel map associates an adapter address with an adapter label. Refer to the *System Configuration Guide* for information about using SYSTEM/CONFIGURATOR to define the HYPERchannel map.

If you choose to use NETEX, refer to the *NETEX Software Reference Manual H330* for information about programming with NETEX.

Defining the Characteristics of an HY File

The following programming tasks must be performed before a READ or WRITE operation is invoked and in a language that supports direct I/O:

- Identify the unit name of the adapter by using the TITLE attribute. The unit name is specified in the configuration file of the system.
- Specify the KIND value as HY. Only one HY file can be assigned to a HYDLP.
- Specify the FILEUSE attribute value as IN, OUT, or IO.
- Define a direct array buffer that can contain the message proper as well as any associated data. You can have a 10- to 64-byte long message proper, but the length must be an even number of bytes. Usually the message proper is 12 bytes long in order to align the associated data on a word boundary, although this is not required.

  Any number of simultaneous READ and WRITE operations can be executed using a single adapter. The number of requests in progress is determined by the number of direct buffers defined by the program.
- Once the HY file is opened and the adapter is assigned to your HY file, a set of logical addresses is assigned to the file. Your program is now an endpoint of one or more logical dialogues, up to the maximum that your adapter can support.

Writing a HYPERchannel Message

Perform the following tasks to write a message to another remote host:

- Set the HYCOMMAND direct I/O buffer attribute for the direct array buffer to 22 (WRITE DATA). Once the HYCOMMAND attribute is set for any given direct array buffer, it does not require subsequent changes if the selected operation does not change.

  If you invoke a READ statement for this direct array buffer, an IOERRORTYPE value of 4 is issued when the READ statement is invoked, and the READ operation is not initiated.

  If you do not set the value of HYCOMMAND, a default HYCOMMAND is used by the operating system. That default value is 23 (READ DATA) if the READ operation is used and 22 (WRITE DATA) if the WRITE operation is used.
- Prepare the message proper and the associated data, if any. You cannot send a message to a logical address of the initiating adapter.

  The first 10 bytes of the message proper a standard format, but you can use the remaining bytes for your own use. The following table describes the format of the first 10 bytes and identifies possible values.

Word	Field Name	Possible Value
[0].[47:4]	SENDTRUNKS	Specifies which trunk or trunks can be used to send the message. You can set the field to the following values: 0 (zero), which indicates that the MCP is to set the field equal to the value corresponding to the trunk or trunks connected to the adapter. A value of 1 through 16 that indicates which of the available four trunks can be used. SENDTRUNKS is a 4-bit field, where 47:1, 46:1, 45:1, and 44:1 correspond to trunks 0, 1, 2, and 3, respectively. The message is sent on the first available trunk of those selected. If you use an invalid trunk value, the I/O operation is not initiated, and an IOERRORTYPE value of 4 is issued.
[0].[43:4]	RESPONSETRUNKS	Indicates to the remote host, which trunk or trunks are to be used to respond to the message. You can set this field to the following values: 0 (zero), which indicates that the MCP is to set the field equal to the value corresponding to the adapter. A value of 1 through 16 that indicates which of the available four trunks can be used. RESPONSETRUNKS is a 4-bit field, where 43:1, 42:1, 41:1, and 40:1 correspond to trunks 0, 1, 2, and 3, respectively. If you use an invalid trunk value, the I/O operation is not initiated, and an IOERRORTYPE value of 4 is issued.
[0].[39:6]		Not used.
[0].[33:1]	BURST MODE	This field is not used by the direct I/O HY file implementation. This feature can be used at the discretion of the user. Refer to the Network Systems Corporation (NSC) documentation listed in the bibliography of this guide.

Word	Field Name	Possible Value
[0].[32:1]	ASSOCIATEDDATABIT	If the message proper has associated data, set this bit. If you do not set ASSOCIATEDDATABIT and a WRITE operation with associated data is executed, an IOERRORTYPE value of 4 is issued. The direct array buffer logical result descriptor (LRD) issues a COMMAND REJECT condition.
[0].[31:16]	ACCESS CODE	This field is not used by the direct I/O HY file implementation. This feature can be used at the discretion of the user. Refer to the Network Systems Corporation (NSC) documentation listed in the bibliography of this guide.
[0].[15:8]	PHYSICAL TO ADDRESS	Indicates the adapter address of the receiving adapter. You can set the field to any adapter address. If the specified adapter address corresponds to the sending address, the I/O operation is not initiated, and an IOERRORTYPE of 4 is issued.
[0].[7:8]	LOGICAL TO ADDRESS	Indicates the logical address within the receiving adapter. You can set the field to any logical address.
[1].[47:8]	PHYSICAL FROM ADDRESS	Indicates the address of the sending adapter. If a value of 0 (zero) is specified, the MCP assigns the adapter address of the sending adapter. You can set the field to any adapter address.
[1].[39:8]	LOGICAL FROM ADDRESS	Indicates the logical address with the sending adapter. You can set the field to any logical address in the range from 0 (zero) through 63. If you use an invalid logical from address—a value greater than 63—the I/O operation is not initiated, and an IOERRORTYPE of 4 is issued.

Word	Field Name	Possible Value
[1].[31:8]	FUNCTION INDICATOR	This field is not used by the direct I/O HY file implementation. This feature can be used at the discretion of the user. Refer to the Network Systems Corporation (NSC) documentation listed in the bibliography of this guide.
[1].[23:8]	FUNCTION	This field is not used by the direct I/O HY file implementation. This feature can be used at the discretion of the user. Refer to the NSC documentation listed in the bibliography of this guide.

- Set the HYMPLENGTH attribute of the direct buffer being used to the length of the message proper. If you set the length to 0 (zero), 12 is selected. Remember that the value must be an even value between 10 and 64, inclusive.
- You must invoke a WRITE statement that has an arithmetic expression parameter that reflects the length of the message proper and the associated data, if any. The maximum number of words that can be written is 65536, including the message proper.

  The WRITE DATA operation is a composite function to the HYPERchannel DLP. For this operation, the DLP automatically separates the buffer into the TRANSMIT MESSAGE and TRANSMIT DATA operations for transfer to the adapter.

Reading a HYPERchannel Message

Perform the following tasks to read a message from another remote host:

- Set the value of the HYCOMMAND attribute for the direct array buffer to 23 (READ DATA). If you issue a WRITE statement for this direct array buffer, the WRITE operation is not initiated and an IOERRORTYPE value of 4 is issued. If you do not set the value of HYCOMMAND, a default HYCOMMAND value is used by the operating system. That default value is 23 (READ DATA) if the READ operation is used and 22 (WRITE DATA) if the WRITE operation is used.
- Invoke a READ statement for the direct buffer array. The HYMPLENGTH attribute for the direct buffer array contains the length in bytes of the message proper received. The direct array buffer logical result descriptor (LRD) contains the total number of words and bytes received, message proper and associated data, in the format of the IORESULT word. The IOCHARACTERS attribute returns the number of bytes transmitted including both the message proper and any associated data.

  The maximum number of words that can be received by a READ DATA request is 65536, including the message proper.
- Interrogate the IOERRORTYPE attribute to determine if the data you received is valid information for your direct array buffer.

Adapter Command Codes

Besides data transfer, HYPERchannel adapters also support statistics gathering, error recovery, and error analysis. Each adapter command fits into one of the following four categories: data transfer, adapter statistical inquiry, adapter or logical device address clearing, and adapter maintenance.

Not all categories of commands are appropriate for use by all users. Thus, varying interfaces exist to control the use of some commands.

Data transfer operations are the most accessible and have the least control.

Interfaces to GETSTATUS and SETSTATUS are defined to control access to commands that can affect the entire adapter or produce information that is relative to the entire adapter. Refer to the *GETSTATUS/SETSTATUS Reference Manual* for information on this interface.

Adapter maintenance and error analysis commands are restricted to maintenance mode access. These commands are documented with the information on peripheral test driver (PTD) in the *System Software Utilities Manual.* Simultaneous READ and WRITE operations can be invoked, and any READ or WRITE operation can be canceled.

Using I/O Buffer Attributes for HYPERchannel Files

The direct I/O buffer attributes IOCANCEL, IOCHARACTERS, IOCOMPLETE, IOPENDING, IORESULT, IOTIME, and IOWORDS are valid for the direct I/O HY file and have the same semantics as for any other device. Refer to the *File Attributes Reference Manual* for descriptions of those attributes. The direct I/O buffer attributes IOCW and IOMASK are not used for HY files.

Table 11–1 identifies the possible values that an HY file returns to your program.

Table 11–1. HY File IOERRORTYPE Values

Mnemonic Value	Value	Description
PARITYERRORTYPE	2	This value corresponds with the LRDPARITYERRORF bit of the logical result descriptor described in the IORESULT attribute.

Table 11–1. HY File IOERRORTYPE Values

Mnemonic Value	Value	Description
DESCRIPTORERROR	4	When the device associated with the buffer is a HYPERchannel unit, this error can reflect either a software or hardware error condition. For a hardware-detected error, the logical result descriptor reports a COMMAND REJECT condition. This corresponds to the LRDCOMMANDREJECTF bit in the logical result descriptor described later in this subsection with the IORESULT buffer attribute. A software-detected error can be the result of one of the following: An invalid command code was executed. A WRITE-type command code was executed with a READ statement. A READ-type command code was executed with a WRITE statement. The trunk information provided in a WRITE statement is not valid, when applicable. An out-of-bounds value for the HYMPLENGTH of a WRITE statement was specified. The message proper specified ASSOCIATEDDATABIT, and no associated data is in the buffer. The message proper did not specify ASSOCIATEDDATABIT, and the WRITE statement attempts to transfer more than HYMPLENGTH bytes.
WRITELOCKOUT	6	Because of the presence of messages in the destination adapter, the adapter is currently unable to receive new messages. This message was not delivered.
OPERATIONABORT	22	This value corresponds with the LRDABORTF bit in the logical result descriptor described in the IORESULT attribute.
REMOTEOPERATIONABORT	23	This value corresponds with the LRDREMABORTF bit in the logical result descriptor described in the IORESULT attribute.

Table 11–1. HY File IOERRORTYPE Values

Mnemonic Value	Value	Description
DEADMANTIMEOUT	24	This value corresponds with the LRDDEADMANTIMEOUTF bit in the logical result descriptor. This error specifies that the associated data for the message has been lost due to the failure of the host to read the message within the amount of time set in the deadman timer of the adapter.
DATACONFLICT	25	This value corresponds with the LRDTIMEOUTF and LRDMSGINQORASSODATAF bits in the logical result descriptor. This error specifies that the WRITE operation cannot be initiated due to data present within the adapter that must first be read.
HYDATALOST	120	Reserved.
HYMESSAGEOVERFLOW	121	Reserved.

The IOERRORTYPE values for devices other than HYPERchannel devices are documented in the *File Attributes Reference Manual*.

The IORESULT buffer attribute returns the logical result descriptor (LRD) as a Boolean value, in its entirety, after a HYPERchannel operation. An error-free HYPERchannel operation returns an LRD value of FALSE, while a HYPERchannel operation that has caused an error returns an LRD value of TRUE.

The following table describes bit definitions in the LRD that are specific to HYPERchannel operations:

Value	Description
LRDWRITELOCKOUTF [6:1]	A WRITE operation was aborted by the local adapter because the destination adapter did not respond. The most likely reason for this error is that data was not being read by the remote host from its adapter; consequently, this adapter did not execute transfer requests for additional messages.
LRDPARITYERRORF [7:1]	An adapter memory parity error occurred on a data transfer from the adapter to the host, or a parity error was detected in data received at the adapter from the host.
LRDABORTF [8:1]	A trunk operation was aborted by the local adapter. The message was undeliverable. More specific information is available in the maintenance log. The IOERRORTYPE associated with this condition is an OPERATIONABORT (22) error.
LRDCOMMANDRETRYF [10:1]	A WRITE operation was not completed to the local adapter due to a command retry condition presented by the local adapter. This condition occurred after the TRANSMIT MESSAGE was delivered and the adapter began a trunk reception, prior to executing TRANSMIT DATA.
LRDDEADMANTIMEOUTF [11:1]	Associated data has been lost due to the failure of the host to perform a READ operation within the time required by the deadman timer of the adapter.
LRDCOMMANDREJECTF [12:1]	An invalid command or command sequence has been issued by the HYPERchannel DLP (HYDLP) to the adapter that is inconsistent with the state of the adapter. The IOERRORTYPE associated with this condition is a DESCRIPTORERROR (4) error.
LRDMSGINQORASSODATAF [13:1]	A message proper has been received and is being held, or a message proper queued at the adapter has associated data. This condition is not an error condition, except in conjunction with LRDTIMEOUTF on a WRITE operation. This bit can be set with any other HYPERchannel specific bits in the logical result descriptor.

Value	Description
LRDREMABORTF [14:1]	A trunk operation was aborted by the remote adapter. See the maintenance log for detailed status information. The IOERRORTYPE associated with this condition is a REMOTEOPERATIONABORT (23) error.
LRDTIMEOUTF [15:1]	The requested operation could not be completed within the time limit set in the HYDLP.

The IORESULT values for files other than HYPERchannel files are documented in the *File Attributes Reference Manual*.

The LOGANALYZER system, documented in the *System Software Utilities Manual*, includes analysis of HYPERchannel exceptions. For WRITE operations, this analysis includes the first 10 bytes of the message proper.

Example Program

The following ALGOL program is an example program that uses direct I/O HY files. The example does not include error-handling procedures for the direct I/O HY files.

Each endpoint process validates the received data. The HYPERchannel unit on the writer system is labeled HY5C, and the HYPERchannel units on the reader systems are labeled HY52 and HY53. The trunk addresses are *5C*, *52*, and *53*, respectively.

```
WRITER PROGRAM

    BEGIN

    DIRECT FILE SHORTWRITE (KIND=HY,TITLE="HY5C.",FILEUSE=OUT);
    DIRECT ARRAY MSGPROPER1, MSGPROPER2 [0:10];
    REAL SZ;
    SZ := (64 DIV 6) & (64 MOD 6) [19:3];
    REPLACE POINTER (MSGPROPER1 [1],8)+4 BY "HI THERE";    % DATA
    REPLACE POINTER (MSGPROPER2 [1],8)+4 BY "HI THERE";    % DATA

    MSGPROPER1.HYCOMMAND  := 22;           % WRITE DATA
    MSGPROPER1 [0].[15:8] := 4"52";        % PHYSICALTO ADDRESS
    MSGPROPER1 [0].[7:8]  := 14;           % LOGICALTO ADDRESS
    MSGPROPER1 [0].[47:4] := 2;            % USE TRUNK 2
    MSGPROPER1 [1].[47:8] := 4"5C";        % PHYSICAL-FROM ADDRESS
    MSGPROPER1 [1].[39:8] := 13;           % LOGICAL-FROM ADDRESS
    MSGPROPER1.HYMPLENGTH := 64;
    WRITE (SHORTWRITE, SZ, MSGPROPER1);    WAIT (MSGPROPER1);

    MSGPROPER2.HYCOMMAND  := 22;           % WRITE DATA
    MSGPROPER2 [0].[15:8] := 4"53";        % PHYSICALTO ADDRESS
    MSGPROPER2 [0].[7:8]  := 13;           % LOGICALTO ADDRESS
    MSGPROPER2 [1].[47:16]:= 4"5C0E";      % FROM ADDRESS
    MSGPROPER2 [0].[47:4] := 1;            % USE TRUNK 3
    MSGPROPER2.HYMPLENGTH := 64;
    WRITE (SHORTWRITE, SZ, MSGPROPER2);    WAIT (MSGPROPER2);
    END.
```

```
READER PROGRAM ONE

    BEGIN

    DIRECT FILE SHORTREAD (KIND=HY, TITLE="HY52.", FILEUSE=IN);

    DIRECT ARRAY MSGPROPER1 [0:10];

    MSGPROPER1.HYCOMMAND := 23;            % READ DATA
    READ (SHORTREAD, 11, MSGPROPER1);

    IF MSGPROPER1 [1].[47:8] EQL 4"5C" AND
       MSGPROPER1 [1].[39:8] EQL 13
    THEN
      % PROCESS MESSAGE
    ELSE
      %
    END.

READER PROGRAM TWO

    BEGIN

    DIRECT FILE SHORTREAD (KIND=HY, TITLE="HY53.", FILEUSE=IN);

    DIRECT ARRAY MSGPROPER1 [0:10];

    MSGPROPER1.HYCOMMAND := 23;            % READ DATA
    READ (SHORTREAD, 11, MSGPROPER1);

    IF MSGPROPER1 [1].[47:8] EQL 4"5C" AND
       MSGPROPER1 [1].[39:8] EQL 14
    THEN
      % PROCESS MESSAGE
    ELSE
      %
    END.
```

Section 12
Using Host Control (HC) Files

For installations where the convenience and extra features of a BNA link are outweighed by efficiency considerations, direct I/O HC files provide an alternative means of using an intersystem control (ISC) link between two or more large systems for simple, high-speed data transfers.

An ISC consists of a central hub, identified by its 16-bit HUBNUMBER, and its attached HC units. Each HC unit occupies a unique HUBINDEX position, 0 through 15, on its hub, and is one connection to a host system. For systems that use data link processors (DLPs), a single HC connection permits bidirectional communication through a direct I/O HC file with a MYUSE attribute value of IO. On other systems, two HCs and a pair of direct I/O HC files, one with a MYUSE value of IN, the other with a MYUSE value of OUT, are necessary. ISC hardware enforces the desired mode of use for an HC by using an access mask register (AMR). The possible AMR modes are CLOSED (no communication possible), IN, OUT, and IO. To effect a data transfer, a WRITE operation directed to a specific HUBINDEX target from an HC with an AMR mode of OUT or IO is paired with a READ request at the HC target with an AMR mode of IO or IN. A READ operation does not require direction information. The information received includes the initiating WRITE operation HUBINDEX information along with the data received.

A direct I/O HC file program dedicates an HC or HC pair for its exclusive use. Two or more programs, controlling HCs attached to the same hub, communicate directly through the ISC link. The controlling programs are responsible for the flow of data across an ISC link, and for recovery from I/O errors occurring when READ and WRITE operations are invoked.

For a direct I/O HC file to be used, an ISC hub must be given a name in the configuration HUBMAP of each host. This is normally accomplished by running as configured groups, with each group description containing the desired HUBMAP information. If a pair of HC file programs is making permanent use of an ISC connection between two systems, the READPARTNER and WRITEPARTNER options should be included in the HUBMAP specifications. Refer to the *System Configuration Guide* for information about creating a group.

Changes to the named HCs and ISCs of a system can be made without a reconfiguration halt/load, using the LB (Host Control Unit), PG (Purge), and MODE (Unit Mode) system commands. Refer to the *System Commands Reference Manual* for information about these commands. If the running group of a system was not initially configured with HUBMAP information, direct I/O HC file control of two ISC hubs is possible by using the default HUBMAP that is created when HCs are named with the LB command and the use of the HCs are specified with the MODE command.

Defining the Characteristics of an HC File

The following programming tasks need to be performed before a READ or WRITE operation is invoked and in a language that supports direct I/O:

- Specify the KIND value as HC.
- Set the FILENAME value to the proper hub name label. When an ISC hub has a name specified in the configuration HUBMAP of its host, all online HC units connected to it are either scratch units that have no label or are labeled with the hub name.
- Specify the MYUSE value as IO. If desired or necessary, a pair of direct I/O HC files with MYUSE values of IN and OUT can be used instead.
- Define separate direct array buffers for the READ and WRITE operations. A less convenient method is to switch use of a single direct array between READ and WRITE operations. Each buffer can be any even length between 2 and the hardware maximum of 65534 characters.
- When a direct I/O HC file is opened, the operating system attempts to find an available HC whose label matches the FILENAME of the file, and whose logical mode of use matches the MYUSE value of the file. If exactly one match is found, the HC is assigned to the file; otherwise, a "DUP FILE" or "NO FILE" waiting message results. The file OPEN request initializes the AMR for the selected HC to the proper mode; IN, OUT, or IO. Regardless of the specified logical mode, the physical AMR mode of any unassigned HC unit is CLOSED.

Writing an HC Message

The following tasks must be performed to write a message to another remote host:

- Prepare the message in the write buffer array.
- If a target WRITEPARTNER hubindex was not specified in the HUBMAP, set the WRITEPARTNER buffer attribute to the appropriate integer value. If the WRITEPARTNER buffer attribute is unspecified, a value of –1, and a WRITE operation is attempted, a NOWRITEPARTNER (20) IOERRORTYPE error results. A WRITE operation receives a PARTNERVIOLATION (19) IOERRORTYPE error result when the WRITEPARTNER buffer attribute value is not –1 and a WRITEPARTNER was specified in the hub map.
- Invoke a WRITE operation.
- Wait on the buffer completion event, or interrogate the IOCOMPLETE attribute of the buffer until TRUE is returned. Once the message is initiated to the ISC hardware, it can take up to 30 seconds for a WRITE operation to complete.
- Interrogate the IOERRORTYPE buffer attribute to determine whether the WRITE operation has completed correctly. The WLOOREOF (6) IOERRORTYPE error result indicates that WRITE access was denied because the target WRITEPARTNER hubindex HC did not have an AMR mode of IN or IO. The HCWRITETIMEOUT (18) IOERRORTYPE error result indicates that the WRITE operation timed out because no corresponding READ operation was present at the target HC. Refer to I/O results in the *File Attributes Reference Manual* for error values.

Reading an HC Message

Perform the following tasks to read a message from another remote host:

- If the READPARTNER option was not set in the hub map, and input is expected from only a single hubindex source, set the appropriate integer value in the READPARTNER buffer attribute. A READ operation receives a PARTNERVIOLATION (19) IOERRORTYPE error result when the READPARTNER buffer attribute value is not –1 and a READPARTNER was specified in the HUBMAP.
- Invoke a READ operation.
- Wait on the buffer completion event, or interrogate the IOCOMPLETE attribute of the buffer until TRUE is returned. You must exercise care when waiting for a READ operation to complete, since the operation remains active in the hardware indefinitely while waiting for a corresponding WRITE operation to occur.
- Interrogate the IOERRORTYPE buffer attribute to determine if the input is from an inappropriate hubindex. The WRONGREADPARTNER (21) IOERRORTYPE error result is reported whenever input from the wrong sending hubindex is received. The WRONGREADPARTNER (21) IOERRORTYPE error result takes precedence over the SHORTRECORD (9) and PARITYERRORTYPE (2) IOERRORTYPE error results. The SHORTRECORD result indicates a long block error occurred because the hubindex HC that wrote the data sent more data than the READ buffer could hold. Refer to I/O results in the *File Attributes Reference Manual* for error values.
- Interrogate the IORESULT buffer attribute to determine the sending hubindex, if input from more than one source is allowed, or if a WRONGREADPARTNER error was returned. IORESULT also indicates if SHORTRECORD or PARITYERRORTYPE errors occurred on an input from the wrong sending hubindex (WRONGREADPARTNER error).
- Interrogate the IOCHARACTERS buffer attribute to determine the number of data characters written by the sending hubindex. Note that this can be longer than the READ buffer length if a SHORTRECORD result occurred.

Section 13
Understanding Port Files

Port files provide you with a mechanism through which a program in the MCP environment can communicate within itself or with one or more other programs.

From the point of view of the programmer, port files appear to be just another type of file. They can be opened, closed, read from, and written to. However, instead of communicating with an I/O device as do other kinds of files, a program communicates with another program when it uses a port file.

A port file is composed of one or more *port subfiles*. It is through these subfiles that actual communication takes place.

Each subfile is an *endpoint* of communication. A port subfile must connect to a corresponding port subfile in order for communication to take place. The other endpoint of communication is called the *correspondent endpoint*, and the communication established between the endpoints is called a *dialogue.* A dialogue is a single instance of a two-way communication between two endpoints. A subfile supports one dialogue.

Dialogues can be either local or remote. For a local dialogue, both endpoints reside on the same MCP system. For a remote dialogue, one endpoint resides on an MCP system and the other endpoint is either on another MCP system or on a non-MCP system.

In order to establish a dialogue between two active endpoints, the system compares the descriptions of the two endpoints. This process is called *matching*. If the descriptions match, the system establishes the dialogue and marks the subfiles as opened. You can use the subfiles in a port file to establish separate dialogues with the same correspondent process, or with different correspondent processes. Thus, one program using one port file with 10 subfiles can establish dialogues with 10 different programs. Note that these correspondent programs need not be programs in the MCP environment, and that the correspondent endpoints need not be port subfiles.

As mentioned previously, you can perform the same operations on port files that you can on files used with I/O devices: you can set attributes for port files, and you can open, close, read from, and write to port files. However, because your port file is communicating with another endpoint, you must understand the following:

- How to use a subfile index to reference a port file dialogue
- How to describe and monitor a subfile dialogue through file attributes
- How different environments can affect the service offered by a port file
- How the system matches port file descriptions and establishes dialogues
- How to manage communications between two endpoints once the dialogue is established
- How to terminate a dialogue

There are several different environments, also known as providers, available on your MCP system that provide the port file interprocess communication (IPC) capability. Some of these providers support communication between port files that are located on

- The same MCP system
- Different MCP systems
- An MCP system and a non-MCP system

You can specify a provider for your port file or you can let the system select a provider for you when the port file is opened. If you want to use a specific provider for a dialogue, use the PROVIDERGROUP file attribute. Refer to the *File Attributes Reference Manual* for information about the PROVIDERGROUP attribute.

A port service is a specific set of features and functions that a program can use for subport dialogues. You can select the port service for your port file. If you want to communicate with another endpoint using a minimum set of features or functions, set the SERVICE file attribute of your port file to the BASICSERVICE value. The default value for SERVICE is BNANATIVESERVICE. Refer to Section 21, "Understanding Port Services," for more information.

Table 13–1 shows the providers and port services that are available.

Table 13–1. Providers and Port Services

PROVIDERGROUP Value	Network Environment	Port Services Supported
*BNAV2	BNA Version 2	BASICSERVICE BNANATIVESERVICE
*BNAOSI	OSI	BASICSERVICE OSINATIVESERVICE OSISESSIONSERVICE
*HLCN	NetWare	NETBIOSSESSIONSERVICE
*LPP	None	BASICSERVICE BNANATIVESERVICE
*TCPIP	TCP/IP	TCPIPNATIVESERVICE TCPNATIVESERVICE

The minimum port functions are supported by ALGOL, COBOL68, COBOL74, COBOL85, FORTRAN77, Pascal, and Pascal83. Additional port functions required by OSI and TCPIP are supported by ALGOL, COBOL74, and Pascal. Other languages in the MCP environment that support files in the MCP environment also access port files; however, for these languages, full support of port file statements might not be available.

Since BASICSERVICE is compatible with more than one service, it is used to illustrate the common port file concepts presented in the following sections. Service-specific information is presented in separate sections under the appropriate title.

Read the following sections for basic information on port file use. Read Section 21, "Understanding Port Services," for information on how the services work. Read the service-specific sections for detailed information for each service.

Examples of a Requesting Program

The following program calls up the WEATHERMAN port file to obtain and display the weather forecast.

ALGOL Requesting Example

```
BEGIN
FILE
   MARINE_WEATHER(KIND=PORT,
                  MYNAME="WEATHERST2.",
                  SERVICE=BASICSERVICE,
                  SECURITYTYPE=PUBLIC,
                  REQUESTEDMAXRECSIZE=72,
                  FRAMESIZE=8),
   OUTPUT(KIND=REMOTE,
          FRAMESIZE=8);
EBCDIC ARRAY
   IOBUF[0:71];

IF OPEN(MARINE_WEATHER[SUBFILE 1]) = VALUE(OKRSLT) THEN
   BEGIN
   REPLACE IOBUF[0] BY 48"00", "XYZ HARBOR AREA     ";
   WRITE(MARINE_WEATHER[SUBFILE 1],72,IOBUF);
   WHILE NOT (READ(MARINE_WEATHER[SUBFILE 1],72,IOBUF) OR
              IOBUF = "##") DO
         WRITE(OUTPUT,72,IOBUF);
   CLOSE(MARINE_WEATHER[SUBFILE 1]);
   END;
END.
```

Example COBOL74 Requesting Program

```
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
   SELECT MARINE-WEATHER ASSIGN TO PORT,
          ACTUAL KEY IS MARINE-WEATHER-SUB,
          FILE STATUS IS MARINE-WEATHER-FS.
DATA DIVISION.
FILE SECTION.
FD MARINE-WEATHER
   VALUE OF
     MYNAME IS "WEATHERST2",
     SECURITYTYPE IS PUBLIC,
     REQUESTEDMAXRECSIZE IS 72,
     FRAMESIZE IS 8.
    01 MARINE-WEATHER-REC                 PIC X(72).
    01 MARINE-WEATHER-AUX-REC.
      05 MARINE-72                        PIC X(72).

WORKING-STORAGE SECTION.
01 MARINE-WEATHER-SUB                     PIC 9(5).
01 MARINE-WEATHER-FS                      PIC XX.
01 IOBUF.
   03 CODEF                               PIC XX.
   03 TEXTF                               PIC X(70).
01 FLAG                                   PIC A(5).
77 FS-SUCCESSFUL-READ                     PIC XX VALUE "00".

PROCEDURE DIVISION.
DISPLAY-INFO SECTION.
MAIN SECTION.
BEGIN.
CHANGE ATTRIBUTE YOURNAME OF MARINE-WEATHER(1) TO "WEATHERMAN.".
CHANGE ATTRIBUTE YOURHOST OF MARINE-WEATHER(1) TO "<host>.".
CHANGE ATTRIBUTE AVAILABLEONLY OF MARINE-WEATHER(1) TO TRUE.
MOVE 1 TO MARINE-WEATHER-SUB.
OPEN I-O MARINE-WEATHER.

STRING @00@,"XYZ HARBOR AREA" FOR 17 INTO IOBUF
   MOVE 1 TO MARINE-WEATHER-SUB
   WRITE MARINE-WEATHER-AUX-REC FROM IOBUF
   READ MARINE-WEATHER RECORD INTO IOBUF.
   IF MARINE-WEATHER-FS IS EQUAL TO FS-SUCCESSFUL-READ AND
      CODEF IS NOT EQUAL TO "##" THEN
      MOVE "TRUE" TO FLAG
   ELSE
      MOVE "FALSE" TO FLAG.
   PERFORM DISPLAY-IOBUF UNTIL FLAG IS EQUAL TO "FALSE".
   MOVE 1 TO MARINE-WEATHER-SUB.
```

```
   CLOSE MARINE-WEATHER.
   STOP RUN.
DISPLAY-IOBUF.
   DISPLAY IOBUF.
   MOVE 1 TO MARINE-WEATHER-SUB.
   READ MARINE-WEATHER RECORD INTO IOBUF.
   IF MARINE-WEATHER-FS IS EQUAL TO FS-SUCCESSFUL-READ AND
      CODEF IS NOT EQUAL TO "##" THEN
         MOVE "TRUE" TO FLAG
   ELSE
         MOVE "FALSE" TO FLAG.
```

Examples of a Server Program

The following program is a simple server program that issues dummy weather information through an endpoint called WEATHERMAN.

ALGOL Server Example

```
BEGIN
FILE
   MARINE_WEATHER (KIND=PORT,
                   MYNAME="WEATHERMAN.",
                   SERVICE=BASICSERVICE,
                   SECURITYTYPE=PUBLIC,
                   MAXSUBFILES=10,
                   REQUESTEDMAXRECSIZE=72,
                   FRAMESIZE=8);
EBCDIC ARRAY
   RCVBUF[0:71],
   SENDBUF[0:23,0:71];
LABEL
   EXIT;

PROCEDURE HANDLE_INPUT (INX,INPUT);
%         -----------
   VALUE INX;
   INTEGER INX;
   EBCDIC ARRAY INPUT[0];
BEGIN
INTEGER I;
% INPUT has code in first byte, location is in bytes 1-20
CASE REAL(INPUT,1) OF



   BEGIN
   0: % DUMMY MARINE FORECAST
      REPLACE SENDBUF[0,0] BY "Marine Forecast for ", INPUT[1] FOR 20,
                              " Sat. November 18              ";
      REPLACE SENDBUF[1,0] BY "      Light and variable winds,",
```

```
                              " prevailing westerlies",
                              " from noon to late ";
      REPLACE SENDBUF[2,0] BY "      afternoon.  Tomorrow gusty winds",
                              " with a small chance of rain.     ";
      REPLACE SENDBUF[3,0] BY "##";
      FOR I := 0 STEP 1 UNTIL 3 DO
         WRITE(MARINE_WEATHER[SUBFILE INX],72,SENDBUF[I,*]);
   1: % LAST 5 DUMMY BAROMETRIC READINGS
      REPLACE SENDBUF[0,0] BY "27 27 28 28 29";
      WRITE(MARINE_WEATHER[SUBFILE INX],72,SENDBUF[0,*]);

ELSE:
      REPLACE SENDBUF[0,0] BY "##";
      WRITE(MARINE_WEATHER[SUBFILE INX],72,SENDBUF[0,*]);
   END;

END OF HANDLE_INPUTEVENT;
PROCEDURE HANDLE_CHANGEEVENT (INX);
%         ----------------
   VALUE INX;
   INTEGER INX;
BEGIN
CASE MARINE_WEATHER(INX).FILESTATE OF
   BEGIN
   VALUE(AWAITINGOFFER):
   VALUE(AWAITINGHOST):
   VALUE(OPENED):
   VALUE(BLOCKED):
   VALUE(CLOSEPENDING):
   VALUE(DEACTIVATIONPENDING):
      ;
   VALUE(SHUTTINGDOWN):
      CLOSE(MARINE_WEATHER[SUBFILE INX]);
   VALUE(CLOSED):
      REPLACE MARINE_WEATHER(INX).YOURNAME BY ".";
      REPLACE MARINE_WEATHER(INX).YOURHOST BY ".";
      AWAITOPEN(MARINE_WEATHER[SUBFILE INX],DONTWAIT);
   VALUE(DEACTIVATED):
      CLOSE(MARINE_WEATHER[SUBFILE INX]);
      REPLACE MARINE_WEATHER(INX).YOURNAME BY ".";
      REPLACE MARINE_WEATHER(INX).YOURHOST BY ".";
      AWAITOPEN(MARINE_WEATHER[SUBFILE INX],DONTWAIT);
   END;
END OF HANDLE_CHANGEEVENT;
```

```
% Start program body.  Await open requests from anyone.
REPLACE MARINE_WEATHER(0).YOURNAME BY ".";
REPLACE MARINE_WEATHER(0).YOURHOST BY ".";
AWAITOPEN(MARINE_WEATHER[SUBFILE 0],DONTWAIT);
DO
   BEGIN
   % Monitor port file events
   CASE WAIT((300), % 5 minute idle
             MARINE_WEATHER.CHANGEEVENT,
             MARINE_WEATHER.INPUTEVENT) OF

BEGIN
      1:  DISPLAY ("WEATHERMAN idle.  Bye");
          GO TO EXIT;
      2:  HANDLE_CHANGEEVENT(MARINE_WEATHER.CHANGEDSUBFILE);
      3:  READ(MARINE_WEATHER[SUBFILE 0],72,RCVBUF);
          HANDLE_INPUT(MARINE_WEATHER.LASTSUBFILE,RCVBUF);
      END;
   END
UNTIL FALSE;
EXIT:
END.
```

COBOL74 Server Example

```
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
   RESERVE WORDS IS NETWORK.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
   SELECT MARINE-WEATHER ASSIGN TO PORT,
          ACTUAL KEY IS MARINE-WEATHER-SUB,
          FILE STATUS IS MARINE-WEATHER-FS.
DATA DIVISION.
FILE SECTION.
FD MARINE-WEATHER
   VALUE OF
     MYNAME IS "WEATHERMAN",
     SECURITYTYPE IS PUBLIC,
     MAXSUBFILES IS 10,
     REQUESTEDMAXRECSIZE IS 72,
     FRAMESIZE IS 8.
    01 MARINE-WEATHER-REC                 PIC X(72).

WORKING-STORAGE SECTION.
01 MARINE-WEATHER-SUB                     PIC 9(5).
01 MARINE-WEATHER-FS                      PIC XX.
01 RCVBUF                                 PIC X(72).
01 SENDBUF                                PIC X(72).
01 HOLD-TIME                              PIC 999 VALUE 300.
01 WAIT-STATE                             PIC 9(5).
01 SUBFILE-STATE                          USAGE REAL.
01 INX                                    PIC 9(5).
01 INPUT-RECORD.
   03 CODE-AREA                           PIC 9.
   03 LOCATION                            PIC 9(20).
   03 FILLER                              PIC 9(51).
01 INFINITE-LOOP                          PIC 9.
PROCEDURE DIVISION.
MAIN SECTION.
BEGIN.
CHANGE ATTRIBUTE YOURNAME OF MARINE-WEATHER(0) TO ".".
CHANGE ATTRIBUTE YOURHOST OF MARINE-WEATHER(0) TO ".".

MOVE 0 TO MARINE-WEATHER-SUB.
AWAIT-OPEN NO WAIT MARINE-WEATHER.
MOVE 1 TO INFINITE-LOOP.
PERFORM BODY-ROUTINE UNTIL INFINITE LOOP IS EQUAL TO 0.
STOP RUN.
HANDLE-INFO SECTION.
BODY-ROUTINE.
   WAIT HOLD-TIME
        ATTRIBUTE CHANGEEVENT OF MARINE-WEATHER
```

```
        ATTRIBUTE INPUTEVENT OF MARINE-WEATHER
        GIVING WAIT-STATE.
   IF WAIT-STATE IS EQUAL TO 1 THEN
        PERFORM STOP-PROGRAM
   ELSE
   IF WAIT-STATE IS EQUAL TO 2 THEN
        PERFORM HANDLE-CHANGEEVENT
   ELSE
   IF WAIT-STATE IS EQUAL TO 3 THEN
        PERFORM HANDLE-INPUT.
STOP-PROGRAM.
   DISPLAY "WEATHERMAN idle. Bye".
   CLOSE MARINE-WEATHER WITH RELEASE.
   STOP RUN.
HANDLE-CHANGEEVENT.
   MOVE 0 TO MARINE-WEATHER-SUB.
   MOVE ATTRIBUTE CHANGEDSUBFILE OF MARINE-WEATHER TO INX.
   MOVE INX TO MARINE-WEATHER-SUB.
   MOVE ATTRIBUTE FILESTATE OF MARINE-WEATHER(INX) TO
   SUBFILE-STATE.
   IF SUBFILE-STATE IS EQUAL TO
        VALUE(SHUTTINGDOWN) THEN
      MOVE INX TO MARINE-WEATHER-SUB
      CLOSE MARINE-WEATHER
   ELSE IF SUBFILE-STATE IS EQUAL TO
        VALUE(CLOSED) THEN
      CHANGE ATTRIBUTE YOURNAME OF MARINE-WEATHER(INX) TO "."
      CHANGE ATTRIBUTE YOURHOST OF MARINE-WEATHER(INX) TO "."
      MOVE INX TO MARINE-WEATHER-SUB
      AWAIT-OPEN NO WAIT MARINE-WEATHER

ELSE IF SUBFILE-STATE IS EQUAL TO
             VALUE(DEACTIVATIONPENDING) THEN
      PERFORM HANDLE-INPUT
   ELSE IF SUBFILE-STATE IS EQUAL TO
             VALUE(DEACTIVATED) THEN
      MOVE INX TO MARINE-WEATHER-SUB
      CLOSE MARINE-WEATHER
      CHANGE ATTRIBUTE YOURNAME OF MARINE-WEATHER(INX) TO "."
      CHANGE ATTRIBUTE YOURHOST OF MARINE-WEATHER(INX) TO "."
      MOVE INX TO MARINE-WEATHER-SUB
      AWAIT-OPEN NO WAIT MARINE-WEATHER.
HANDLE-INPUT.
   MOVE 0 TO MARINE-WEATHER-SUB.
   READ MARINE-WEATHER RECORD INTO INPUT-RECORD.
   MOVE ATTRIBUTE LASTSUBFILE OF MARINE-WEATHER TO INX.
   IF CODE-AREA IS EQUAL TO 0 THEN
      STRING "Marine Forecast for ",
             LOCATION,
             " Sat. November 18             "
             FOR 72 INTO SENDBUF
      PERFORM WRITE-INX-SENDBUF
```

```
           STRING "      Light and variable winds,",
                  " prevailing westerlies",
                  " from noon to late "
                  FOR 72 INTO SENDBUF
           PERFORM WRITE-INX-SENDBUF
           STRING "    afternoon.  Tomorrow gusty winds",
                  " with a small chance of rain.       "
                  FOR 72 INTO SENDBUF
           PERFORM WRITE-INX-SENDBUF
           STRING "##" FOR 72 INTO SENDBUF
           PERFORM WRITE-INX-SENDBUF
        ELSE IF CODE-AREA IS EQUAL TO 1 THEN
           MOVE "27 27 28 28 29" TO SENDBUF
           PERFORM WRITE-INX-SENDBUF
        ELSE
           MOVE "##" TO SENDBUF
           PERFORM WRITE-INX-SENDBUF.

     WRITE-INX-SENDBUF.
        MOVE INX TO MARINE-WEATHER-SUB.
        WRITE MARINE-WEATHER-REC FROM SENDBUF.
```

Section 14
Using Subfile Indexes

You access subfiles of a port file by applying an index to the file. If your port file has only one subfile, you need not specify a subfile index. Otherwise, the subfile index is mandatory. To perform an operation on the entire port file (on all the subfiles of a port file), use a subfile index of 0 (zero).

Note that an exception to this rule is that if you invoke a READ or WRITE statement on a port file that has more than one subfile, and do not specify a subfile index, the READ or WRITE operation does not fail. Instead, an index of 0 (zero) is assumed. It is preferred syntax to explicitly specify an index of 0 (zero), however.

You can specify the number of subfiles in your port file with the MAXSUBFILES port file attribute. By default, the value of MAXSUBFILES is 1.

The valid range of values to use as a subfile index is from 0 (zero) to the value of the MAXSUBFILES attribute. Using a value outside this range causes a run-time error on the file operation.

The following examples illustrate the use of subfile indexes:

Example 1

ALGOL
```
PORTF(3).MAXCENSUS := 10;
```

COBOL74
```
CHANGE ATTRIBUTE MAXCENSUS OF PORTF(3) TO 10.
```

Some port attributes, like MAXCENSUS, are *subfile attributes.* (File attributes are discussed in Section 15, "Using Attributes.") Example 1 shows how you access MAXCENSUS for subfile 3 of the port file PORTF (3 is the subfile index). In Example 1, the attribute MAXCENSUS of subfile 3 is set to the value 10.

If the subfile index 3 is outside the valid range 0 (zero) to the value of MAXSUBFILES, a run-time error occurs and the following message is displayed:

```
ATTRIBUTE ERROR: PORTF.MAXCENSUS ILLEGAL SUBFILE INDEX @ (line number)
```

Example 2

ALGOL

```
PORTF(0).MAXCENSUS := 10;
```

COBOL74

```
CHANGE ATTRIBUTE MAXCENSUS OF PORTF(0) TO 10.
```

Example 2 shows how you set the subfile attribute MAXCENSUS for all the subfiles of the port file PORTF to the value 10.

Example 3

ALGOL

```
I := PORTF(INX).ACTUALMAXRECSIZE;
```

COBOL74

```
MOVE ATTRIBUTE ACTUALMAXRECSIZE OF PORTF(INX) TO I.
```

Example 3 shows how you interrogate the value of the subfile attribute ACTUALMAXRECSIZE for subfile INX of the port file PORTF. The value of INX must be in the valid range of subfile index values for PORTF; otherwise, a run-time error occurs and the message mentioned in Example 1 is displayed.

Note that you cannot use the subfile index 0 (zero) when interrogating the value of a subfile attribute; 0 (zero) is an invalid value for subfile attribute interrogation.

Example 4

ALGOL

```
OPEN (PORTF[SUBFILE 3]);
```

COBOL74

```
FILE-CONTROL.
     SELECT PORTF
          ASSIGN TO PORT,
          ACTUAL KEY IS PORTF-SUB.
.
.
.
PROCEDURE DIVISION.
   .
   .
   MOVE 3 TO PORTF-SUB.
   OPEN I-O PORTF.
```

This example shows how you open subfile 3 of the port file PORTF. If 3 is not in the valid subfile range of 0 (zero) to the value of MAXSUBFILES, a run-time error occurs and your program is terminated with the following error message:

```
FILE PORTF OPEN ERROR: INVALID SUBFILE @ (line number)
```

Example 5

ALGOL

```
OPEN (PORTF[SUBFILE 0]);
```

COBOL74

```
FILE_CONTROL.
     SELECT PORTF
          ASSIGN TO PORT,
          ACTUAL KEY IS PORTF-SUB.
.
.
.
PROCEDURE DIVISION.
   .
   .
   MOVE 0 TO PORTF-SUB.
   OPEN I-O PORTF.
```

Example 5 shows you how to open all subfiles of the port file PORTF that are presently in a closed state. This procedure is called an OPEN ALL SUBFILES operation.

Example 6

ALGOL

```
RSLT := OPEN (PORTF[SUBFILE INX]);
```

COBOL74

```
FILE-CONTROL.
     SELECT PORTF ASSIGN TO PORT,
                  ACTUAL KEY IS PORTF-SUB,
                  FILE STATUS IS PORTF-FS.
     .
     .
77   FS-SUBPORT-NOT-OPENED   PIC XX VALUE "81".
     .
     .
PROCEDURE DIVISION.
  DECLARATIVES.
  ERR-HANDLING SECTION.
     USE AFTER STANDARD EXCEPTION PROCEDURE ON PORTF.
  BEGIN-ERR.
     MOVE ATTRIBUTE SUBFILEERROR OF PORTF(INX) TO TEMP.
     IF PORTF-FS IS EQUAL FS-SUBPORT-NOT-OPENED
       PERFORM ERROR-PROC.
  END DECLARATIVES.
  .
  .
  .
  MOVE INX TO PORTF-SUB.
  OPEN I-O PORTF.
```

If you interrogate the result of the OPEN operation as shown in Example 6, your program is not terminated if the OPEN operation returns an error. If INX is not in the valid subfile range of 0 (zero) to the value of MAXSUBFILES, RSLT is set to equal VALUE(BADSUBFILEINDEXRSLT) and your program continues. Note that not all languages can return file operation results.

Example 7

ALGOL
```
CLOSE (PORTF[SUBFILE INX]);
```

COBOL74
```
MOVE INX TO PORTF-SUB.
CLOSE PORTF.
```

Subfile indexes for other file operations are handled in the same way as the OPEN case described in Example 6. In Example 7, subfile INX of the port file PORTF is being closed.

As with OPEN, you can also interrogate the result of a CLOSE operation with some languages like ALGOL, and your program is not terminated if an error is returned.

Example 8

ALGOL
```
WRITE (PORTF[SUBFILE 0],72,DATA);
READ (PORTF[SUBFILE 0],72,DATA);
```

COBOL74
```
MOVE 0 TO PORTF-SUB.
WRITE PORTF-RECORD-NAME FROM DATA.
MOVE 0 TO PORTF-SUB.
READ PORTF RECORD INTO DATA.
```

When you index PORTF by 0 (zero) on a WRITE operation, DATA is sent on all opened subfiles of PORTF. This particular kind of WRITE operation is called a *broadcast WRITE*.

When you index PORTF by 0 (zero) on a READ operation, the next available input from a subfile of PORTF is read in. This particular kind of READ operation is called a *nonselective READ*. Refer to Section 19, "Exchanging Data" for more information about a nonselective READ.

Section 15
Using Attributes

Just as with other kinds of files, you describe and monitor your port file through file attributes. Before you open a dialogue on a subfile, you need to describe both dialogue endpoints through attributes. Attributes also allow you to perform some dialogue configuration before and during the dialogue connection. Information about dialogue activity such as incoming messages and changes in dialogue state are also passed to you through attributes.

This section discusses how to use port file attributes, rather than discussing the attributes themselves in detail. For detailed descriptions of the attributes, refer to the *File Attributes Reference Manual.*

Setting and Interrogating Attributes

You set and interrogate attributes for port files in much the same way as for other types of files. Subfile attributes are accessed by applying a subfile index on the port file. You can change the default setting or the current value of a file attribute in the following ways:

- In a file declaration statement
- In an attribute assignment statement
- At run time, through file equation

The following examples illustrate the various ways to set and interrogate port file attributes.

Example 1

ALGOL
```
INX :=PORTF.LASTSUBFILE;
```

COBOL74
```
MOVE ATTRIBUTE LASTSUBFILE OF PORTF TO INX.
```

You interrogate port file attributes in the same way you would other kinds of files. This example gets the value of the LASTSUBFILE attribute of PORTF and puts it in INX.

Example 2

ALGOL

```
FILE PORTF (KIND=PORT,
            SECURITYTYPE=PRIVATE,
            MAXSUBFILES=1024);
```

COBOL74

```
DATA DIVISION.
FILE SECTION.
FD PORTF
   VALUE OF
     SECURITYTYPE IS "PRIVATE",
     MAXSUBFILES IS 1024.
```

You can change the default values of attributes in your file declaration statement. In this example, the file PORTF is declared with its KIND attribute set to PORT, its SECURITYTYPE attribute set to PRIVATE, and its MAXSUBFILES attribute set to 1024. The attributes KIND and SECURITYTYPE are mnemonic-type attributes, and MAXSUBFILES is an integer-type attribute.

Example 3

ALGOL

```
PORTF(0).SECURITYTYPE=PRIVATE;
```

COBOL74

```
CHANGE ATTRIBUTE SECURITYTYPE OF PORTF(0) TO VALUE
PRIVATE
```

You can change the default values of port subfile attributes by setting them with a subfile index of zero. In this example, the SECURITYTYPE attribute of the file PORTF is set with a subfile index of zero specified. This sets the value of the SECURITYTYPE attribute for all the allocated subfiles and for any new subfiles that are allocated later, even after MAXSUBFILES is increased.

Example 4

ALGOL

```
PORTF.SECURITYTYPE := VALUE(PUBLIC);
PORTF.MAXSUBFILES := 100;
```

COBOL74

```
CHANGE ATTRIBUTE SECURITYTYPE OF PORTF TO VALUE PUBLIC.
CHANGE ATTRIBUTE MAXSUBFILES OF PORTF TO 100.
```

You can set attribute values in assignment statements. This example sets the value of the SECURITYTYPE attribute to PUBLIC, and the value of the MAXSUBFILES attribute to 100.

Example 5

ALGOL

```
PORTF (SECURITYTYPE=PUBLIC,MAXSUBFILES=100);
```

Some languages, like ALGOL, have a multiple attribute assignment statement. The statement in this example is equivalent to the attribute assignments in Example 3.

Example 6

CANDE, MARC, or WFL

```
RUN SERVER; FILE
PORTF(SECURITYTYPE=PUBLIC,REQUESTEDMAXRECSIZE=80)
```

When you run your program through CANDE, MARC, or WFL, you can change the value of file attributes through file equation. In Example 5, SERVER has a file whose INTNAME (internal name) attribute is PORTF. The example syntax changes the PORTF SECURITYTYPE attribute to PUBLIC, and its REQUESTEDMAXRECSIZE attribute to 80. These values override the values of SECURITYTYPE and REQUESTEDMAXRECSIZE at file declaration time.

Understanding the Difference between File and Subfile Attributes

Port attributes can be file attributes, subfile attributes, or both. A port *file attribute* applies to the file in general. A port *subfile attribute* applies on a per-subfile basis so that each subfile has its own value for the subfile attribute.

Certain attributes are both file and subfile attributes because their meaning or function can be applied to the file in general, or to each subfile individually. To reference these attributes as a file attribute, access the attribute without using a subfile index. To reference these attributes as a subfile attribute, access the attribute with a subfile index. You can use the subfile index of 0 (zero) to modify the subfile attributes of all the subfiles of a port file at once, but you cannot obtain information about a subfile attribute for all the subfiles of a port file at once.

Table 15–2 indicates whether a port attribute is a file attribute, a subfile attribute, or both. For attributes that are not both file and subfile attributes, referencing a file attribute as a subfile attribute, or referencing a subfile attribute as a file attribute, is a syntax error and is flagged at compilation time.

Example 1

ALGOL
```
REPLACE PORTF.MYNAME BY "SERVER.";
```

COBOL74
```
CHANGE ATTRIBUTE MYNAME OF PORTF TO "SERVER.".
```

In this example, the value "SERVER." of the MYNAME attribute applies to the entire port file. MYNAME is a port file attribute; you *cannot* assign a different MYNAME value for each subfile.

Example 2

ALGOL
```
REPLACE PORTF(10).YOURNAME BY "SITE21LAX.";
```

COBOL74
```
CHANGE ATTRIBUTE YOURNAME OF PORTF(10) TO "SITE21LAX.".
```

The attribute YOURNAME is a port subfile attribute that identifies the correspondent endpoint. You can assign a different YOURNAME value to each subfile. In this example, the value "SITE21LAX." is assigned to the YOURNAME attribute of subfile 10 of PORTF.

Example 3

ALGOL
```
IF PORTF.CENSUS = 0 THEN FINISH;
```

COBOL74
```
IF ATTRIBUTE CENSUS OF PORTF IS EQUAL TO ZERO
   PERFORM FINISH.
```

The attribute CENSUS is both a file and subfile attribute. When you access CENSUS as a file attribute, as in this example, it returns the number of input messages queued for the file. This example checks whether the CENSUS value of the file PORTF is 0 (zero).

Example 4

ALGOL
```
IF PORTF(10).CENSUS = 0 THEN FINISH_DIALOG(10);
```

COBOL74
```
IF ATTRIBUTE CENSUS OF PORTF(10) IS EQUAL TO ZERO
   MOVE 10 TO DIALOG-INX
   PERFORM FINISH-DIALOG.
```

When you access CENSUS as a subfile attribute, it returns the number of input messages queued for the specified subfile. This example checks whether the CENSUS value of subfile 10 of PORTF is 0 (zero).

Setting Proper Attribute Values

You can avoid errors when you are using port file attributes if you consider what checks are made at compile time and what checks are made at run time.

The following checks are made on port files at compile time:

- Whether the attribute is a file attribute or a subfile attribute (or both).
- What type the attribute is (for example, integer, event, mnemonic, Boolean, or string).
- Whether the attribute is read-only, write-only, or read/write. This characteristic determines if you can modify an attribute, interrogate it, or both.

All other checks are made at run time. To avoid attribute errors, you need to know the following information:

- The default value for the attribute.
- The file states in which the attribute can be interrogated and modified. All port attributes can be interrogated at any time, but the file states in which an attribute can be modified are attribute-dependent.
- For subfile attributes, the valid range for the subfile index.
- The range of valid values for the attribute.
- Whether the attribute is a port file attribute, and whether the attribute is supported by the port file service you are using. If an attribute is not applicable to the service of the port file, the only valid value for the attribute is the default value.

  This type of range checking and consistency checking across all attributes is performed by the system when the subfile is activated through an OPEN or AWAITOPEN statement.

If an attribute error is detected during attribute interrogation or modification, your program is flagged with an attribute error message specifying the line number of the error. Attribute errors related to event-valued attributes are always fatal; all others are nonfatal.

You can obtain much of the necessary information mentioned in the preceding two lists from Tables 15–1 and 15–2 later in this section.

Example 1

ALGOL
```
PORTF.REQUESTEDMAXRECSIZE := 100;
```

COBOL74
```
CHANGE ATTRIBUTE REQUESTEDMAXRECSIZE OF PORTF TO 100.
```

All subfiles of PORTF must be closed when you modify the value of the REQUESTEDMAXRECSIZE attribute. If PORTF was not closed, the preceding example would cause your program to be flagged with the following attribute error message:

```
ATTRIBUTE ERROR : PORTF.REQUESTEDMAXRECSIZE FILE MUST BE CLOSED @
(line number)
```

If an attribute error is detected during an OPEN, AWAITOPEN, or RESPOND operation, the operation fails with a specified error. If an attribute warning is detected during these operations, the operation continues with a warning.

A warning is generated when the attribute that is modified to an improper value might not affect the correct operation of the program. No error warning is generated if the attribute with the improper value does not affect the program at all. An example of such an attribute is the SAVEFACTOR attribute, which has no effect on port file operation.

When an attribute error or warning occurs, you may interrogate the attributes ATTERR, ATTYPE, and ATTVALUE to obtain diagnostic information about the error. ATTERR is a Boolean-type attribute that returns TRUE if an attribute error occurred. ATTYPE contains the attribute number of the last attribute that was incorrectly used. ATTVALUE contains a diagnostic code that might aid you in diagnosing the error.

Example 2

ALGOL
```
PORTF.REQUESTEDMAXRECSIZE := 100;
IF PORTF.ATTERR THEN
  HANDLE_ATT_AND_XIT (1, PORTF.ATTYPE,
                      PORTF.ATTVALUE);
```

COBOL74
```
CHANGE ATTRIBUTE REQUESTEDMAXRECSIZE OF PORTF TO 100.
IF ATTRIBUTE ATTERR OF PORTF IS EQUAL TO VALUE TRUE
   PERFORM HANDLE_ATT_AND_XIT.
```

This version of the Example 1 code handles the result of the attribute assignment. In this example, it is assumed that HANDLE_ATT_AND_XIT is a procedure declared earlier in the program to handle attribute diagnostic information.

Table 15-1 contains the port services that support each attribute. If an attribute is not applicable to a service, the only valid value for that attribute is the default value. Note that this table does not contain nonpreferred attributes. Table 15–2 contains additional information you might need when using port files.

Table 15–1. Port File Attributes and Associated Services

Attribute	BASIC	BNA	NETBIOS	OSIN	OSIS	TCPIPN	TCPN
ACTUALMAX RECSIZE	X	X	X	X	X	X	X
APPLICATION CONTEXT				X			
APPLICATION GROUP	X†	X		X	X		
ATTERR	X	X	X	X	X	X	X
ATTVALUE	X	X	X	X	X	X	X
ATTYPE	X	X	X	X	X	X	X
AVAILABLE	X	X	X	X	X	X	X
AVAILABLE ONLY	X	X	X	X	X		
BLANK		X					
BLOCKEDTIME OUT	X	X		X	X	X	X
BLOCKSTRUC TURE	X	X	X	X	X		
BUFFERS		X	X				
CENSUS	X	X	X	X	X		
CHANGEDSUB FILE	X	X	X	X	X	X	X
CHANGE EVENT	X	X	X	X	X	X	X
COMPRESSING	X	X	X	X	X		
COMPRESSION	X	X					
COMPRESSION CONTROL	X	X					
COMPRESSION REQUESTED	X	X					
CURRENTRE CORDLENGTH	X	X	X	X	X	X	X
DIALOGCHECK INTERVAL		X				X	X
DIALOGPRIO RITY		X				X	
DONOTSEARC HNETWORK		X					

Table 15–1. Port File Attributes and Associated Services

Attribute	BASIC	BNA	NETBIOS	OSIN	OSIS	TCPIPN	TCPN
EXTDELIMITER	X	X	X	X	X	X	X
EXTMODE		X					
FILECLASS	X	X	X	X	X	X	X
FILEEQUATED	X	X	X	X	X	X	X
FILENAME	X	X		X	X		
FILESTATE	X	X	X	X	X	X	X
FRAMESIZE	X	X	X	X	X	X	X
FRAMESIZE CENSUS						X	X
INPUTEVENT	X	X	X	X	X	X	X
INTERACTIVE FILE	X	X	X	X	X	X	X
INTMODE		X					
INTNAME	X	X	X	X	X	X	X
KIND	X	X	X	X	X	X	X
LASTSUBFILE	X	X	X	X	X	X	X
LFILENAME	X	X		X	X		
LTITLE	X	X	X	X	X	X	X
MAXCENSUS	X	X	X	X	X		
MAXFRAMESIZECENSUS						X	X
MAXSUBFILES	X	X	X	X	X	X	X
MYDOMAIN NAME						X	X
MYHOST	X	X		X	X	X	X
MYHOST GROUP		X					
MYIPADDRESS						X	X
MYNAME	X	X	X	X	X	X	X
NETACCESS POINT			X				
OUTPUTEVENT	X	X	X	X	X	X	X
PASSIVEOPEN			X			X	X
PATHNAME	X	X	X	X	X	X	X

Table 15–1. Port File Attributes and Associated Services

Attribute	BASIC	BNA	NETBIOS	OSIN	OSIS	TCPIPN	TCPN
PORTSEGMENTIO				X	X		
PROVIDER GROUP	X	X	X	X	X	X	X
REINITIALIZE	X	X	X	X	X	X	X
REQUESTED MAXRECSIZE	X	X	X	X	X		X
RESULTLIST	X	X	X	X	X	X	X
SECURITY TYPE	X	X	X	X	X		
SERVICE	X	X	X	X	X	X	X
STATE	X	X	X	X	X	X	X
SUBFILEERROR	X	X	X	X	X	X	X
TRANSLATE		X					
TRANSLATING		X					
YOURDOMAIN NAME						X	X
YOURHOST	X	X		X	X	X	X
YOURHOST GROUP		X					
YOURIPADDRESS						X	X
YOURNAME	X	X	X	X	X	X	X
YOURNSAPA				X	X		
YOURPRESENTATIONSEL				X			
YOURSESSIONSEL				X	X		
YOURTRANSPORTSEL				X	X		
YOURUSERCODE		X					

Note

† For BNA Version 2 and BNAOSI only.

Legend

BASIC	BASICSERVICE
BNA	BNANATIVESERVICE
NETBIOS	NETBIOSSESSIONSERVICE
OSIN	OSINATIVESERVICE
OSIS	OSISESSIONSERVICE
TCPIPN	TCPIPNATIVESERVICE
TCPN	TCPNATIVESERVICE

Use Table 15–2 for quick reference information when programming with port files. For more extensive descriptions of port file characteristics, see the *File Attributes Reference Manual.*

Table 15–2. Port File Attribute Characteristics

Attribute	No.	F,S	Type	Default	Modify
ACTUALMAXRECSIZE	201	S	Integer	N/A	Never
APPLICATIONCONTEXT	237	S	String	Null	CLOSED, OPEN RESPONSE PLEASE
APPLICATIONGROUP	202	F	String	Null	CLOSED
ATTERR	74	F	Boolean	N/A	Never
ATTVALUE	76	F	Real	N/A	Never
ATTYPE	75	F	Mnemonic	N/A	Never
AVAILABLE	48	F	Mnemonic	N/A	Never
AVAILABLEONLY	203	S	Boolean	FALSE	CLOSED
BLANK	180	S	Mnemonic	NULL	Anytime
BLOCKEDTIMEOUT	204	S	Integer	0	When CLOSED, or when OPEN has completed
BLOCKSTRUCTURE	165	F	Mnemonic	FIXED	CLOSED
BUFFERS	26	S	Integer	†	CLOSED
CENSUS	108	F,S	Integer	N/A	Never
CHANGEDSUBFILE	158	F	Integer	N/A	Never

Table 15–2. Port File Attribute Characteristics

Attribute	No.	F,S	Type	Default	Modify
CHANGEEVENT	154	F,S	Event	N/A	Never
COMPRESSING	205	S	Boolean	N/A	Never
COMPRESSION	156	S	Boolean	FALSE	When CLOSED, or when OPEN has completed
COMPRESSIONCONTROL	206	S	Mnemonic	USER	When CLOSED, or when OPEN has completed
COMPRESSIONREQUESTED	207	S	Boolean	FALSE	When CLOSED, or when OPEN has completed
CURRENTRECORDLENGTH	168	F,S	Integer	N/A	Never
DIALOGCHECKINTERVAL	208	S	Integer	0	When CLOSED, or when OPEN has completed
DIALOGPRIORITY	209	S	Integer	0	When CLOSED, or when OPEN has completed (BNA V2 only)
DONOTSEARCHNETWORK	210	S	Boolean	FALSE	CLOSED
EXTDELIMITER	418	F	Mnemonic	CRCC or UNSPECIFIED	When file is unassigned , or when file is assigned to a permanent disk and is closed
EXTMODE	10	S	Mnemonic	Value of INTMODE	CLOSED
FILECLASS	416	F	Mnemonic	N/A	Never

Table 15–2. Port File Attribute Characteristics

Attribute	No.	F,S	Type	Default	Modify
FILEEQUATED	215	F	Boolean	N/A	Never
FILENAME	162	F	String	Value of INTNAME	CLOSED
FILESTATE	170	S	Mnemonic	N/A	Never
FRAMESIZE	161	F	Integer	Value of UNITS	CLOSED
FRAMESIZECENSUS	245	F,S	Integer	N/A	Never
INPUTEVENT	152	F,S	Event	N/A	Never
INTERACTIVEFILE	427	F	Boolean	FALSE	Never
INTMODE	29	F	Mnemonic	†	CLOSED
INTNAME	72	F	String	Declared in program	When unassigned
KIND	8	F	Mnemonic	DONTCARE	CLOSED
LASTSUBFILE	106	F	Integer	N/A	Never
LFILENAME	430	F	String	INTNAME, if TITLE is not set	For disk files, anytime. For port files, when all subfiles are closed. For all other files, when unassigned .
MAXCENSUS	157	F,S	Integer	63	When CLOSED, or when OPEN has completed
MAXFRAMESIZECENSUS	246	F,S	Integer	†	When CLOSED, or when OPEN has completed
MAXSUBFILES	150	F	Integer	1	†
MYDOMAINNAME	284	F	String	Local hostname	Never
MYHOST	151	F	String	Local hostname	Never

Table 15–2. Port File Attribute Characteristics

Attribute	No.	F,S	Type	Default	Modify
MYHOSTGROUP	211	F	String	Hostgroup of local host	Never
MYIPADDRESS	286	S	String	Null	CLOSED
MYNAME	140	F	String	Null	CLOSED
NETACCESSPOINT	56	S	String	Null	CLOSED
OUTPUTEVENT	153	S	Event	N/A	Never
PASSIVEOPEN	234	S	Boolean	FALSE	CLOSED
PATHNAME	379	F	String	FILENAME or TITLE	Anytime (disk); When CLOSED (other devices)
PORTSEGMENTIO	289	S	Boolean	FALSE	CLOSED
PROVIDERGROUP	240	S	String	Null	CLOSED
REINITIALIZE	55	F	Boolean	N/A	When assigned
REQUESTEDMAXRECSIZE	212	F	Integer	320 words	CLOSED
RESULTLIST	243	F,S	String	N/A	Never
SECURITYTYPE	80	F	Mnemonic	‡	CLOSED
SERVICE	235	F	Mnemonic	‡	CLOSED
STATE	35	F,S	Word	N/A	Never
SUBFILEERROR	169	S	Mnemonic	N/A	Never
TRANSLATE	91	F	Mnemonic	NOTRANS	CLOSED
TRANSLATING	92	S	Boolean	N/A	Never
YOURDOMAINNAME	283	S	String	Null	CLOSED
YOURHOST	143	S	String	Value of MYHOST	CLOSED
YOURHOSTGROUP	213	S	String	Null	CLOSED
YOURIPADDRESS	285	S	String	Null	CLOSED
YOURNAME	141	S	String	Null	CLOSED
YOURNSAPA	297	S	String	Null	CLOSED
YOURPRESENTATIONSEL	294	S	String	Null	CLOSED
YOURSESSIONSEL	295	S	String	Null	CLOSED
YOURTRANSPORTSEL	296	S	String	Null	CLOSED

Table 15–2. Port File Attribute Characteristics

Attribute	No.	F,S	Type	Default	Modify
YOURUSERCODE	155	S	String	Value of MYSELF. USER CODE	CLOSED

Notes

† See the *File Attributes Reference Manual* for information pertaining to this cell.

‡ Depends on the value the site manager set the default to by using the NW NS MIGRATETOTBASICSERVICE system command.

Legend

No.	The number of the file attribute
F,S	Whether the attribute is a file attribute, subfile attribute, or both
Type	The attribute type (see the *File Attributes Reference Manual* for an explanation of the types)
Default	The default attribute value. ("N/A" means that there is no applicable default value, because the attribute is not modifiable.)
Modify	The file state or states in which you can modify the attribute

Section 16
Understanding Port Statements

You act on your port file or subfile by invoking port file statements. For example, you open a port subfile dialogue by invoking the OPEN statement. As mentioned previously in Section 14, "Using Subfile Indexes," you can specify a subfile index of 0 (zero) to invoke the statement on all the subfiles of a port file, or you can invoke the statement on a specific subfile. An exception is the RESPOND statement, which currently cannot accept a 0 (zero) subfile index.

The minimum port functions are supported by ALGOL, COBOL74, COBOL85, FORTRAN77, Pascal, and Pascal83. Additional port functions required by OSI and TCPIP are supported by ALGOL, COBOL74, and Pascal. Other languages in the MCP environment that support MCP environment files also access port files; however, for these languages, full support of port file statements might not be available.

All port I/O operations return a result. If you want to obtain the result on a subfile basis, you can interrogate an appropriate subfile attribute.

The READ and WRITE operations return a Boolean result. The format of this result is identical to the format of the STATE attribute. You can obtain a READ or WRITE result on a subfile basis by interrogating the STATE file or subfile attribute. While the READ or WRITE result of the STATE attribute provides faster access, there can be additional information than that included in this value. If you want to obtain the complete list of results, including those returned by the STATE attribute, you can interrogate the RESULTLIST file or subfile attribute.

All other port I/O operations return a mnemonic (integer) result. The mnemonics for the OPEN, CLOSE, and RESPOND results and their integer values are documented in the *File Attributes Reference Manual.* You can obtain the result of an operation on a subfile basis by interrogating the SUBFILEERROR subfile attribute. The SUBFILEERROR attribute also returns a mnemonic (integer) result. The list of values returned by the SUBFILEERROR attribute is consistent with, but not identical to, the values returned by a port I/O operation.

As a port subfile goes through dialogue establishment, data transfer, or dialogue termination, the subfile transits through file states. The subfile attribute FILESTATE reflects the current file state of the subfile.

Your program is notified of correspondent-initiated actions that cause a change in file state (a correspondent-initiated OPEN operation, for example) through the CHANGEEVENT attribute. CHANGEEVENT is both a file and a subfile attribute. When accessed as a subfile attribute, CHANGEEVENT is in the happened state when there is any change in the FILESTATE attribute for the subfile. CHANGEEVENT is reset when FILESTATE is interrogated. When accessed as a file attribute, CHANGEEVENT is in the happened state if any subfile of the port file has a CHANGEEVENT in the happened state.

Note that the file state of a subfile might have changed more than once between the time CHANGEEVENT arrives at the happened state and the time you interrogate the FILESTATE attribute. If the logical file is shared by more than one process and any process interrogates the FILESTATE file attribute, none of the processes subsequently receive the CHANGEEVENT in the happened state.

Correspondent-initiated actions are discussed more fully in the following sections. An example showing how you might poll on the CHANGEEVENT and INPUTEVENT attributes is found in "Understanding Event-Driven Input Techniques" in Section 19.

Table 16-1 lists the port statements and their parameters as well as the port services that support them. The exact syntax of the port statements are language-dependent and are documented in the reference manual of each programming language. For completeness, the table includes attribute interrogation and modification statements.

Table 16–1. Port Statements Used with Port Services

	BASIC	BNA	NETBIOS	OSIN	OSIS	TCPIPN	TCPN
Attribute Interrogation	X	X	X	X	X	X	
Attribute Modification	X	X	X	X	X	X	X
OPEN	X	X	X	X	X	X	X
WAIT Option	X	X	X	X	X	X	X
DONTWAIT Option	X	X	X	X	X	X	X
AVAILABLE Option	X	X	X	X	X		
OFFER Option		X					
CONNECTTIMELIMIT	X	X	X	X	X	X	X
Associated Data				X	X		
AWAITOPEN	X	X	X	X	X	X	X
WAIT Option	X	X	X	X	X	X	X
DONTWAIT Option‡	X	X	X	X	X	X	X
AVAILABLE Option	X	X	X	X	X		
CONNECTTIMELIMIT	X	X	X	X	X	X	X
PARTICIPATE				X	X		

Table 16–1. Port Statements Used with Port Services

	BASIC	BNA	NETBIOS	OSIN	OSIS	TCPIPN	TCPN
READ	X	X	X	X	X	X	X
WAIT/DONTWAIT‡	X	X	X	X	X	X	X
WRITE	X	X	X	X	X	X	X
WAIT/DONTWAIT‡	X	X	X	X	X	X	X
MOREDATA				X	X		
URGENT						X	X
CLOSE	X	X	X	X	X	X	X
Abort Disposition	X	X	X	X	X	X	X
Orderly Disposition				X	X	X	X
WAIT Option	X	X	X	X	X	X	X
DONTWAIT Option‡	X	X	X	X	X	X	X
Associated Data				X	X		
RESPOND				X	X		
ACCEPTOPEN				X	X		
REJECTOPEN				X	X		
ACCEPTCLOSE				X	X		
Associated Data				X	X		

Note

‡ NO WAIT in COBOL74

Legend

BASIC	BASICSERVICE
BNA	BNANATIVESERVICE
NETBIOS	NETBIOSSESSIONSERVICE
OSIN	OSINATIVESERVICE
OSIS	OSISESSIONSERVICE
TCPIPN	TCPIPNATIVESERVICE
TCPN	TCPNATIVESERVICE

Section 17
Preparing Your Subfile for Dialogue Establishment

You need to prepare your port file before establishing a dialogue. This preparation involves setting file attributes to configure the dialogue. Some attributes must be set before a subfile is opened.

Port attributes that describe the endpoints of the dialogue are called *matching attributes*. The set of matching attributes depends upon the service you have chosen for your port file. The specific set of matching attributes used by a service is discussed in the section for that service.

For any port service, you must always identify the possible endpoint or endpoints that your subfile could connect to. You must also identify your own endpoint to the potential correspondent. The following attributes are included in the list of attributes used for identifying or *naming* endpoints:

Matching Attribute	Description
MYHOST	The name of the host system on which your program is running. The value of the MYHOST attribute must match the YOURHOST value of the correspondent endpoint. MYHOST is always the name of the local host.
MYNAME	The name you are using for the dialogue. It must match the YOURNAME value of the correspondent endpoint. MYNAME can have a null value, but then the value of YOURNAME at the correspondent endpoint must also be null. The default value is null.
YOURHOST	The host name of the system where the correspondent endpoint is located. The value of YOURHOST must match the MYHOST value of the correspondent endpoint. A null value for YOURHOST matches to any host during dialogue establishment. The default value for this attribute is the value of MYHOST (a local dialogue).
YOURNAME	The name of the correspondent port subfile. YOURNAME must match the correspondent's MYNAME value. A null value for YOURNAME matches any MYNAME value. The default value of YOURNAME is null.

Note that these four attributes are the matching attributes common to all port services. Refer to the discussion of a specific service for the full set of matching attributes used by that service.

When your program opens a subfile, a dialogue request is issued, and an attempt is made to match your subport to a correspondent endpoint using the matching attributes. You can find more information about how your subfile is matched with a correspondent endpoint, and about how a dialogue is established, in Section 18, "Establishing a Subfile Dialogue."

In addition to the matching attributes, certain other attributes also must be configured prior to dialogue establishment. You do not necessarily have to set them, however, as all configurable attributes have default values. You should be aware of the default values so that you can determine if the defaults apply to your needs. The default values of the port file attributes are listed in Table 15–2, "Port File Attribute Characteristics."

The following are examples of attributes that you would configure while the subfile is still closed:

- AVAILABLEONLY. This attribute, when set to TRUE, causes dialogue establishment to fail instead of suspend when the correspondent endpoint is not currently reachable.
- SECURITYTYPE. This attribute can have the value PUBLIC or PRIVATE. When set to PRIVATE, your subfile matches only to a correspondent endpoint whose task is running under the usercode you have specified in the YOURUSERCODE attribute. If you are not using BNANATIVESERVICE, the value of this attribute must be PUBLIC. Your site manager can control the setting of the default with the *NW NS SET MIGRATETOBASICSERVICE* system command.

Section 18
Establishing a Subfile Dialogue

To initiate dialogue establishment on a subfile, invoke the OPEN statement. When you invoke an OPEN statement, a request for dialogue establishment is sent to the correspondent endpoint of the subfile.

To allow a subfile to await a request for dialogue establishment from its potential correspondent endpoint, invoke the AWAITOPEN statement. When you invoke the AWAITOPEN statement, no dialogue request is sent to the correspondent endpoint; your subfile is simply made7 available to receive incoming dialogue requests.

A dialogue request must be sent by at least one communication endpoint before a dialogue can be established. If both endpoints send dialogue requests at the same time, the colliding requests might be resolved into one dialogue, or they might be considered as two separate requests, depending on the service of the subfile.

Also, an OPEN or AWAITOPEN on a BNANATIVESERVICE or BASICSERVICE subport with the YOURHOST attribute set to "." may result in a small delay before the open action completes. This delay is often long enough to cause the NO MATCHING PORT message to be displayed.

Since YOURHOST = "." implies a server program that offers a subport and waits for a client, the way to avoid the delay is to offer more subports so one is always available when an open from a client comes in regardless if it comes through LPP or BNA.

Using the OPEN Statement

Use the OPEN statement to initiate dialogue establishment. When your program invokes an OPEN operation, a request for a dialogue is sent to the correspondent endpoint specified by the matching attributes.

An OPEN statement can only be invoked on a subfile when its FILESTATE value is CLOSED. The OPEN operation returns a FILENOTCLOSEDRLST (40) OPEN error when a subfile has a FILESTATE value other than CLOSED.

When your program invokes an OPEN statement, the FILESTATE value of the subfile is changed to OFFERED. If the request is accepted by the correspondent endpoint, the operation succeeds and the FILESTATE value changes to OPENED. When this happens, data transfer can begin. If the OPEN operation fails because the dialogue request was rejected or an abort occurred, the FILESTATE value changes from OFFERED to CLOSED.

Exactly how the OPEN statement works is determined by the value of the AVAILABLEONLY file attribute, and by how the OPEN control options are set. The effects of the AVAILABLEONLY attribute and the control options are discussed following the examples of OPEN statement syntax. Refer to the specific port service discussions for additional information about using the OPEN statement.

Your program should always request that the OPEN statement return a result indicating whether the OPEN operation was completed successfully, or if the operation was completed unsuccessfully, the reason for the failure. For a listing of the possible OPEN results, refer to the *File Attributes Reference Manual*.

Example Syntax

ALGOL

```
OPEN (<file name>[SUBFILE <subfile index>],<control
option>,
      <connect time limit option>)
```

COBOL74

```
MOVE <subfile index> TO <file subfile control>
OPEN <control option> <file name>
  USING <connect time limit option>.
```

The syntax elements of the OPEN statement are as follows:

Element	Purpose
File name	Identifies the port file whose subfile (or subfiles) is to be opened.
Subfile index	Specifies the subfile (or subfiles) for which the OPEN operation is to be performed.
Control option	Indicates when process control should be returned to the program. This option can have the values DONTWAIT (NO WAIT in COBOL74), WAIT, and AVAILABLE. The control option is discussed later in this section under "Understanding the OPEN Control Option Parameter."
Connect time limit option	Specifies how long (in minutes) the system is to allow for dialogue establishment. The connect time limit option is explained later in this section under "Understanding the OPEN CONNECTTIMELIMIT Parameter."

Additional parameters or parameter values that might be supported by a particular port service are defined in the description of that service.

Understanding the AVAILABLEONLY File Attribute for OPEN

The AVAILABLEONLY attribute specifies whether the OPEN operation fails or suspends when a dialogue cannot currently be established with the correspondent endpoint.

If AVAILABLEONLY is set to TRUE, the OPEN operation fails when a dialogue cannot be established with the correspondent endpoint. For example, if no matching endpoint is found, a NOFILEFOUND (4) SUBFILEERROR is returned. If a correspondent host is not reachable, an UNREACHABLEHOST (5) SUBFILEERROR is returned.

If AVAILABLEONLY is set to FALSE, the OPEN operation is suspended when a dialogue cannot currently be established with the correspondent endpoint. The subfile is suspended if the FILESTATE value is either OFFERED or AWAITINGHOST. If a correspondent host was specified but is not reachable, the FILESTATE value is set to AWAITINGHOST. In this situation, the FILESTATE value might switch between OFFERED and AWAITINGHOST. If the host should become unreachable during the opening of the subfile, the OPEN operation fails with an UNREACHABLEHOST (5) SUBFILEERROR.

The AVAILABLEONLY attribute defaults to FALSE. For detailed information about the AVAILABLEONLY file attribute for each port service, see the specific service sections.

Example 1

ALGOL

```
PORTF(1).AVAILABLEONLY := TRUE;
RSLT := OPEN(PORTF[SUBFILE 1];
```

COBOL74

```
     77 FS-SUBPORT-NOT-OPENED PIC XX VALUE "81".
.
.
.
CHANGE ATTRIBUTE AVAILABLEONLY OF PORTF(1) TO TRUE.
MOVE 1 TO PORTF-SUB.
OPEN NO WAIT PORTF.
IF PORTF-FS IS EQUAL TO FS-SUBPORT-NOT-OPENED
   IF ATTRIBUTE SUBFILEERROR OF PORTF(1) IS EQUAL TO
   VALUE(NOFILEFOUND)...
```

In this example, the system fails to establish a dialogue if the correspondent of subfile 1 of PORTF is not currently reachable. RSLT contains the result of the OPEN operation. The SUBFILEERROR attribute of subfile 1 is also set to the appropriate value.

Example 2

ALGOL

```
PORTF(1).AVAILABLEONLY :=FALSE;
RSLT := OPEN(PORTF[SUBFILE 1]);
```

COBOL74

```
     77 FS-SUBPORT-NOT-OPENED  PIC XX VALUE "81".
     77 FS-AWAITINGOFFER       PIC XX VALUE "13".
     77 FS-OFFERED             PIC XX VALUE "02".
     77 FS-OPENED              PIC XX VALUE "03".
     77 FS-AWAITINGHOST        PIC XX VALUE "01".
.
.
.
CHANGE ATTRIBUTE AVAILABLEONLY OF PORTF(1) TO FALSE.
MOVE 1 TO PORTF-SUB.
OPEN NO WAIT PORTF.
IF PORTF-FS IS EQUAL TO FS-SUBPORT-NOT-OPENED
   IF ATTRIBUTE FILESTATE OF PORTF(1) IS EQUAL TO
FS-AWAITINGOFFER
     OR
   IF ATTRIBUTE FILESTATE OF PORTF(1) IS EQUAL TO FS-
OFFERED
     PERFORM WAIT-FOR-OPEN
     IF ATTRIBUTE FILESTATE OF PORTF(1) IS EQUAL TO
FS-AWAITINGHOST
.
.
.
```

In this example, the system attempts to reach the correspondent of subfile 1 of PORTF until a dialogue can be established.

Understanding the OPEN Control Option Parameter

Your program does not have to wait for an OPEN operation to complete. The control option parameter of the OPEN statement specifies when the system returns control to your program.

Control Option	When Control Is Returned to the Program
DONTWAIT (NO WAIT in COBOL74)	Control is returned to your program as soon as possible after the OPEN operation is invoked and the subfile is in the OFFERED, AWAITINGHOST, or CLOSED file state.
WAIT	Control is returned to your program when the OPEN operation is complete and the subfile is in either the OPENED or CLOSED file state. WAIT is the default.
AVAILABLE	The OPEN operation is treated as if an OPEN WAIT operation was requested with AVAILABLEONLY set to TRUE. Use of the AVAILABLE control option is not recommended. Using the AVAILABLEONLY attribute instead is recommended.

The OPEN control option defaults to WAIT if unspecified. Other OPEN control options might be available with other port services.

When the system returns control to your program before the OPEN operation completes, the result returned by the OPEN operation, as well as the value in the SUBFILEERROR attribute, reflects only what the result of the OPEN operation is at the time control is returned to your program. When the OPEN operation completes and the subfile is in the OPENED or CLOSED file state, your program is signaled through the CHANGEEVENT, FILESTATE, and SUBFILEERROR attributes.

If you did not specify DONTWAIT (NO WAIT in COBOL74) and your program is suspended in the OPEN operation for any reason, an RSVP message is issued for your program. For example, if your program invokes an OPEN WAIT statement and no matching endpoint can be found, your program suspends at the OPEN statement with a "NO MATCHING PORT" message. Remember that this would occur only if AVAILABLEONLY is FALSE. If AVAILABLEONLY is TRUE, the OPEN operation fails immediately if dialogue establishment is not possible when the dialogue request is made, regardless of the value of the control option.

Example 1

ALGOL

```
PORTF(1).AVAILABLEONLY := FALSE;
RSLT := OPEN(PORTF[SUBFILE 1],WAIT);
```

COBOL74

```
CHANGE ATTRIBUTE AVAILABLEONLY OF PORTF TO FALSE.
MOVE 1 TO PORTF-SUB.
OPEN WITH WAIT PORTF.
```

If AVAILABLEONLY is FALSE (it defaults to FALSE), and the OPEN operation on subfile 1 of PORTF is suspended for any reason, the program is suspended with an RSVP message. When the OPEN operation continues and completes, the program can interrogate RSLT or the SUBFILEERROR attribute to determine the success or failure of the OPEN operation.

Example 2

ALGOL

```
RSLT := OPEN(PORTF[SUBFILE 1],DONTWAIT);
```

COBOL74

```
MOVE 1 TO PORTF-SUB.
OPEN WITH NO WAIT PORTF.
```

In this example, control is returned to the program as soon as possible, and the OPEN operation continues in parallel with the program. RSLT contains the result of the OPEN operation at the point when control is returned. If RSLT is equal to OKRSLT (1), the program is signaled that the OPEN operation completed through the CHANGEEVENT attribute. The program can find out the final result of the OPEN operation through the SUBFILEERROR and FILESTATE attributes.

Understanding the OPEN CONNECTTIMELIMIT Parameter

The CONNECTTIMELIMIT parameter of the OPEN statement specifies the maximum number of minutes that the program allows for the system to establish a dialogue with the subfile.

If the subfile does not move to the OPENED file state in the number of minutes specified in CONNECTTIMELIMIT, an implicit CLOSE ABORT operation is performed on the subfile, and the OPEN operation fails with a TIMELIMITEXCEEDED (27) SUBFILEERROR.

If the CONNECTTIMELIMIT parameter is not specified or is set to 0 (zero), the time limit is indefinite.

Example 1

ALGOL

```
RSLT := OPEN(PORTF[SUBFILE 1],
             WAIT, CONNECTTIMELIMIT = 10);
```

COBOL74

```
MOVE 1 TO PORTF-SUB.
OPEN WITH WAIT PORTF USING CONNECT-TIME-LIMIT OF HOLD-
TIME.
  where
       WORKING-STORAGE had
       01 HOLD-TIME PIC 99 VALUE 10.
```

In this example, control is returned to the program after the OPEN operation either completes successfully or fails. The program has specified a time limit of 10 minutes in which the OPEN operation is to complete. If the operation exceeds 10 minutes, the operation fails.

Example 2

ALGOL

```
RSLT := OPEN(PORTF[SUBFILE 1],
             DONTWAIT, CONNECTTIMELIMIT = 10);
```

COBOL74

```
MOVE 1 TO PORTF-SUB.
OPEN WITH NO WAIT PORTF USING CONNECT-TIME-LIMIT OF HOLD-
TIME.
  where
       WORKING-STORAGE had
       01 HOLD-TIME PIC 99 VALUE 10.
```

In this example, control is returned to the program as soon as possible. RSLT reflects the result of the OPEN operation only at the point when control is returned. The OPEN operation then continues in parallel with the program and must complete within 10 minutes. The program is notified of the final result of the OPEN operation through the CHANGEEVENT, FILESTATE, and SUBFILEERROR attributes.

Using the AWAITOPEN Statement

Use the AWAITOPEN statement to direct your program to wait for requests for dialogue establishment. AWAITOPEN differs from OPEN in that AWAITOPEN does not attempt to establish a dialogue; it merely makes a subfile available to accept dialogue requests from other programs.

When an AWAITOPEN statement is invoked, the local system is searched for an outstanding dialogue request that matches the subfile. If no matching dialogue request has already been received, the AWAITOPEN operation either fails or is saved for matching with future dialogue requests, depending upon the value of the AVAILABLEONLY attribute.

AWAITOPEN can be invoked on a subfile only when its FILESTATE value is CLOSED. The AWAITOPEN operation returns a FILENOTCLOSEDRLST (40) OPEN error when a subfile has a FILESTATE value other than CLOSED.

When you invoke an AWAITOPEN statement on a subfile, the subfile moves from the CLOSED to the AWAITINGOFFER file state. When a matching dialogue request is received, the subfile moves from the AWAITINGOFFER to the OPENED file state. If an error or abort occurred during the establishment procedure, the subfile is moved to the CLOSED file state.

Exactly how the AWAITOPEN statement works is determined by the value of the AVAILABLEONLY file attribute, and by how the AWAITOPEN control options are set. The effects of the AVAILABLEONLY attribute and the control options are discussed in the following subsections.

Example Syntax

ALGOL

```
AWAITOPEN (<file name>[SUBFILE <subfile index>],<wait
option>,
           <connect time limit option>)
```

COBOL74

```
MOVE <subfile index> TO <file subfile control>.
AWAIT-OPEN <control option> <file name>
  USING <connect time limit option>.
```

The syntax elements of the AWAITOPEN statement are as follows:

Element	Purpose
File name	Identifies the port file whose subfile (or subfiles) is to be made available for dialogue establishment.
Subfile index	Specifies the subfile (or subfiles) for which the AWAITOPEN operation is to be performed.
Control option	Indicates when process control should be returned to the program. This option can have the values DONTWAIT (NO WAIT in COBOL74), WAIT, and AVAILABLE. The control option is discussed later in this section under "Understanding the AWAITOPEN Control Option Parameter."
Connect time limit option	Specifies how long (in minutes) the system is to allow for dialogue establishment. The connect time limit option is explained later in this section under "Understanding the AWAITOPEN CONNECTTIMELIMIT Parameter."

Additional parameters or parameter values that might be supported by a particular port service are defined in the description of that service.

Understanding the AVAILABLEONLY File Attribute for AWAITOPEN

The AVAILABLEONLY file attribute controls whether the system suspends or terminates the AWAITOPEN operation when no matching dialogue request already exists at the local host.

If AVAILABLEONLY is set to TRUE, the local host is searched for an outstanding dialogue request that matches the subfile. If no matching dialogue request has been received, the AWAITOPEN operation fails with a NOFILEFOUNDRSLT (2) OPEN result and a NOFILEFOUND (4) SUBFILEERROR.

If AVAILABLEONLY is FALSE, the subfile OPEN request is saved until a matching dialogue request is received.

The default value for AVAILABLEONLY is FALSE.

Note that some services do not maintain a list of incoming dialogue requests that could not be matched when received. For these services, the AVAILABLEONLY attribute should be set to FALSE. Otherwise, the window in which a dialogue can be established is very small. Refer to the section that pertains to the service you are interested in to determine if the service maintains a list of incoming requests.

Example 1

ALGOL

```
PORTF(1).AVAILABLEONLY := TRUE;
RSLT := AWAITOPEN(PORTF[SUBFILE 1]);
```

COBOL74

```
CHANGE ATTRIBUTE AVAILABLEONLY OF PORTF(1) TO TRUE.
MOVE 1 TO PORTF-SUB.
AWAIT-OPEN NO WAIT PORTF.
IF PORTF-FS IS EQUAL TO FS-SUBPORT-NOT-OPENED
   IF ATTRIBUTE SUBFILEERROR OF PORTF(1) IS EQUAL TO
NOTFILEFOUND
   .
   .
   .
```

If no matching dialogue request is available at the local host, the AWAITOPEN operation fails and RSLT contains the reason for the failure. For example, if the correspondent host is reachable but no dialogue request has been received, then RSLT contains the value NOFILEFOUNDRSLT (2).

Example 2

ALGOL

```
PORTF(1).AVAILABLEONLY := FALSE;
RSLT := AWAITOPEN(PORTF[SUBFILE 1]);
```

COBOL74

```
CHANGE ATTRIBUTE AVAILABLEONLY OF PORTF(1) TO FALSE.
MOVE 1 TO PORTF-SUB.
AWAIT-OPEN NO WAIT PORTF.
IF PORTF-FS IS EQUAL TO FS-SUBPORT-NOT-OPENED
   IF ATTRIBUTE FILESTATE OF PORTF(1) IS EQUAL TO
FS-AWAITINGOFFER
     OR
   IF ATTRIBUTE FILESTATE OF PORTF(1) IS EQUAL TO FS-
OFFERED
   PERFORM WAIT-ON-OPEN.
IF ATTRIBUTE FILESTATE OF PORTF(1) IS EQUAL TO FS-
AWAITINGHOST
.
.
.
```

If no matching dialogue request is available at the local host, the AWAITOPEN operation suspends until a match is received.

Understanding the AWAITOPEN Control Option Parameter

The control option parameter of the AWAITOPEN statement allows you to specify when control is to be returned to your program. If control is returned to your program before the AWAITOPEN operation is complete, the AWAITOPEN operation continues in parallel with your program, and your program is informed of the result of the operation through the CHANGEEVENT, FILESTATE, and SUBFILEERROR file attributes.

The control option parameter has the following values:

Control Option	When Control Is Returned to the Program
DONTWAIT (NO WAIT in COBOL74)	Control is returned to your program as soon as possible and the subfile is in either the AWAITINGOFFER or the CLOSED file state.
WAIT	Control is returned to your program only after the subfile is matched or the AWAITOPEN operation fails. Control is returned to your program when the FILESTATE value of the subfile is either OPENED or CLOSED. WAIT is the default value.
AVAILABLE	The AWAITOPEN operation is treated as if you requested an AWAITOPEN WAIT operation with AVAILABLEONLY set to TRUE. Unisys recommends that you do not use the AVAILABLE option. Use the AVAILABLEONLY attribute instead.

The default value for the control option parameter is WAIT.

Example 1

ALGOL

```
RSLT :=AWAITOPEN(PORTF[SUBFILE 1],WAIT);
```

COBOL74

```
MOVE 1 TO PORTF-SUB.
AWAIT-OPEN WAIT PORTF
IF PORTF-FS IS NOT EQUAL TO "00"
.
.
.
```

If the control option of the AWAITOPEN operation is WAIT, execution control is returned to the program after dialogue establishment either fails or succeeds. RSLT returns the result of dialogue establishment. The duration of the AWAITOPEN operation depends on the AVAILABLEONLY attribute. If AVAILABLEONLY is TRUE, the AWAITOPEN operation fails immediately if the subfile cannot currently be matched.

Example 2

ALGOL

```
PORTF(1).AVAILABLEONLY := FALSE;
RSLT := AWAITOPEN(PORTF[SUBFILE 1],DONTWAIT);
```

COBOL74

```
CHANGE ATTRIBUTE AVAILABLEONLY OF PORTF(1) TO FALSE.
MOVE 1 TO PORTF-SUB.
AWAIT-OPEN NO WAIT PORTF.
IF PORTF-FS NOT EQUAL TO "00"
.
.
.
```

If AVAILABLEONLY is FALSE (the default), the AWAITOPEN operation is suspended until the subfile can be matched to an incoming dialogue request. The AWAITOPEN DONTWAIT statement or AWAITOPEN NO WAIT statement in COBOL74 returns control to the program as soon as possible, and the AWAITOPEN operation continues in parallel with the program. RSLT contains the result of the AWAITOPEN operation at the time control is returned. You can obtain the final result by using the CHANGEEVENT, FILESTATE, and SUBFILEERROR attributes.

Example 3

ALGOL

```
PORTF(1).AVAILABLEONLY := TRUE;
RSLT := AWAITOPEN(PORTF[SUBFILE 1],DONTWAIT);
```

COBOL74

```
CHANGE ATTRIBUTE AVAILABLEONLY OF PORTF(1) TO TRUE.
MOVE 1 TO PORTF-SUB.
AWAIT-OPEN NO WAIT PORTF.
IF PORTF-FS IS NOT EQUAL TO "00"
.
.
.
```

If AVAILABLEONLY is TRUE, the AWAITOPEN operation fails immediately if the subfile cannot currently be matched to an available dialogue request. The AWAITOPEN DONTWAIT statement or the AWAITOPEN NO WAIT statement in COBOL74 returns control to the program as soon as possible, and the AWAITOPEN operation continues in parallel with the program. RSLT contains the result of the AWAITOPEN operation at the time control is returned. You can obtain the final result by using the CHANGEEVENT, FILESTATE, and SUBFILEERROR attributes.

Understanding the AWAITOPEN CONNECTTIMELIMIT Parameter

The CONNECTTIMELIMIT parameter of the AWAITOPEN statement specifies the maximum number of minutes that the program allows for the system to match the subfile to an incoming dialogue request.

If the subfile does not move to OPENED in the number of minutes specified in CONNECTTIMELIMIT, an implicit CLOSE ABORT is performed on the subfile, and the AWAITOPEN operation fails with a TIMELIMITEXCEEDED (27) SUBFILEERROR.

The default for this parameter is indefinite.

Example 1

ALGOL

```
RSLT := AWAITOPEN(PORTF[SUBFILE 1],
                  WAIT, CONNECTTIMELIMIT = 10);
```

COBOL74

```
MOVE 1 TO PORTF-SUB.
AWAIT-OPEN WITH WAIT PORTF USING CONNECT-TIME-LIMIT OF
HOLD-TIME.
       where
            WORKING-STORAGE had
            01 HOLD-TIME PIC 99 VALUE 10.
```

In this example, control is returned to the program after the AWAITOPEN operation either completes successfully or fails. The program has specified a time limit of 10 minutes in which the AWAITOPEN operation is to complete. If the operation exceeds 10 minutes, the operation fails.

Example 2

ALGOL

```
RSLT := AWAITOPEN(PORTF[SUBFILE 1],
                  DONTWAIT, CONNECTTIMELIMIT = 10);
```

COBOL74

```
MOVE 1 TO PORTF-SUB.
AWAIT-OPEN WITH NO WAIT PORTF USING CONNECT-TIME-LIMIT OF
HOLD-TIME.
       where
            WORKING-STORAGE had
            01 HOLD-TIME PIC 99 VALUE 10.
```

In this example, control is returned to the program as soon as possible. RSLT reflects the result of the AWAITOPEN operation only at the point when control is returned. The AWAITOPEN operation then continues in parallel with the program and must complete within 10 minutes. The program is notified of the final result of the AWAITOPEN operation through the CHANGEEVENT, FILESTATE, and SUBFILEERROR attributes.

Section 19
Exchanging Data

Once your program has established a dialogue with a correspondent endpoint, it can begin exchanging data. Data transfer is performed on subfiles through the READ and WRITE statements.

There are two general types of data transfer: message-oriented and data-stream-oriented. Most services offer message-oriented data transfer, as does BASICSERVICE, which is used to illustrate the information about exchanging data. Data-stream data transfer is described separately under "Reading Data" and "Writing Data" in this section.

Message-oriented services maintain message boundaries during transfer. On a READ or WRITE operation, data is delivered to and received from the program in message units (although message truncation can occur for long messages). Data-stream-oriented services do not maintain message boundaries during data transport. All data is considered as a sequence of bytes that are relayed in the network in implementation-dependent fragments. TCPIPNATIVESERVICE is an example of a data-stream-oriented service.

Reading Data

All input data passed through subfiles are retrieved from the READ queue. You can use the same READ techniques on port files as you use on other kinds of files, although only array-row I/O is supported for port files. Aside from the subfile specification and the control option, the port file READ syntax is the same as with other kinds of files:

Example Syntax

ALGOL

```
READ (<file name>[SUBFILE<subfile
index>,DONTWAIT],<arithmetic
      expression>, <array row>)
```

COBOL74

```
MOVE <subfile index> TO <file actual key>.
READ <file name> RECORD
     WITH NO WAIT INTO <array row>.
```

The syntax elements of the READ statement are as follows:

Element	Purpose
File name	Identifies the port file whose subfile is to be read.
Subfile index	Identifies the index number of the subfile. A subfile index of 0 indicates a nonselective READ operation. Note that if you omit the subfile specification and your port file has more than one subfile, the READ operation fails with an invalid subfile error.
DONTWAIT (NO WAIT in COBOL74)	An optional parameter. When specified, as in the example, your program is not suspended at the READ statement when no input data is currently available. If you do not specify DONTWAIT or NO WAIT in COBOL74, the default WAIT is assumed, in which case your program is suspended at the READ statement until input is available.
Arithmetic expression	Indicates the number of data units to be read. (The data units are words if the FRAMESIZE attribute value is 48, and bytes if the FRAMESIZE value is 8.)
Array row	Represents a row in an array in your program. This array acts as the buffer area for your program.

Example

ALGOL

```
READ (PORTF[SUBFILE 1,DONTWAIT], 72, IOBUF);
```

COBOL74

```
MOVE 1 TO PORTF-SUB.
READ PORTF RECORD WITH NO WAIT INTO IOBUF.
```

Understanding Nonselective READ Operations

You can invoke a READ operation on a specified subfile, or you can invoke the READ operation on the whole port file by specifying a subfile index of 0 (zero). When you invoke a READ operation with a subfile index of 0 (zero), you are invoking what is called a *nonselective READ*.

On a nonselective READ, the system returns the next available input for the port file. You can determine which subfile the input came from by interrogating the LASTSUBFILE attribute after the READ statement.

Example 1

ALGOL

```
CASE WAIT(PORTF.INPUTEVENT, ...) OF
  BEGIN
  1:  READ (PORTF[SUBFILE 0],72,IOBUF);
      INX := PORTF.LASTSUBFILE;
                .
                .
                .
  END;
```

COBOL74

```
WAIT ATTRIBUTE CHANGEEVENT OF PORTF
     ATTRIBUTE INPUTEVENT OF PORTF GIVING WAIT-STATE.
     GO TO HANDLE-CHANGEEVENT, HANDLE-INPUT DEPENDING ON
WAIT-STATE.
```

ALGOL also allows you to retrieve the value of LASTSUBFILE in the following manner. In this example, the value of LASTSUBFILE is placed in INX. If you use this syntax, another program that shares the logical file cannot change the value of LASTSUBFILE before your program obtains the LASTSUBFILE value.

Example 2

ALGOL

```
READ (PORTF[SUBFILE INX:0],72,IOBUF);
```

Understanding the READ WAIT/DONTWAIT Option Parameter

Normally, the READ statement waits until input is available. If you specify the DONTWAIT (NO WAIT in COBOL74) option, the READ operation returns immediately with a NODATAFORREAD (53) I/O result if the READ queue is empty. The default option is WAIT.

Regardless of the option, a READ operation always returns EOF when the READ queue is empty and the subfile is in a file state where no data can be received. Note that an exception to this is a READ operation on a subfile in a CLOSED file state, which causes an implicit OPEN operation to be performed. In this case the OPEN parameters are assigned all the default OPEN options.

Example 1

ALGOL
```
READ (PORTF[SUBFILE 1, DONTWAIT],72,IOBUF);
```

COBOL74
```
MOVE 1 TO PORTF-SUB.
READ PORTF RECORD WITH NO WAIT INTO IOBUF.
```

This READ DONTWAIT statement or READ NO WAIT statement in COBOL74 causes your program to terminate with the following error if no input data is available for reading:

```
FILE PORTF I/O ERROR; NO AVAILABLE MESSAGE @ <line number>
```

Example 2

ALGOL
```
RSLT: = READ(PORTF[SUBFILE 1, DONTWAIT],72,IOBUF);
IF RSLT THEN
   NO_INPUT(1)             % NO INPUT ON SUBFILE 1
ELSE
   HANDLE_INPUT(1,IOBUF);  % PROCESS SUBFILE 1 INPUT
```

COBOL74
```
MOVE 1 TO PORTF-SUB.
READ PORTF RECORD WITH NO WAIT INTO IOBUF AT END
PERFORM
EOF-SECT.
```

If you interrogate the result of the READ DONTWAIT statement or READ NO WAIT statement in COBOL74, as in the preceding example, your program is not terminated if a READ operation failure should occur. You can use RSLT or interrogate the value of the STATE or RESULTLIST attribute for the subfile (in the previous NO_INPUT routine) to determine if the READ DONTWAIT operation or READ NO WAIT operation failed because of a no input condition.

Determining Message Size for Message-Oriented Services READ Operations

Three file attributes affect the size of messages read through port files when message-oriented services are used:

Attribute	Purpose
FRAMESIZE	Specifies the data unit size used for the port file
REQUESTEDMAXRECSIZE	Specifies the maximum I/O length allowed for the port subfile
BLOCKSTRUCTURE	Specifies whether the message length is fixed or variable

Setting the Data Unit Size for READ Operations

Use the FRAMESIZE file attribute to set the size of the unit of data used for I/O operations in the port file. FRAMESIZE can have the value 8 when the data unit is a byte, or 48 when the data unit is a word. The REQUESTEDMAXRECSIZE, ACTUALMAXRECSIZE, and CURRENTRECORDLENGTH attributes are all expressed in FRAMESIZE units. So, for example, ACTUALMAXRECSIZE refers to bytes if FRAMESIZE is set to 8, and refers to words if FRAMESIZE is set to 48. The default value for FRAMESIZE is 48.

Setting the Maximum Message Length for READ Operations

The ACTUALMAXRECSIZE attribute value is derived from the value of the REQUESTEDMAXRECSIZE file attribute, which is an attribute you configure while the port file is closed. REQUESTEDMAXRECSIZE is the maximum message length that you would like to handle on a READ or a WRITE operation. The default value of REQUESTEDMAXRECSIZE is 1920 bytes or 320 words. When the subfile is opened, you can use the ACTUALMAXRECSIZE subfile attribute to determine the actual maximum data length allowed on a READ or a WRITE operation.

ACTUALMAXRECSIZE might be lowered from REQUESTEDMAXRECSIZE because of record size restrictions of the subfile service provider. If the underlying network is BNA, ACTUALMAXRECSIZE is lowered to the lowest REQUESTEDMAXRECSIZE value of the two endpoints.

Other factors could result in the READ length being less than the maximum specified by ACTUALMAXRECSIZE. The length of the message returned by the READ is the smallest of the following:

- The length specified in the READ statement
- The length of the I/O buffer, although you would normally declare the length of the buffer to be at least ACTUALMAXRECSIZE long
- The ACTUALMAXRECSIZE attribute of the subfile
- The actual length of the data received (if BLOCKSTRUCTURE is EXTERNAL)

CURRENTRECORDLENGTH is both a file and a subfile attribute. When interrogated as a file attribute, the CURRENTRECORDLENGTH value reflects the value as of the last I/O operation for the file. When interrogated as a subfile attribute, the CURRENTRECORDLENGTH value reflects the value as of the last I/O operation on the specified subfile. The CURRENTRECORDLENGTH attribute returns either the ACTUALMAXRECSIZE value if the BLOCKSTRUCTURE value is FIXED, or the length of the message received from the correspondent endpoint if the BLOCKSTRUCTURE value is EXTERNAL. CURRENTRECORDLENGTH is not necessarily indicative of the amount of data actually given to the user.

If the length of the actual message received from the correspondent endpoint is greater than the smallest value from the previous list, then the message is truncated when it is passed to your program and the MESSAGETRUNCATEDWARNING (79) I/O result is returned by the READ operation. A READ operation that is given a MESSAGETRUNCATIONWARNING, as a result of the receiving I/O buffer not being large enough, experiences some performance overhead.

Since the factors described in the previous paragraphs can make the actual length of the messages sent to your program different than the length you specified, you can reduce message truncation (and loss of data and loss of performance) by checking the ACTUALMAXRECSIZE value for a subfile after the dialogue is established. The port service determines the value during dialogue establishment and sets ACTUALMAXRECSIZE. Check this value and use it in your READ statements and when declaring the size of your I/O buffer.

Example

ALGOL

```
MSGSIZE[1] := PORTF(1).ACTUALMAXRECSIZE;
RSLT := READ (PORTF[SUBFILE 1], MSGSIZE[1], IOBUF);
```

COBOL74

```
FILE SECTION.
FD PORTF.
   01 PORTF-REC PIC X(1920).
   01 PORTF-AUX-REC.
      03 PORTF-INDEXED PIC X(1920) INDEXED BY IO-SUB-A.
  .
  .
  .
WORKING STORAGE.
  77 IO-SUB PIC 9999.
  01 IO-BUF PIC X(1920).
  01 IO-AUX REDEFINES IO-BUF.
     03 IO-INDEXED PIC X(1920) INDEXED BY IO-SUB-B.
  .
  .
  .
PROCEDURE DIVISION.
  .
  .
  .
  MOVE ATTRIBUTE ACTUALMAXRECSIZE OF PORTF(1) TO MSGSIZE.
  READ PORTF RECORD.
  PERFORM MOVE-DATA
     VARYING IO-SUB-A FROM 0 BY 1 UNTIL IO-SUB-A IS EQUAL
TO
     MSGSIZE.
  .
  .
  .
MOVE-DATA.
  SET IO-SUB-B TO IO-SUB-A.
  MOVE PORTF-INDEXED(IO-SUB-A) TO IO-INDEXED(IO-SUB-B).
```

Note that this method does not prevent truncation if the correspondent program sends a message with a data length longer than ACTUALMAXRECSIZE. The only way to guarantee that no message truncation occurs is by prior agreement with the correspondent program. You can check to see if RSLT has the value MESSAGETRUNCATEDWARNING (79) in field [26:10].

Setting Message Length to Variable or Fixed

You can use the BLOCKSTRUCTURE attribute to indicate whether port file messages are to be fixed or variable in length. This attribute affects only READ statements.

For port files, the following values are valid:

Mnemonic Value	Meaning
FIXED	Messages are supposedly all the same length: the value of ACTUALMAXRECSIZE. Messages read that are shorter than the maximum allowed by ACTUALMAXRECSIZE are increased to the fixed size by the addition of blanks to the right of the last character in the message. This process is called *blank filling*. The CURRENTRECORDLENGTH attribute always returns the value of ACTUALMAXRECSIZE.
EXTERNAL	Messages are not blank filled. The actual length of the message that was available for a program to read is indicated by the CURRENTRECORDLENGTH attribute value.

Notes:

- *The actual length of the message in the program buffer, including blanks, might be less than the CURRENTRECORDLENGTH value. Refer to "Setting the Maximum Message Length for READ Operations" earlier in this section.*
- *The READ operation performance is somewhat slower when the BLOCKSTRUCTURE file attribute value is FIXED, because of the blank-filling process.*

FIXED is the default value for this attribute.

Understanding Data-Stream-Oriented Services READ Operations

The TCPIPNATIVESERVICE and the TCPNATIVESERVICE are data-stream-oriented services. No message boundaries are maintained during the transmission of data in the network. As a result, all data that the provider delivers is received by the port program, because the next READ operation returns the next series of available bytes. If you request more bytes of information than exist, the READ operation returns only the existing information and does not wait to receive additional bytes of information. The READ operation does wait for data, if 0 (zero) bytes are available and a READ WAIT operation was invoked.

The FRAMESIZE attribute specifies the data unit size used for the port file and must be set to 8 (8 bits).

The ACTUALMAXRECSIZE file attribute value is the maximum segment size, in FRAMESIZE units, transmitted by the TCP provider.

Data-stream-oriented services use the FRAMESIZECENSUS and MAXFRAMESIZECENSUS attributes as opposed to the message-oriented attributes, CENSUS and MAXCENSUS.

Notes:

- *If you use TCPNATIVESERVICE, the REQUESTEDMAXRECSIZE file attribute value is used to calculate the ACTUALMAXRECSIZE value.*
- *Refer to "Setting the Maximum Message Length for READ Operations" earlier in this section for information about the REQUESTEDMAXRECSIZE and ACTUALMAXRECSIZE attributes and how to ensure that the entire message has been read.*
- *The value of FRAMESIZE can be either 8 or 48 if you are using TCPNATIVESERVICE; however, Unisys recommends 8.*

Understanding Event-Driven Input Techniques

Your program is made aware of the presence of data in the READ queue by the INPUTEVENT attribute. For programs that poll on input, you can include this attribute as one of the events in your WAIT statement.

INPUTEVENT is an event-typed attribute that is in the happened state if there is data in the READ queue. The system resets INPUTEVENT when the READ queue is empty. INPUTEVENT is both a file and a subfile attribute. When you interrogate it as a file attribute, INPUTEVENT indicates if there is any input for the port file. If you interrogate INPUTEVENT as a subfile attribute, INPUTEVENT indicates whether there is any input for the specified subfile.

Typically, your WAIT statement also includes the attribute CHANGEEVENT as well as a timeout specification. CHANGEEVENT is both a file and subfile attribute, and it notifies the program when there is a change of state in any of the port subfiles. The timeout specification prevents your program from being infinitely suspended in the WAIT statement.

If you are using multiple WAIT statements, CHANGEEVENT should come before INPUTEVENT.

Although the CENSUS attribute for message-oriented service and the FRAMESIZECENSUS attribute for data-stream-oriented service can be used, Unisys recommends that INPUTEVENT be used to detect the presence of data.

Example

ALGOL

```
DO
   BEGIN
   CASE WAIT((60),PORTF.CHANGEEVENT,PORTF.INPUTEVENT) OF
      BEGIN
      1:  HANDLE_TIMEOUT;
      2:  HANDLE_CHANGEEVENT
            (PORTF.CHANGEDSUBFILE);
      3:  READ(PORTF[SUBFILE 0],72,IOBUF);
          HANDLE_INPUT(PORTF.LASTSUBFILE,IOBUF);
      END;
   END
UNTIL TERMINATE;
```

COBOL74

```
WAIT HOLD-TIME
     ATTRIBUTE CHANGEEVENT OF PORTF
     ATTRIBUTE INPUTEVENT OF PORTF
     GIVING WAIT-STATE.
GO TO HANDLE-TIMEOUT
      HANDLE-CHANGEEVENT
      HANDLE-INPUT DEPENDING ON WAIT-STATE.
```

Writing Data

Use the WRITE statement to send out data on a subfile dialogue. You can use the same write techniques on port files as you use on other kinds of files, although only array-row I/O is supported with port files. Aside from the subfile specification, the port file WRITE syntax is the same as with other kinds of files:

Example Syntax

ALGOL

```
WRITE (<file name>[SUBFILE <subfile
index>,DONTWAIT],<arithmetic
        expression>,<array row>);
```

COBOL74

```
MOVE <subfile index> TO <file actual key>.
WRITE <file name>-REC WITH NO WAIT FROM <array row>.
```

The syntax elements of the WRITE statement are as follows:

Element	Purpose
File name	Identifies the port file whose subfile (or subfiles) is to be written to.
Subfile index	Specifies the subfile for which the WRITE operation is to be performed. A subfile index of 0 indicates a broadcast WRITE operation. Note that if you omit the subfile specification and your port file has more than one subfile, the WRITE operation fails with an invalid subfile error.
DONTWAIT (NO WAIT in COBOL74)	An optional parameter. If you specify DONTWAIT, your program is not suspended at the WRITE statement when no output buffers are available.
Arithmetic expression	Indicates the number of data units to be written. This expression is interpreted by the system as the length of the output data being written. You can avoid truncating messages by setting this equal to the ACTUALMAXRECSIZE value.
Array row	Identifies a row in an array in your program. This array contains the output data for your program.

Example

ALGOL

```
WRITE (PORTF[SUBFILE 1], 72, IOBUF);
```

COBOL74

```
FILE SECTION.
  FD PORTF.
  01 PORTF-REC PIC X(1920).
  01 PORTF-AUX-REC.
    05 PORTF-72 PIC X(72).
    05 FILLER PIC X(1848).
.
.
.
MOVE 1 TO PORTF-SUB.
WRITE PORTF-AUX-REC FROM IOBUF.
```

Understanding Broadcast WRITE Operations

You can invoke a WRITE operation on a specified subfile, or you can invoke the WRITE on the whole port file by specifying a subfile index of 0 (zero). When you invoke WRITE with a subfile index of 0 (zero), you are invoking what is called a *broadcast WRITE*. For a broadcast WRITE, the system sends the data you specify on all opened subfiles of the port file.

Example

ALGOL
```
WRITE (PORTF[SUBFILE 0],72,IOBUF);
```

COBOL74
```
MOVE 0 TO PORTF-SUB.
WRITE PORTF-AUX-REC FROM IOBUF.
```

Understanding the WRITE WAIT/DONTWAIT Option Parameter

When you invoke a WRITE operation on a specified subfile without using the DONTWAIT (NO WAIT in COBOL74) option, your program waits while the system looks for an available buffer. If a buffer is available, the I/O subsystem transmits the data to the correspondent subfile and returns control to your program. The program doing the READ operation on the subfile queues the transmitted data as an input message to be read.

If no buffers are available, your program waits until a system buffer is available.

System buffers can become unavailable when the CENSUS—FRAMESIZECENSUS for message-oriented services—attribute value of the correspondent endpoint reaches the MAXCENSUS—MAXFRAMESIZECENSUS for message-oriented services—attribute value of the correspondent endpoint. System buffers become available again when enough messages have been read by the correspondent endpoint to lower the CENSUS or FRAMESIZECENSUS value several numbers below the MAXCENSUS or MAXFRAMESIZECENSUS value.

System buffers can also become unavailable because of other flow control situations in the underlying network protocol. For example, buffers can become unavailable when the network processor runs out of memory for remote dialogues.

Regardless of the option, a WRITE operation always returns EOF if the subfile is in a file state where output is not allowed. Note that if you perform a WRITE on a subfile in a CLOSED file state, an implicit OPEN is performed. In this case, the OPEN parameters are assigned all the default OPEN options.

If you want your program to continue executing without waiting until buffers are available, you should specify the DONTWAIT (NO WAIT in COBOL74) option in the WRITE statement and then interrogate the result of the WRITE operation. Your program can determine if a buffer is unavailable either by

- Interrogating the STATE attribute value for the subfile. If buffers are unavailable, bits [8:1] and [0:1] are set.
- Interrogating the RESULTLIST attribute. If buffers are unavailable, the value is NOBUFFERFORWRITE (55).

Once your program has determined that no buffer is available, your program can wait on the value of the OUTPUTEVENT attribute to determine when to attempt the next WRITE operation.

When the OUTPUTEVENT happened state is TRUE, buffers are available for transmitting the data specified in the WRITE statement. Once your program detects this new condition, it can invoke another WRITE statement. However, your program still needs to determine if buffers still are available, since it is possible for the happened state of the OUTPUTEVENT attribute to be reset between the time your program interrogated it and the time your program actually attempts a WRITE operation.

Example 1

ALGOL

```
WHILE NOT DONE DO
    BEGIN
    .
    .
    .
    IF RSLT :=WRITE (PORTF [SUBFILE 1,DONTWAIT],72,IOBUF)
THEN
        BEGIN
        WAIT (PORTF(1).OUTPUTEVENT);
                 % Can attempt the next write

        .
        .
        .
        END;
        .
        .
        .
    END WHILE;
```

COBOL74

```
*      In this example, PORTF-SUB is the ACTUAL KEY and
*      PORTF-FS is the FILE STATUS key.
*
 MOVE 1 PORTF-SUB.
 PERFORM WRITE-DATA VARYING INDX FROM 1 BY 1 UNTIL INDX
IS EQUAL TO
     100.

 WRITE-DATA.
.
.
.
 WRITE PORTF-AUX-REC WITH NO WAIT FROM IOBUF.
 IF PORTF-FS NEQ 0 THEN
     IF PORTF-FS EQUAL TO 95 THEN
*      No buffers condition
        WAIT ATTRIBUTE OUTPUTEVENT OF PORTF.
*      Can proceed with the write
       .
       .
       .
```

Example 1 shows how to use the DONTWAIT or NO WAIT option of the WRITE statement. The code also interrogates bit [8:1] of the STATE attribute value to determine if a buffer is available. If a buffer is unavailable, the program waits on the value of the OUTPUTEVENT attribute before it invokes another WRITE statement.

Example 2

ALGOL
```
WRITE (PORTF[SUBFILE 1, DONTWAIT],72,IOBUF);
```

COBOL74
```
MOVE 1 TO PORTF-SUB.
WRITE PORTS-AUX-REC WITH NO WAIT FROM IOBUF.
```

This WRITE DONTWAIT statement or WRITE WITH NO WAIT statement in COBOL74 causes the program to terminate with the following error if no system buffer is available for processing the WRITE data:

```
FILE PORTF I/O ERROR: NO AVAILABLE BUFFER @ <line number>
```

Determining Message Size for Message-Oriented Services WRITE Operations

Two file attributes affect the size of messages written through port files for message-oriented services:

Attribute	Use
FRAMESIZE	Use FRAMESIZE to specify the data unit size used for the port file.
REQUESTEDMAXRECSIZE	For all services except TCPIPNATIVESERVICE, use REQUESTEDMAXRECSIZE to specify the maximum message length allowed for the port file.

Note: *The file attribute BLOCKSTRUCTURE has no effect on WRITE operations.*

Setting the Data Unit Size for WRITE Operations

Use the FRAMESIZE file attribute to set the size of the unit of data used for I/O operations in the port file. FRAMESIZE can have the value 8 when the data unit is a byte, or 48 when the data unit is a word. The REQUESTEDMAXRECSIZE, ACTUALMAXRECSIZE, and CURRENTRECORDLENGTH attributes are all expressed in FRAMESIZE units. So, for example, ACTUALMAXRECSIZE refers to bytes if FRAMESIZE is set to 8, and refers to words if FRAMESIZE is set to 48. The default value for FRAMESIZE is 48.

Setting the Message Length for WRITE Operations

The ACTUALMAXRECSIZE is derived from the value of the REQUESTEDMAXRECSIZE file attribute, which is a value you assign when the port file is closed. REQUESTEDMAXRECSIZE is the maximum message length that you would like to handle on a READ or a WRITE operation. The default value of REQUESTEDMAXRECSIZE is 1920 bytes or 320 words. When the subfile is opened, you can use the ACTUALMAXRECSIZE subfile attribute to determine the actual maximum data length allowed on a READ or a WRITE operation. ACTUALMAXRECSIZE might be lowered from REQUESTEDMAXRECSIZE because of service provider record size restrictions. If the underlying network is BNA, ACTUALMAXRECSIZE is lowered to the lower of the two REQUESTEDMAXRECSIZE endpoint values.

Other factors could result in the WRITE length being less than the maximum specified by ACTUALMAXRECSIZE. The length written is the smallest of the following:

- The length specified in the WRITE statement
- The length of the I/O buffer
- The ACTUALMAXRECSIZE value of the subfile

The length specified in the WRITE statement is assumed to be the data length that you want the program to transmit. If the actual WRITE length is smaller than this message length, the message is truncated when it is transmitted and the MESSAGETRUNCATEDWARNING (79) I/O result is returned.

The length, written in FRAMESIZE units, is returned by the CURRENTRECORDLENGTH attribute. CURRENTRECORDLENGTH is both a file and a subfile attribute. When interrogated as a file attribute, CURRENTRECORDLENGTH returns the length of the last I/O operation for the file. When interrogated as a subfile attribute, CURRENTRECORDLENGTH returns the length of the last I/O operation on the specified subfile.

Since these factors can make the actual length of the messages written from your program different than the length you specified, prevent truncation of messages (and loss of data) by checking the ACTUALMAXRECSIZE value for a subfile after the dialogue is established. The system determines the value during dialogue establishment and sets ACTUALMAXRECSIZE. Check this value and ensure that the length you use in your WRITE statements is not longer than ACTUALMAXRECSIZE.

Example

ALGOL

```
MSGSIZE[1] := COMFILE(1).ACTUALMAXRECSIZE;
RSLT := WRITE(COMFILE[SUBFILE1],MSGSIZE[1],IOBUF);
```

COBOL74

```
WORKING-STORAGE SECTION
   01 TEMP-REAL1     USAGE REAL.
   01 TEMP-REAL2     USAGE REAL.
   01 TEMP-REAL3     USAGE REAL.
.
.
.
MOVE 1 TO PORTF-SUB.
WRITE PORTF-SMALL FROM IOBUF
IF PORTF-FS NOT EQUAL TO ZERO.
.
.
MOVE ATTRIBUTE STATE OF PORTF TO TEMP-REAL1.
MOVE TEMP-REAL1 TO TEMP-REAL2 [26:9:10].
MOVE PORTF-FS TO TEMP-REAL3.
IF TEMP-REAL2 IS  EQUAL TO TEMP-REAL3
.
.
.
```

You can check to see if truncation has occurred (or verify that it has not) by interrogating the result of the WRITE statement. If a message has been truncated, the WRITE operation returns the value MESSAGETRUNCATEDWARNING (79) in field [26:10].

Understanding Message Size for Data-Stream-Oriented Services WRITE Operations

Since no message boundaries are maintained during transmission of data in a network with data-stream-oriented messages, all data bytes are transmitted unless a WRITE DONTWAIT operation is invoked and the provider cannot handle the number of bytes identified for transmission. Refer to Section 26, "Using TCPIPNATIVESERVICE" for more information.

The FRAMESIZE file attribute value specifies the data unit size. You must set the FRAMESIZE value to 8 when you are using TCPIPNATIVESERVICE.

For TCPIPNATIVESERVICE, the ACTUALMAXRECSIZE file attribute value is the optimum segment size, in FRAMESIZE units, transmitted by the TCP provider. You can use this information to maximize the utilization of the underlying transmission mechanisms. If your program delivers data through the WRITE operation to the provider in multiples of ACTUALMAXRECSIZE bytes, the performance of the program might increase.

Notes:

- *If you use TCPNATIVESERVICE, the ACTUALMAXRECSIZE file attribute value controls the size of the WRITE operation. Refer to "Setting the Message Length for WRITE Operations" earlier in this section for information about using ACTUALMAXRECSIZE.*
- *It is recommended that you set the FRAMESIZE value to 8 when you are using TCPNATIVESERVICE.*

Section 20
Closing a Dialogue

A program terminates dialogues through the CLOSE statement. Dialogue termination is also caused if your program exits the block where the port file is declared without closing the port file explicitly.

Your program is informed of correspondent-initiated and provider-initiated dialogue termination through the CHANGEEVENT and FILESTATE attributes.

Two types of dialogue termination are available: orderly termination and abort termination. Note that only OSINATIVESERVICE, OSISESSIONSERVICE, and TCPIPNATIVESERVICE currently support orderly termination.

When orderly termination is used, the system steps the endpoints through a handshaking process so that no data is lost. All data sent during the dialogue is guaranteed to be delivered by the service provider. The dialogue is not terminated by a service provider until a confirmation request is received from the correspondent—that is, both sides must agree that no more data is to be sent when either endpoint terminates the dialogue.

Abort termination does not guarantee transmission of all data; the service provider might terminate the dialogue before all data is delivered. When an ABORT statement is issued by either endpoint, the service provider might or might not wait for confirmation of the ABORT before it terminates the dialogue. Service providers return a DATALOST (2) SUBFILEERROR if data might be lost during dialogue termination. Note that your program can take steps to ensure that no data is lost even though the service provider does not support orderly termination. See "Using ABORT Termination for Orderly Release" later in this section for more information.

If a subport close takes longer than 60 seconds to complete, the message

```
<filename> WAITING FOR CLOSE RESPONSE FROM HOST(s) :  <host list>
```

is displayed. The reason for the delay depends on the service but is generally related to network congestion or failure.

Example Syntax

ALGOL

```
CLOSE (<file name>[SUBFILE <subfile index>],
       CLOSEDISPOSITION=<close disposition>,<control
option>)
```

COBOL74

```
MOVE <subfile index> TO <file subfile control>
CLOSE <file name> WITH <control option>
  USING CLOSE-DISPOSITION OF <close disposition>
```

The syntax elements of the CLOSE statement are as follows:

Element	Purpose
File name	Identifies the port file on which the CLOSE operation is to be done.
Subfile index	Specifies the subfile or subfiles on which the CLOSE is to be done.
Close disposition	Identifies the type of termination. Either the ABORT or ORDERLY value can be used. ABORT is the default.
Control option	Specifies when the system returns control to the program. Either the WAIT or DONTWAIT (NO WAIT in COBOL74) value can be used.

Additional parameters or parameter values that might be supported by a particular port service are defined in the description of that service.

Understanding the CLOSE Disposition Parameter

The CLOSE disposition parameter of the CLOSE statement specifies the type of dialogue termination to be performed: orderly or abort. The default disposition is ABORT.

The CLOSE disposition parameter can have the following values:

Value	Meaning
ABORT	Termination of the subfile is immediate. A CLOSE ABORT is destructive; all pending input and output data are purged. When a CLOSE ABORT is invoked, the subfile moves to a CLOSEPENDING or CLOSED file state. In a CLOSEPENDING file state, the dialogue is considered closed, but the CLOSE operation is still ongoing. You cannot read from or write to a subfile in this file state. When the CLOSE operation completes, the subfile moves to the CLOSED file state. If a CLOSE ABORT is invoked on a subfile already in a CLOSEPENDING or CLOSED file state, a FILENOTOPENRSLT (30) CLOSE result is returned.

Value	Meaning
ORDERLY	An orderly termination procedure is performed on the subfile. The specific file states that a subfile moves through during orderly release are dependent on the dialogue termination procedure of the service. The file eventually goes to either CLOSED or back to an opened state (OPENED, BLOCKED, SHUTTINGDOWN, or URGENTDATAWAITING, for example). Note that some port services do not support orderly CLOSE operations.

CLOSE ABORT is the default disposition.

Understanding the CLOSE Control Option Parameter

Your program does not have to wait for a CLOSE operation to complete. The control option parameter of the CLOSE statement determines when control is returned to your program. The control option can have the values WAIT or DONTWAIT (NO WAIT in COBOL74). The default control option value is WAIT.

The effects of the control option are as follows:

Control Option	When Control Is Returned to the Program
WAIT	When control is returned is dependent on the disposition parameter. If the disposition is ABORT, control is returned to the program when the file state is CLOSED for all the subfiles being closed. If the disposition is ORDERLY, control is returned to the program after the correspondent endpoint has acknowledged the CLOSE request or when program participation is required to complete orderly release.
DONTWAIT (NO WAIT in COBOL74)	Control is returned to the program as soon as possible and after the file state has been updated to either CLOSED, CLOSEPENDING, or CLOSEREQUESTED (for orderly termination).

When your program exits the block in which a port file is declared before its subfiles are either CLOSED or CLOSEPENDING, an implicit CLOSE ABORT operation with a WAIT control option is performed on these subfiles.

Example 1

ALGOL
```
CLOSE(PORTF[SUBFILE 1]);
```

COBOL74
```
MOVE 1 TO PORTF-SUB.
CLOSE PORTF USING CLOSE-DISPOSITION OF ABORT.
```

This example closes subfile 1 of PORTF with a default CLOSE disposition of ABORT and a default control option of WAIT. Control is returned to the program when the CLOSE operation is complete and subfile 1 is in the CLOSED file state.

Example 2

ALGOL
```
CLOSE (PORTF[SUBFILE 1],DONTWAIT);
```

COBOL74
```
MOVE 1 TO PORTF-SUB.
CLOSE PORTF WITH NO WAIT USING CLOSE-DISPOSITION OF ABORT.
```

This example closes subfile 1 of PORTF with a default CLOSE disposition of ABORT and a control option of DONTWAIT (NO WAIT in COBOL74). Control is returned to the program as soon as possible, and the CLOSE operation continues in parallel with the program. The program is notified that the CLOSE operation is complete and the subfile is in a CLOSED file state through the CHANGEEVENT and FILESTATE attributes.

Understanding Correspondent-Initiated Dialogue Termination

Your program must be able to monitor and handle dialogue termination from the correspondent endpoint. Your program is informed of a correspondent-initiated CLOSE on a subfile through the FILESTATE and CHANGEEVENT file attributes.

When a CLOSE ABORT indication is received on a subfile from the correspondent endpoint after a dialogue has been established, the subfile moves to the DEACTIVATIONPENDING or DEACTIVATED file state. It moves to DEACTIVATIONPENDING if the subfile still has data in the READ queue. When the READ queue is empty, the subfile moves to the DEACTIVATED file state.

When the file state of a subfile changes to DEACTIVATED, your program must acknowledge by performing a CLOSE ABORT operation on the subfile. A CLOSE ABORT operation on a subfile in the DEACTIVATIONPENDING file state purges any data in the READ queue and closes the subfile.

When a CLOSE ABORT indication is received from the correspondent endpoint during dialogue establishment (the file state is OFFERED, AWAITINGOFFER, or OPENRESPONSEPLEASE, for example), dialogue establishment terminates with a DISCONNECTEDDURINGOPEN (17)SUBFILEERROR and the subfile is moved to the CLOSED file state.

Note: *Not true of a passive open.*

Example

ALGOL

```
DO
  BEGIN
  CASE WAIT(TIMEOUT,
            PORTF.CHANGEEVENT,
            PORTF.INPUTEVENT) OF
    BEGIN
    1: HANDLE_TIMEOUT;
    2: INX := PORTF.CHANGEDSUBFILE;
       CASE PORTF(INX).FILESTATE OF
         BEGIN
         .
         .
         .
         DEACTIVATED:
           END_TRANS(INX);
           CLOSE(PORTF[SUBFILE INX]);
         END;
    3: HANDLE_INPUT;
    END;
```

COBOL74

```
WAIT HOLD-TIME, ATTRIBUTE CHANGEEVENT OF PORTF,
     ATTRIBUTE INPUTEVENT OF PORTF GIVING WAIT-STATE.
GO TO HANDLE-TIMEOUT,
     HANDLE-ERROR,
     HANDLE-INPUT DEPENDING ON WAIT-STATE.

HANDLE-ERROR.
    IF ATTRIBUTE FILESTATE OF PORTF(INX) IS EQUAL TO
VALUE DEACTIVATED
        PERFORM END-TRANS
        CLOSE PORTF.
```

The program waits on CHANGEEVENT and INPUTEVENT. If a subfile transits to the DEACTIVATED file state, the procedure END-TRANS is called and the subfile is closed if END_TRANS does an explicit CLOSE; otherwise, the subfile stays in the DEACTIVATED file state.

Understanding Service Provider-Initiated Dialogue Aborts

A service provider can also abort a dialogue. Your program is notified of a provider-initiated abort through the FILESTATE and CHANGEEVENT file attributes. Your program handles service provider-initiated aborts in the same way as correspondent-initiated aborts.

Using ABORT Termination for Orderly Release

Not all services offer an orderly dialogue release mechanism. If the port file service you are using does not, your program must exchange handshakes with the correspondent endpoint before your program invokes a CLOSE ABORT operation in order to release the dialogue in an orderly manner.

Your program can ensure orderly release in the same way that the orderly release mechanism of some port services can guarantee data delivery. Your program, however, exchanges handshakes using data transfer operations instead of orderly release operations.

By agreeing that nothing is to be sent after a handshake is complete, your program can invoke a CLOSE ABORT operation to terminate the dialogue without destructive effect since there are no more data or primitives that can be disrupted. If the handshake is completed successfully prior to the ABORT operation, then the processes can determine that no data loss occurred.

Examples

ALGOL

```
REPLACE IOBUF BY "ORDER_END           ";
% ORDER_END IS LAST MESSAGE FROM THIS SUBFILE
WRITE(PORTF[SUBFILE 1],20,IOBUF);
ERR := READ(PORTF[SUBFILE 1],20,IOBUF);
IF ERR THEN
  % SUBFILE GOT DEACTIVATED?
  HANDSHAKE_ERR
ELSE
  % ORDER_ACKNOWLEDGE IS LAST MESSAGE FROM OTHER SIDE
  IF IOBUF = "ORDER_ACKNOWLEDGE   " THEN
    BEGIN
    DISPLAY "ORDER OK";
    CLOSE(PORTF[SUBFILE 1]);
    END
  ELSE
    PROGRAMDUMP;
```

COBOL74

```
PROCEDURE DIVISION
DECLARATIVES.
ERR-HANDLING SECTION.
   USE AFTER STANDARD EXCEPTION PROCEDURE ON PORTF.
BEGIN-ERR.
   PERFORM HANDSHAKE-ERR.
END DECLARATIVES.
   .
   .
   .
   MOVE "ORDER-END" TO IOBUF.
   WRITE PORTF-REC FROM IOBUF.
   READ PORTF RECORD INTO IOBUF.
   IF IOBUF IS EQUAL TO "ORDER-ACKNOWLEDGE"
      DISPLAY "ORDER OK"
      MOVE 1 TO PORTF-SUB
      CLOSE PORTF
   ELSE
      CALL SYSTEM DUMP.
```

In this example, the calling program opens a session to a warehouse program that takes item orders. After opening the dialogue and making its orders, the program commences with the termination handshake procedure (the code shown above). Only the calling program is allowed to release the dialogue. If the warehouse program or the system terminates the dialogue before the handshake procedure, the order is void. If the dialogue terminates during the handshake, the calling program needs to reopen a session to verify the order. If the handshake completes, no data is lost and the calling program can close the subfile.

Section 21
Understanding Port Services

A port service is a specific set of features and functions that a program can use in communicating with another program. Use the SERVICE file attribute to specify the port service you would like to use with your port file. Port services differ in the following ways:

- The types of control operations used for the port file. Control operations manage the port file. For example, a port service might offer additional dialogue establishment features.
- The types of data transfer operations performed with the port file.
- The set of file attributes used for the port file. Some port services support additional file attributes or additional values for file attributes.

The service provided by a port file is dependent on the underlying interprocess communications (IPC) environment or environments that support it. A port provider represents an IPC environment.

Although providers fulfill the same purpose, they vary from one another. One provider can offer IPC functions that another provider cannot provide identically, or cannot provide at all because of differences in an underlying network or lack of an underlying network. The SERVICE attribute value you select should reflect what provider environment or environments you plan to use. If you select a port service that can be provided by different IPC environments, the system selects the provider to use for each subfile when an OPEN or AWAITOPEN operation is invoked. If you want to use a specific provider for a subfile, identify the specific provider with the PROVIDERGROUP file attribute. Refer to the *File Attributes Reference Manual* for a detailed description of the PROVIDERGROUP attribute.

If you do not care which provider will be used for the dialogue, you can set the SERVICE file attribute value of your port file to BASICSERVICE, which contains a set of service functions supported by most providers. A program can use, or can potentially use, more than one port provider for interprocess communication.

The service and provider environment used by a subfile for the dialogue cannot change during the lifetime of the dialogue. You can modify the SERVICE attribute only when all subfiles of a port file are closed. The default value of SERVICE is BNANATIVESERVICE. The default value can be changed by a network operator command.

It is important to note that the SERVICE attribute is not a matching attribute. For example, if your subfile is a BASICSERVICE subfile, this does not imply that the matching endpoint will also be a BASICSERVICE subfile. In fact, the matching endpoint might not be an MCP environment subfile at all.

The SERVICE attribute merely specifies the set of functions and features your endpoint requires over the IPC environment you plan to use. If the IPC environment offers additional features to the service you are using, you should make a prior agreement with your correspondent to restrict feature use. Otherwise, an unsuccessful match can result, or your subfile dialogue can be aborted by the local system with a subfile error.

The following are brief descriptions of each port service:

- BASICSERVICE

  BASICSERVICE is a message-oriented port service. It provides basic dialogue establishment and termination, and READ and WRITE capabilities. Currently BASICSERVICE is supported by BNA Version 2, BNAOSI, and Local Port Provider (LPP).

- BNANATIVESERVICE

  BNANATIVESERVICE is a message-oriented port service. BNANATIVESERVICE is the service native to the BNA network environment. Currently BNANATIVESERVICE is supported by BNA Version 2, and LPP.

- NETBIOSSESSIONSERVICE

  NETBIOSSESSIONSERVICE is a message-oriented service offered over Novell NetWare local area networks (LANs) by way of the HLCN provider.

- OSINATIVESERVICE

  OSINATIVESERVICE offers a basic full-duplex, message-oriented service. OSINATIVESERVICE is the service native to the OSI network environment. It is provided by the BNA OSI network implementation.

- OSISESSIONSERVICE

  OSISESSIONSERVICE is a subset of the OSINATIVESERVICE that includes service functions up to the Session layer only. As with OSINATIVESERVICE, OSISESSIONSERVICE offers a basic full-duplex, message-oriented service; however application context and Presentation Layer functions do not apply.

- TCPIPNATIVESERVICE

  TCPIPNATIVESERVICE is the service native to the TCP network environment. This port service is a data-stream-oriented service.

- TCPNATIVESERVICE

  TCPNATIVESERVICE is the service native to the TCP network environment. This port service is a data-stream-oriented service.

  Note: *This service will be deimplemented in a future release.*

Tables 15–1 and 15–2 summarize the port attributes and port statements available with each service.

Section 22
Using BASICSERVICE

BASICSERVICE is a message-oriented service that is provided by the BNA Version 2, Local Port Provider (LPP), and MCP environment OSI implementations. You should use BASICSERVICE for your port file if you want a set of features and functions that are common to most providers.

File Attributes Supported by BASICSERVICE

The following file attributes are supported by BASICSERVICE:

ACTUALMAXRECSIZE	APPLICATIONGROUP	ATTERR
ATTVALUE	ATTYPE	AVAILABLE
AVAILABLEONLY	BLOCKEDTIMEOUT	BLOCKSTRUCTURE
CENSUS	CHANGEDSUBFILE	CHANGEEVENT
COMPRESSING	COMPRESSION	COMPRESSIONREQUESTED
COMPRESSIONCONTROL	CURRENTRECORDLENGTH	FILEEQUATED
FILENAME	FILESTATE	FRAMESIZE
INPUTEVENT	INTERACTIVEFILE	INTNAME
KIND	LASTSUBFILE	LFILENAME
LTITLE	MAXCENSUS	MAXSUBFILES
MYHOST	MYNAME	OUTPUTEVENT
PATHNAME	PROVIDERGROUP	REINITIALIZE
REQUESTEDMAXRECSIZE	RESULTLIST	SECURITYTYPE
SERVICE	STATE	SUBFILEERROR
YOURHOST	YOURNAME	

If an attribute is invalid for this service, the only value of the attribute considered valid is the default value. Invalid values are handled as described under "Setting Proper Attribute Values" in Section 15.

BASICSERVICE attributes that do not apply to a provider are ignored by that provider. Refer to Table 15–1 to identify the attributes that are not valid for a specific provider.

The attribute ACTUALMAXRECSIZE has the following restrictions for each provider implementation:

Provider	Range Allowed
BNA Version 2 remote dialogue	1 through 20000
BNA Version 2 local dialogue	1 through 65513
LPP	1 through 65513
MCP environment OSI	1 through 64512

Statements Supported by BASICSERVICE

Of the set of language statements pertaining to port files, the following are supported by BASICSERVICE:

- Attribute interrogation
- Attribute modification
- OPEN
  - Control option: WAIT/DONTWAIT (NO WAIT in COBOL74)/AVAILABLE
  - CONNECTTIMELIMIT
- AWAITOPEN
  - Control option: WAIT/DONTWAIT (NO WAIT in COBOL74)/AVAILABLE
  - CONNECTTIMELIMIT
- READ
  - Message
  - READ length specification
  - WAIT/DONTWAIT (NO WAIT in COBOL74)
- WRITE
  - Message
  - WRITE length specification
  - WAIT/DONTWAIT (NO WAIT in COBOL74)
- CLOSE
  - Close disposition: ABORT
  - Control option: WAIT/DONTWAIT (NO WAIT in COBOL74)

If you use a port statement not included in the preceding list, an UNSUPPORTEDPRIMITIVERSLT (168) result and an UNSUPPORTEDPRIMITIVE (41) SUBFILEERROR are returned. If you use a statement parameter or a statement parameter value not included in the previous list, an UNSUPPORTEDPARAMETERRSLT (128) result and an UNSUPPORTEDPARAMETER (18) SUBFILEERROR are returned.

File States Supported by BASICSERVICE

The following is a list of all the possible file states a port file can have using BASICSERVICE:

CLOSED	OFFERED
AWAITINGHOST	AWAITINGOFFER
OPENED	BLOCKED
SHUTTINGDOWN	CLOSEPENDING
DEACTIVATIONPENDING	DEACTIVATED

Figures 22–1 through 22–3 illustrate the file state transitions during the dialogue establishment phase, the data transfer phase, and the dialogue termination phase for BASICSERVICE. Note that a subfile could make multiple file state transitions between FILESTATE and CHANGEEVENT interrogations.

In the figures that follow, the following conventions apply:

- All user-initiated primitives, such as CLOSE ABORT, are in capital letters.
- Other events, such as dialogue not established, are enclosed in square brackets.

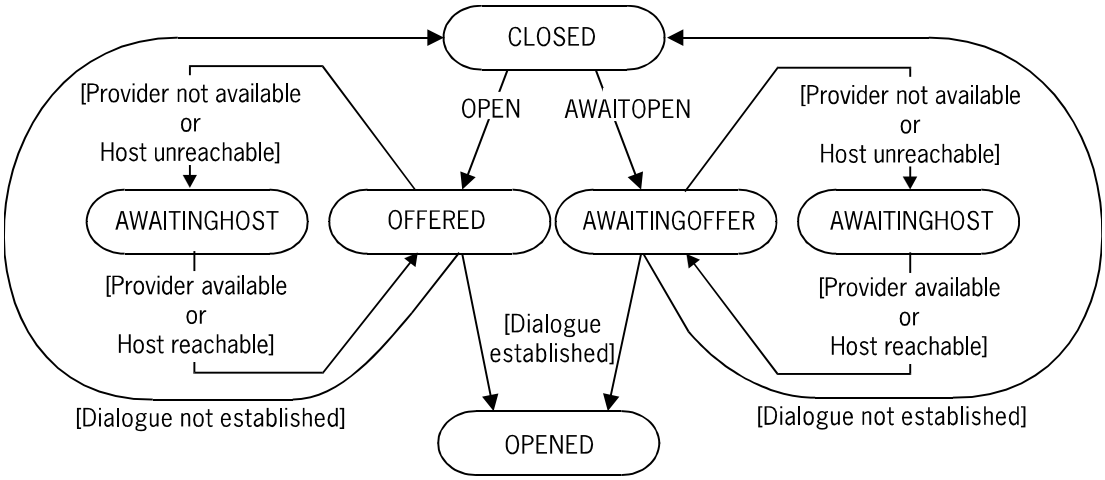


Figure 22–1. BASICSERVICE Dialogue Establishment File State Transitions

If the provider is BNAv2 and the BNAv2 command NW VALIDATE has been set to HOST STRICT (see the *Networking Operations Reference Manual*), and if the remote host has not been declared to BNAv2 with a NW ADD HOST command, then the FILESTA will not stay in AWAITINGHOST state, but will transit back to closed instead (via OFFERED).

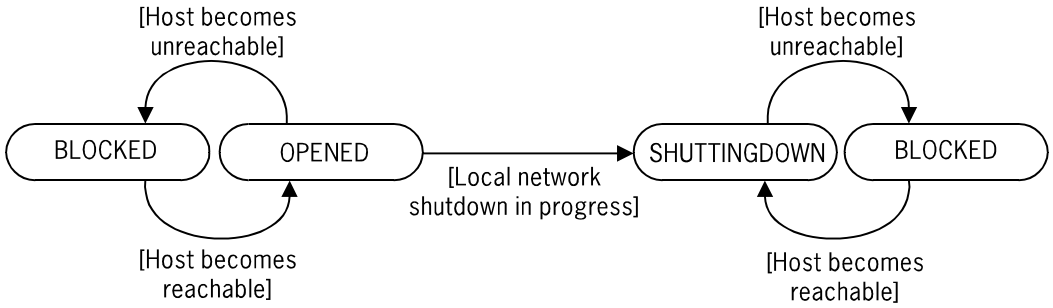


Figure 22–2. BASICSERVICE Probable File State Transitions during Data Transfer

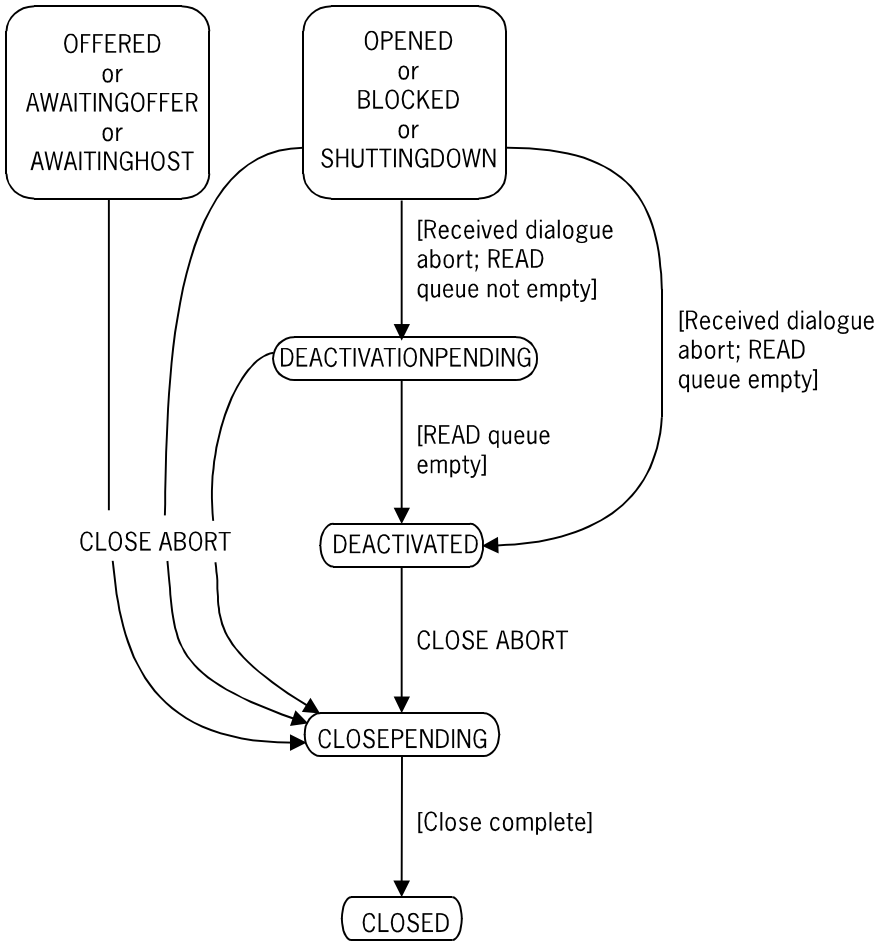


Figure 22–3. BASICSERVICE Dialogue Termination File State Transitions

If the provider is BNAv2 and the BNAv2 command NW VALIDATE has been set to HOST STRICT, and the remote host has not been declared to BNAv2 with an NW ADD HOST command, the FILESTATE will not stay in an AWAITINGHOST state. Instead, the FILESTATE transits back to the CLOSED state by way of the OFFERED state. Refer to the *Networking Operations Reference Manual* for more information on the NW VALIDATE command.

If the remote host becomes unreachable, the file state is marked as BLOCKED. The file state becomes OPENED if the remote host becomes reachable within a specific time. If the remote host does not become reachable within that time period, the system initiates deactivation of the dialogue.

Use the BLOCKEDTIMEOUT attribute to control the length of time that the subport is to remain blocked before the system initiates deactivation of the dialogue.

Preparing for Dialogue Establishment Using BASICSERVICE

The SECURITYTYPE attribute value must be PUBLIC and can default to PRIVATE if the default setting has not been changed by using the *NW NS SET MIGRATETOBASICSERVICE* system command.

BASICSERVICE uses the following matching attributes:

```
MYNAME
MYHOST
YOURNAME
YOURHOST
FILENAME
APPLICATIONGROUP
```

MYNAME, MYHOST, YOURNAME, and YOURHOST are matched as described earlier in Section 17, "Preparing Your Subfile for Dialogue Establishment." The FILENAME and APPLICATIONGROUP values of your subfile must match the FILENAME and APPLICATIONGROUP values of the correspondent endpoint.

The Local Port Provider (LPP) does not service dialogues when YOURHOST is set to foreign hosts. In this case, LPP returns an UNSUPPORTEDFUNCTION subfile error.

BNA 4.3 does not service dialogues when YOURHOST is set to local host (for example, local ports). In this case, BNA 4.3 returns an UNSUPPORTEDFUNCTION subfile error.

When you use BASICSERVICE over an OSI network environment, values for these attributes are associated with OSI endpoint addresses through the *NW ADD ENDPOINTNAME* or the *NW ADD OSIENDPOINTNAME* operator command. The endpoint addresses can be a full OSI address or an OSI Session user address. Communicating OSI endpoints must, however, use the same OSI layer.

Note that currently the BNA OSI network implementation can associate only one OSI address for each attribute set. Endpoints that are reachable through more than one OSI address must have an attribute set for each address. Matching is also affected by OSI parameters used for negotiating user message formats. See "Exchanging Data Using BASICSERVICE" in this section for more information.

Establishing a Dialogue Using BASICSERVICE

BASICSERVICE follows the dialogue establishment procedures described earlier in Section 18, "Establishing a Subfile Dialogue."

Dialogue establishment can be locally initiated through the OPEN statement or be correspondent-initiated through the AWAITOPEN statement. If both endpoints send dialogue requests, the provider attempts to resolve the colliding requests into one dialogue. The colliding requests might not result in a match.

Using the OPEN Statement with BASICSERVICE

The OPEN statement works as described under "Using the OPEN Statement" in Section 18, although the effects of the AVAILABLEONLY attribute vary slightly when used with different network environments.

The network-specific effects of the AVAILABLEONLY attribute on the dialogue establishment procedure for the OPEN operation for each network environment are as follows:

Network	Effect on Dialogue Establishment
BNA	The value of the AVAILABLEONLY attribute is transmitted with the dialogue request. If AVAILABLEONLY is FALSE, the OPEN operation is suspended until the correspondent host is reachable and the dialogue request can be sent. The receiving BNA host then saves the request if it cannot be matched with a subfile. If AVAILABLEONLY is TRUE, the OPEN operation fails if the dialogue request cannot be sent immediately, or if it cannot be matched by the receiving BNA host when the request is received.
OSI	When your port file is establishing a dialogue with another port file on a remote host, the AVAILABLEONLY attribute has no effect.

Using the AWAITOPEN Statement with BASICSERVICE

Refer to "Using the AWAITOPEN Statement" in Section 18 for an explanation of the AWAITOPEN statement.

Exchanging Data Using BASICSERVICE

BASICSERVICE follows the procedures for READ and WRITE operations described earlier in Section 19, "Exchanging Data," with the following considerations if you are using BASICSERVICE in an OSI network environment.

In an OSI network environment, the format of user messages is negotiated during dialogue establishment. The BASICSERVICE provider negotiates on behalf of the subfile an octet string format. On an OPEN operation, the provider proposes and must negotiate successfully the following values for OSI dialogue request parameters:

- Default Context. This defaults to null.
- Presentation Context Set. This defaults to
  - ACSE-1 abstract syntax, object ID {2 2 1 0 1}
  - NIST Octet String abstract syntax, object ID {1 3 14 8 2 1} (as defined by the National Institute of Science and Technology OSI Implementors' Agreements)

On an AWAITOPEN operation, the provider matches the BASICSERVICE subfile only to an incoming call that proposes these values for these parameters.

The way a READ operation is handled by the port service depends upon the file state of the subfile. Table 22–1 describes the way a READ operation is handled on a subfile in each file state supported by BASICSERVICE.

Table 22–1. Effects of File State on the READ Operation for BASICSERVICE

File State	Action
CLOSED	Implicit OPEN
OFFERED	EOF
AWAITINGOFFER	EOF
AWAITINGHOST	EOF
OPENED	If the value of CENSUS is greater than 0, returns input data; otherwise depends on WAIT/DONTWAIT (NO WAIT in COBOL74)
SHUTTINGDOWN	If the value of CENSUS is greater than 0, returns input data; otherwise depends on WAIT/DONTWAIT (NO WAIT in COBOL74)
BLOCKED	If the value of CENSUS is greater than 0, returns input data; otherwise depends on WAIT/DONTWAIT (NO WAIT in COBOL74)

Table 22–1. Effects of File State on the READ Operation for BASICSERVICE

File State	Action
DEACTIVATIONPENDING	Returns input data; when READ queue is empty, subfile moves to DEACTIVATED
CLOSEPENDING	EOF
DEACTIVATED	EOF

The result of the WRITE operation depends upon the file state of the subfile you are writing to. Table 22–2 explains the results of the WRITE operation on a subfile in each of the file states supported by BASICSERVICE.

Table 22–2. Effects of File State on the WRITE Operation for BASICSERVICE

File State	Action
CLOSED	Implicit OPEN
OFFERED	EOF
AWAITINGOFFER	EOF
AWAITINGHOST	EOF
OPENED	If OUTPUTEVENT is in the happened state, then WRITE (OK); otherwise, depends on WAIT/DONTWAIT (NO WAIT in COBOL74)
SHUTTINGDOWN	If OUTPUTEVENT is in the happened state, then WRITE (OK); otherwise, depends on WAIT/DONTWAIT (NO WAIT in COBOL74)
BLOCKED	If OUTPUTEVENT is in the happened state, then WRITE (OK); otherwise, depends on WAIT/DONTWAIT (NO WAIT in COBOL74)
DEACTIVATIONPENDING	EOF
DEACTIVATED	EOF
CLOSEPENDING	EOF

Closing a Dialogue Using BASICSERVICE

BASICSERVICE does not support orderly dialogue release. BASICSERVICE provides abort termination through the CLOSE ABORT statement, as described earlier in Section 20, "Closing a Dialogue."

Section 23
Using OSINATIVESERVICE

OSINATIVESERVICE is the interprocess communication service native to the OSI network environment. You send OSI request or response messages on the port subfile dialogue by invoking port statements like OPEN, and you are notified of incoming OSI messages through the CHANGEEVENT and INPUTEVENT attributes. OSINATIVESERVICE offers a basic full-duplex, message-oriented service.

The fields on OSI protocol messages are made accessible to you through port statement parameters and attributes. Not all fields, however, are currently supported. Fields that are not available to you are sent without values when the OSI protocol message is transmitted. If these fields contain values when an OSI protocol message is received from a correspondent, the system discards the values.

The following subsections summarize the port file attributes and port statements supported by OSINATIVESERVICE.

File Attributes Supported by OSINATIVESERVICE

The following file attributes are supported by OSINATIVESERVICE. Table 15–1 contains the list of attributes supported by each network service.

ACTUALMAXRECSIZE	APPLICATIONCONTEXT	APPLICATIONGROUP
ATTERR	ATTVALUE	ATTYPE
AVAILABLE	AVAILABLEONLY	BLOCKEDTIMEOUT
BLOCKSTRUCTURE	CENSUS	CHANGEDSUBFILE
CHANGEEVENT	COMPRESSING	CURRENTRECORDLENGTH
FILEEQUATED	FILENAME	FILESTATE
FRAMESIZE	INPUTEVENT	INTERACTIVEFILE

INTNAME	KIND	LASTSUBFILE
LFILENAME	LTITLE	MAXCENSUS
MAXSUBFILES	MYHOST	MYNAME
OUTPUTEVENT	PATHNAME	PROVIDERGROUP
REINITIALIZE	REQUESTEDMAXRECSIZE	RESULTLIST
SECURITYTYPE	SERVICE	STATE
SUBFILEERROR	YOURHOST	YOURNAME

In addition, the following file attributes are supported by OSINATIVESERVICE:

PORTSEGMENTIO	YOURNSAPA	YOURPRESENTATIONSEL
YOURSESSIONSEL	YOURTRANSPORTSEL	

If an attribute is invalid for this service, the only value of the attribute considered valid is the default value. Invalid values are handled as described under "Setting Proper Attribute Values" in Section 15. Be aware that the SECURITYTYPE attribute value must be PUBLIC and can default to PRIVATE if the default setting has not been changed by using the *NW NS SET MIGRATETOBASICSERVICE* system command.

If the attributes YOURHOST and YOURNAME are null strings on an OPEN operation, the operation fails with a BADATTRIBUTESFOROPEN (13) SUBFILEERROR. If the MYNAME attribute is a null string on an OPEN or AWAITOPEN operation, the operation fails with either a BADATTRIBUTESFOROPEN (13) or BADATTRIBUTESFORAWAITOPEN (34) SUBFILEERROR.

The ACTUALMAXRECSIZE attribute can have a value of 1 through 64512 for the BNA OSI network implementation.

Statements Supported by OSINATIVESERVICE

Of the set of language statements pertaining to port files, the following are supported by OSINATIVESERVICE:

- Attribute interrogation
- Attribute modification
- OPEN
  - Control option: WAIT/DONTWAIT (NO WAIT in COBOL74)/AVAILABLE
  - CONNECTTIMELIMIT
  - Associated data
  - Associated data length specification
- AWAITOPEN
  - Control option: WAIT/DONTWAIT (NO WAIT in COBOL74)/AVAILABLE
  - CONNECTTIMELIMIT
  - PARTICIPATE
- READ
  - Message
  - READ length specification
  - WAIT/DONTWAIT (NO WAIT in COBOL74)
- WRITE
  - Message
  - WRITE length specification
  - WAIT/DONTWAIT (NO WAIT in COBOL74)
  - MOREDATA
- RESPOND
  - Respond type: ACCEPTOPEN/REJECTOPEN/ACCEPTCLOSE
  - Associated data
  - Associated data length specification
- CLOSE
  - Close disposition: ABORT/ORDERLY
  - Control option: WAIT/DONTWAIT (NO WAIT in COBOL74)
  - Associated data
  - Associated data length specification

If you use a port statement not included in the preceding list, an UNSUPPORTEDPRIMITIVERSLT (168) result and an UNSUPPORTEDPRIMITIVE (41) SUBFILEERROR are returned. If you use a statement parameter or a statement parameter value not included in the previous list, an UNSUPPORTEDPARAMETERRSLT (128) and an UNSUPPORTEDPARAMETER (18) SUBFILEERROR are returned.

Understanding the ASSOCIATEDDATA Parameter of OSINATIVESERVICE

Your program can send user data with control messages exchanged during dialogue establishment and dialogue termination. Your program can do this through the ASSOCIATEDDATA parameter of the OPEN, CLOSE, and RESPOND statements.

Your program receives associated data through the READ queue. The result of your READ statement tells you if the data returned by the READ operation is associated data.

The maximum length of ASSOCIATEDDATA that can be sent primarily depends upon the implementation restrictions of both network providers, as well as the number of bytes needed to encode certain dialogue parameters. The OSINATIVESERVICE provider can guarantee support for 7K bytes of associated data. A correspondent OSI implementation might support a protocol version that can guarantee only up to 200 bytes of associated data and no data at all on an abort. The lesser restriction applies.

Associated data lengths greater than these limits can be sent and received by your program. However, they are not guaranteed by the MCP environment OSI provider. It is the responsibility of an application using OSI to specify its requirements for maximum associated data length, and hence its requirements on OSI network implementations, to other applications that want to communicate with it.

The preceding limits can be guaranteed only if the application context object ID has

- At most, 6 components
- Each component in the range 0 through 127 characters

If the associated data is too long, the OPEN, CLOSE ORDERLY, or RESPOND operation fails with an ASSOCIATEDDATATOOLONGRSLT (156) result and an ASSOCIATEDDATATOOLONG (32) SUBFILEERROR. If the associated data is too long on a CLOSE ABORT operation, the ABORT operation continues with the warning WARNABORTDATAIGNOREDRSLT (17) and a WARNABORTDATAIGNORED (48) SUBFILEERROR, and the ABORT is sent without associated data.

File States Supported by OSINATIVESERVICE

The following is a list of all the possible file states a port file can have using OSINATIVESERVICE:

CLOSED	CLOSEREQUESTED
OFFERED	CLOSERESPONSEPLEASE
AWAITINGHOST	CLOSEDINPUTPENDING
AWAITINGOFFER	CLOSECOLLRESPONSEPLEASE
OPENRESPONSEPLEASE	DEACTIVATIONPENDING
OPENED	DEACTIVATED
SHUTTINGDOWN	
CLOSEPENDING	

Figures 23–1 through 23–4 illustrate the file state transitions during the dialogue establishment phase, the data transfer phase, and the dialogue termination phase for OSINATIVESERVICE. Note that a subfile can have multiple file state transitions between CHANGEEVENT and FILESTATE interrogations.

In the figures that follow, the following conventions apply:

- All user-initiated primitives, such as CLOSE ABORT, are in capital letters.
- Other events, such as dialogue not established, are enclosed in square brackets.

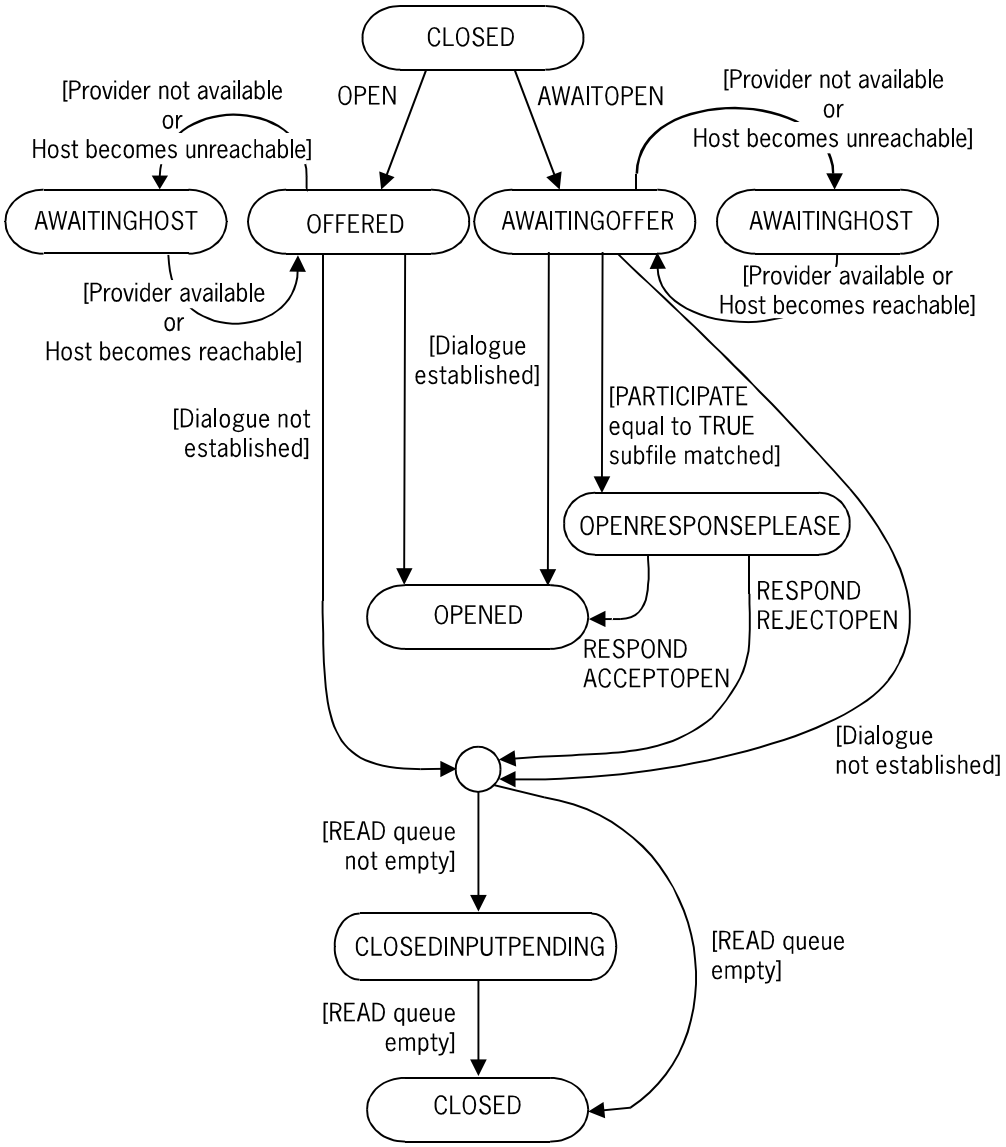


Figure 23–1. OSINATIVESERVICE Dialogue Establishment File State Transitions

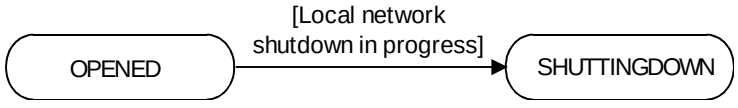


Figure 23–2. OSINATIVESERVICE Probable File State Transitions during Data Transfer

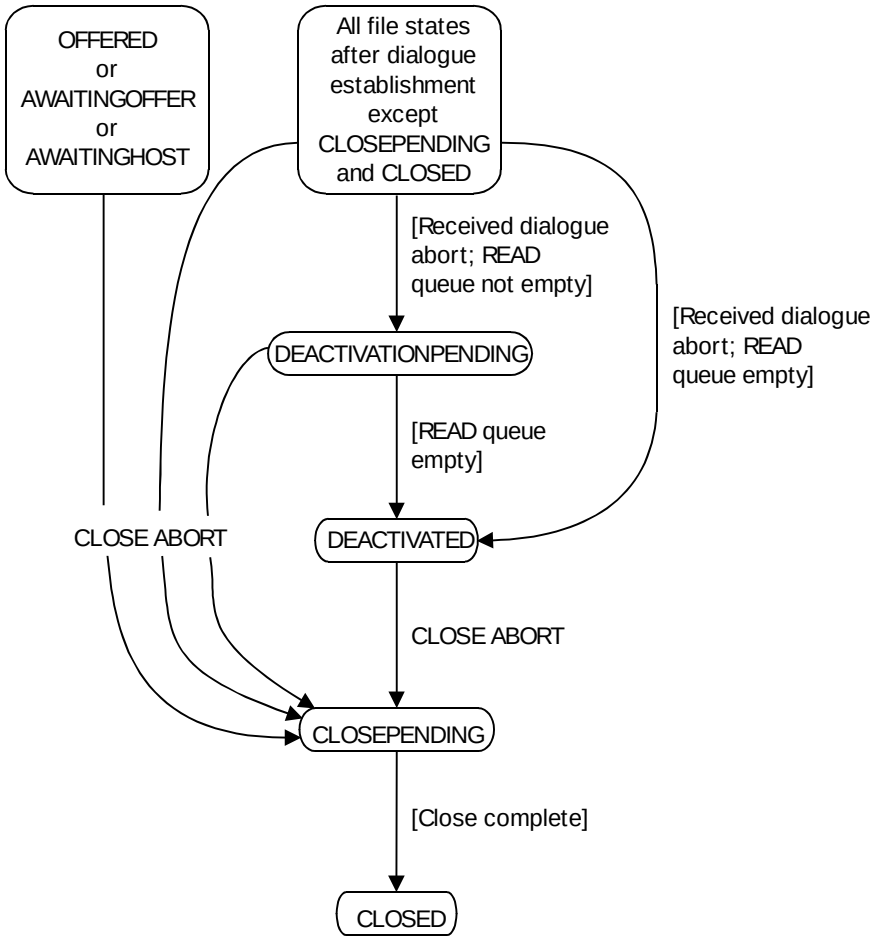


Figure 23–3. OSINATIVESERVICE Dialogue Termination File State Transitions

If the remote host becomes unreachable, the file state is marked as BLOCKED. The file state becomes OPENED if the remote host becomes reachable within a specific time. If the remote host does not become reachable within that time period, the system initiates deactivation of the dialogue.

Use the BLOCKEDTIMEOUT attribute to control the length of time that the subport is to remain blocked before the system initiates deactivation of the dialogue.

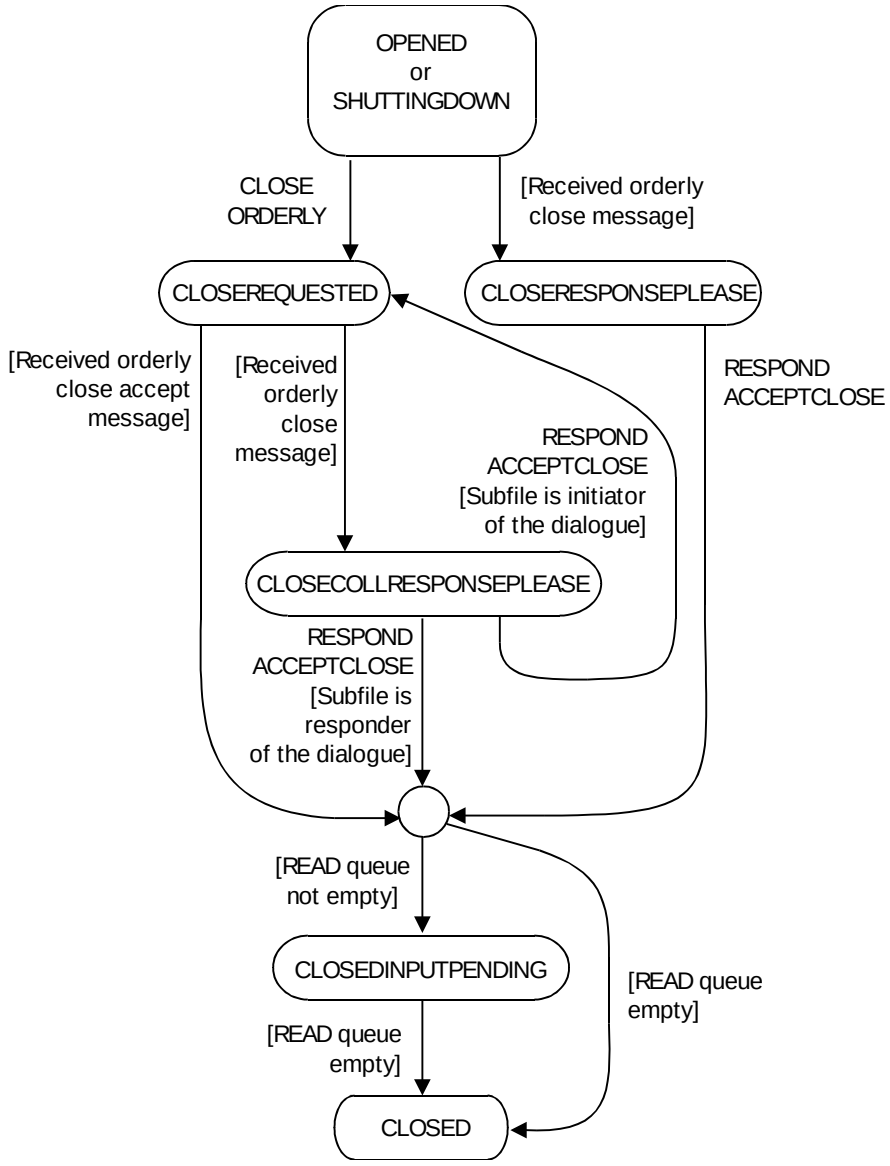


Figure 23–4. OSINATIVESERVICE Orderly Termination File State Transitions

Preparing for Dialogue Establishment Using OSINATIVESERVICE

The following OSI dialogue-establishment fields are used for matching:

OSI Dialogue Request/Response Fields	Attribute for OSI 1.2 or 3.0 Release	Attribute for OSI 3.1 and Later Releases
OSI address (your subfile)	MYNAME MYHOST FILENAME APPLICATIONGROUP	MYNAME MYHOST FILENAME APPLICATIONGROUP
OSI address (correspondent)	YOURNAME YOURHOST FILENAME APPLICATIONGROUP	YOURNAME YOURHOST FILENAME APPLICATIONGROUP YOURNSAPA YOURPRESENTATIONSEL YOURSESSIONSEL YOURTRANSPORTSEL
Presentation Context Set	See "Exchanging Data Using OSINATIVESERVICE" later in this section.	See "Exchanging Data Using OSINATIVESERVICE" later in this section.
Default Context	See "Exchanging Data Using OSINATIVESERVICE" later in this section.	See "Exchanging Data Using OSINATIVESERVICE" later in this section.

OSI endpoint addresses, both local and remote, are statically declared to the system through the *NW ADD ENDPOINTNAME* operator command. Through this command, an OSI endpoint address is declared and mapped into a NAME, HOST, FILENAME, and APPLICATIONGROUP attribute set. If your application program is using the OSI 3.1 or later release, you can use the YOURNSAPA, YOURPRESENTATIONSEL, YOURSESSIONSEL, and YOURTRANSPORTSEL file attributes to directly identify the network end-point addresses of the correspondent application that might not be declared in the static network configuration.

If your application program is using the OSI 1.2 or 3.0 release, the network environment matches MYNAME, MYHOST, YOURNAME, and YOURHOST as described earlier in Section 17, "Preparing Your Subfile for Dialogue Establishment." The FILENAME and APPLICATIONGROUP values of your subfile must match the FILENAME and APPLICATIONGROUP values of the correspondent endpoint.

If your application program is using the OSI 3.1 or later release and you specify YOURPRESENTATIONSEL, YOURSESSIONSEL, YOURTRANSPORTSEL, and YOURNSAPA values, these attributes are matched even if you specify values for the YOURNAME, YOURHOST, FILENAME, and APPLICATIONGROUP values. If you do not specify YOURPRESENTATIONSEL, YOURSESSIONSEL, YOURTRANSPORTSEL, and

YOURNSAPA values, YOURNAME, YOURHOST, FILENAME, and APPLICATIONGROUP values are matched.

Note that currently the BNA OSI network implementation associates only one OSI address for an endpoint for each NAME, HOST, FILENAME, and APPLICATIONGROUP attribute set. Endpoints that are reachable through more than one OSI address (because of multiple network addresses, for example) must have an attribute set for each address.

Establishing a Dialogue Using OSINATIVESERVICE

An OSI dialogue request is transmitted when you invoke an OPEN operation on a subfile. When you invoke an AWAITOPEN operation, your subfile is made available to be matched to an incoming dialogue request. If both endpoints send dialogue requests, the colliding dialogue requests are considered as two separate requests for dialogue. That is, incoming dialogue requests are not matched to subfiles on which you have invoked an OPEN operation.

Using the OPEN Statement with OSINATIVESERVICE

When you invoke an OPEN statement, a dialogue request is sent to the OSI endpoint address named by the YOURNAME, YOURHOST, FILENAME, and APPLICATIONGROUP attributes. You can send additional dialogue parameters with the request, by using attributes or parameters of the OPEN statement. Some dialogue parameters are negotiated by the dialogue endpoints. The values of other dialogue parameters, like associated data, are merely exchanged. See "Understanding Negotiation during Dialogue Establishment" later in this section for more information on this topic.

Example Syntax

ALGOL

```
OPEN (<file name>[SUBFILE <subfile index>],<control
option>,
      <connect time limit option>,<associated data
option>)
```

COBOL74

```
MOVE <subfile index> TO <file subfile control>.
OPEN <control option> <file name> USING
     <connect time limit>
     <associated data option>
```

In addition to the file name, subfile index, control option, and connect time limit option, OSINATIVESERVICE also supports the associated data option. That option enables you to send user data with the dialogue request and consists of the following two elements:

Element	Purpose
ASSOCIATEDDATALENGTH	Specifies the length of the user data to be sent
ASSOCIATEDDATA	Specifies the user data to be sent

Understanding the AVAILABLEONLY File Attribute Using OSINATIVESERVICE

As discussed in "Understanding the AVAILABLEONLY File Attribute for OPEN" in Section 18, the AVAILABLEONLY attribute specifies whether the OPEN operation fails or suspends when a dialogue cannot currently be established with the correspondent endpoint.

If your port file is communicating with another port file on the local system, the following actions will occur depending on the value of the AVAILABLEONLY attribute:

- If AVAILABLEONLY is set to FALSE, the OPEN operation is suspended until a dialogue is established, or until the CONNECTTIMELIMIT value is exceeded.
- If AVAILABLEONLY is set to TRUE and no match is found, the OPEN operation fails, your program resumes, and a NOFILEFOUND (4) SUBFILEERROR is returned and the subfile is not available for subsequent matching.

When your port file is communicating with another port file on a remote host, the AVAILABLEONLY attribute has no effect.

Understanding the OPEN Control Option Parameter for OSINATIVESERVICE

OSINATIVESERVICE allows the following values for the control option parameter of the OPEN statement:

- WAIT
- DONTWAIT (NO WAIT in COBOL74)
- AVAILABLE

These control options function as described earlier under "Understanding the OPEN Control Option Parameter" in Section 18.

Understanding the CONNECTTIMELIMIT Parameter for OSINATIVESERVICE

This parameter functions the same as described earlier under "Understanding the OPEN CONNECTTIMELIMIT Parameter" in Section 18.

Understanding the OPEN ASSOCIATEDDATA Parameter for OSINATIVESERVICE

You can send user data with a dialogue request when using OSINATIVESERVICE. You can do so through the ASSOCIATEDDATA parameter of the OPEN statement.

No user data is sent if the associated data parameter is absent or if the associated data length is 0 (zero).

Example 1

ALGOL
```
OPEN(PORTF[SUBFILE 1],ASSOCIATEDDATA="MYDATA");
```

COBOL74
```
MOVE 1 TO PORTF-SUB.
OPEN I-O PORTF USING ASSOCIATED-DATA OF "MYDATA".
```

This example opens subfile 1 of PORTF. When the dialogue request is sent, the information "MYDATA" is sent to the correspondent process as user data.

Example 2

ALGOL
```
OPEN(PORTF[SUBFILE 1],
     ASSOCIATEDDATALENGTH=14,
     ASSOCIATEDDATA=PTR);
```

COBOL74
```
MOVE 1 TO PORTF-SUB.
OPEN I-O PORTF USING
     associated-DATA-LENGTH OF 14
     ASSOCIATED-DATA OF PORTF-MSG.
```

This example opens subfile 1 of PORTF. When the dialogue request is sent, the data pointed to by PTR for 14 characters is sent to the correspondent process as user data.

Using the AWAITOPEN Statement with OSINATIVESERVICE

Invoke an AWAITOPEN statement to make a subfile available to accept dialogue requests from other programs. When an AWAITOPEN is invoked, the local system is searched for an outstanding dialogue request that matches the subfile. If no matching dialogue request has already been received, the AWAITOPEN fails or is saved for future matching, depending upon the value of the AVAILABLEONLY attribute.

When the system receives a dialogue request, it matches the request to a subfile that invoked an AWAITOPEN statement. If the system cannot match the request, it keeps the request for future matching.

The following fields from the dialogue request received from the correspondent endpoint are used by the OSINATIVESERVICE provider when it matches the request to a subfile:

- The calling endpoint address
- The called endpoint address
- The presentation context set
- The default presentation context

The addresses are looked up on the local system to retrieve matching values for the NAME, HOST, FILENAME, and APPLICATIONGROUP attributes. These values are used to match to the attributes of the subfile. The subfile is matched as described in "Preparing for Dialogue Establishment Using OSINATIVESERVICE" earlier in this section.

Example Syntax

ALGOL

```
AWAITOPEN (<file name>[SUBFILE <subfile index>],
           <control option>, <participate option>,
           <connect time limit option>)
```

COBOL74

```
MOVE <subfile index> TO <file subfile control>.
AWAIT-OPEN <control option> <file name> USING
     <participate option>
     <connect time limit option>.
```

In addition to the file name, subfile index, control option, and connect time limit parameters of the general port AWAITOPEN statement, OSINATIVESERVICE also supports the PARTICIPATE option. This parameter enables you to conditionally or unconditionally establish a dialogue when the subfile is matched to an incoming dialogue request. See "Understanding the AWAITOPEN PARTICIPATE Parameter for OSINATIVESERVICE" later in this section for more information.

Understanding the AWAITOPEN Control Option Parameter for OSINATIVESERVICE

OSINATIVESERVICE allows the following values for the AWAITOPEN control option:

- WAIT
- DONTWAIT (NO WAIT in COBOL74)
- AVAILABLE

These options function as described earlier under "Understanding the AWAITOPEN Control Option Parameter" in Section 18.

Understanding the AWAITOPEN PARTICIPATE Parameter for OSINATIVESERVICE

OSINATIVESERVICE provides you with the ability to conditionally establish a dialogue on an AWAITOPEN operation. The PARTICIPATE option of the AWAITOPEN statement specifies if the program uses conditional or unconditional dialogue establishment when a dialogue request is received. The PARTICIPATE option can be either TRUE or FALSE.

If PARTICIPATE is FALSE when a request for dialogue establishment is received, the request is accepted and the subfile is moved to an OPENED file state. The values of negotiable attributes are unconditionally accepted and no user data is sent with the response.

If PARTICIPATE is TRUE when a request for dialogue establishment is received, the subfile is moved to the OPENRESPONSEPLEASE file state. In this state, the program can interrogate or modify certain attributes for negotiation or for transmission on the response. The program accepts or rejects the call through the RESPOND ACCEPTOPEN/REJECTOPEN statement.

If there is user data in the READ queue that came with the dialogue request (OPENDATA), the program can read the data before it responds. If you want to send user data with the response, use the associated data parameter of the RESPOND statement. If the program accepts the request, the subfile moves to the OPENED file state. If the program rejects the request, the subfile moves to the CLOSED or CLOSEDINPUTPENDING file state.

PARTICIPATE defaults to FALSE.

Example

ALGOL

```
AWAITOPEN(PORTF[SUBFILE 1],PARTICIPATE=TRUE);
```

COBOL74

```
MOVE 1 TO PORTF-SUB.
AWAIT-OPEN PORTF USING PARTICIPATE.
```

When subfile 1 of PORTF is matched to an incoming dialogue request, the subfile transits to the OPENRESPONSEPLEASE file state. In this file state, the program can interrogate attributes of the matching dialogue request, set negotiable attributes, and READ any associated data that might have arrived with the request.

Using the RESPOND Statement with OSINATIVESERVICE

OSINATIVESERVICE allows a program to respond positively or negatively to a request indication received from a correspondent endpoint. The nature of the request is contained in the FILESTATE attribute. The program uses the RESPOND statement to respond to dialogue establishment requests (OPENRESPONSEPLEASE file state) and dialogue termination requests (CLOSERESPONSEPLEASE and CLOSECOLLRESPONSEPLEASE file states) from the correspondent endpoint. The RESPOND statement returns a RESPOND result.

Example Syntax

ALGOL

```
RESPOND (<file identifier>[SUBFILE<subfile index>],
         RESPONDTYPE=<response type>,<associated data
option>)
```

COBOL74

```
MOVE <subfile index> TO <file subfile control>.
RESPOND <file name> WITH RESPONSE-TYPE OF <response type>
     USING <associated data option>.
```

The syntax elements are as follows:

Element	Purpose
File identifier	Specifies the name of the port file on which the RESPOND operation is to be performed.
Subfile index	Indicates the subfile that the response is for. A subfile index of 0 (zero) is currently not allowed.
Response type	Specifies the type of response with one of the following values: ACCEPTOPEN, REJECTOPEN, or ACCEPTCLOSE. The response type is described under "Understanding the Response Type Parameter" in this section.
Associated data option	Specifies that data is being sent with the response. This option consists of two elements: ASSOCIATEDDATALENGTH. This element specifies the length of user data to be sent. ASSOCIATEDDATA. This element specifies the user data to be sent. See "Understanding the ASSOCIATEDDATA Parameter of OSINATIVESERVICE" earlier in this section for more information.

Example 1

ALGOL

```
RESPOND(PORTF[SUBFILE 1], RESPONDTYPE=ACCEPTOPEN);
```

COBOL74

```
MOVE 1 TO PORTF-SUB.
RESPOND PORTF WITH RESPONSE-TYPE OF ACCEPT-OPEN.
```

In this example the program is responding to an offer for subfile 1 on port file PORTF by accepting the offer.

Example 2

ALGOL
```
RESPOND(PORTF[SUBFILE 1],
        RESPONDTYPE=ACCEPTOPEN,
        ASSOCIATEDDATA="INFO");
```

COBOL74
```
MOVE 1 TO PORTF-SUB.
RESPOND PORTF WITH RESPONSE-TYPE OF ACCEPT-OPEN
     USING ASSOCIATED-DATA OF "INFO".
```

The program is responding to an offer for subfile 1 of port file PORTF by accepting the offer. The information "INFO" is sent as user data with the positive response to the dialogue request.

Understanding the Response Type Parameter

You can assign the response type parameter the value ACCEPTOPEN, REJECTOPEN, or ACCEPTCLOSE.

Your program can respond to a dialogue establishment request if your program invoked an AWAITOPEN operation with PARTICIPATE set to TRUE. When a dialogue establishment request is received, the file state moves to OPENRESPONSEPLEASE. Your program can agree to the dialogue by assigning the value ACCEPTOPEN to the response type parameter of the RESPOND statement. Your program can reject the dialogue request by assigning the value REJECTOPEN to the RESPOND response type parameter.

Your program can respond to a dialogue orderly close request when the subfile file state is CLOSERESPONSEPLEASE or CLOSECOLLRESPONSEPLEASE. Your program can agree to the close request by assigning the value ACCEPTCLOSE to the response type. Note that a response type of REJECTCLOSE is not yet supported.

Example 1

ALGOL
```
RESPOND(PORTF[SUBFILE 1],RESPONDTYPE=ACCEPTOPEN);
```

COBOL74
```
MOVE 1 TO PORTF-SUB.
RESPOND PORTF WITH RESPONSE-TYPE OF ACCEPT-OPEN.
```

In this example, subfile 1 of PORTF is in the file state OPENRESPONSEPLEASE. The program is responding to the OPEN request by accepting the offer.

Example 2

ALGOL
```
RESPOND(PORTF[SUBFILE 1],RESPONDTYPE=REJECTOPEN);
```

COBOL74
```
MOVE 1 TO PORTF-SUB.
RESPOND PORTF WITH RESPONSE-TYPE OF REJECT-OPEN.
```

In this example, subfile 1 of PORTF is in the file state OPENRESPONSEPLEASE. The program is responding to the OPEN request by rejecting the offer.

Example 3

ALGOL
```
RESPOND(PORTF[SUBFILE 1],RESPONDTYPE=ACCEPTCLOSE);
```

COBOL74
```
MOVE 1 TO PORTF-SUB.
RESPOND PORTF WITH RESPONSE-TYPE OF ACCEPT-CLOSE.
```

In this example, subfile 1 of PORTF is in the file state CLOSERESPONSEPLEASE or CLOSECOLLRESPONSEPLEASE. The program is positively responding to the request for an orderly close.

Understanding Negotiation during Dialogue Establishment

The initiator of a dialogue can propose values for negotiable dialogue parameters in an OSI dialogue request. The correspondent can negotiate the values of dialogue parameters in its response to the dialogue request.

If the subfile is awaiting a dialogue request, it can participate in negotiation only if your program invoked the AWAITOPEN statement with PARTICIPATE set to TRUE. The program can then modify the negotiable parameters when the subfile is in the OPENRESPONSEPLEASE file state before invoking the RESPOND statement. If PARTICIPATE is FALSE, all negotiable dialogue parameters of a matching dialogue request are automatically accepted.

Currently, the only dialogue parameter that you can negotiate is the application context parameter. See "Exchanging Data Using OSINATIVESERVICE" in this section for information on default context and presentation context set handling.

The application context is an explicitly defined set of application service elements, options, and any other information necessary for two endpoints to interact in a particular dialogue. Application context is a mandatory dialogue parameter.

A dialogue application context is proposed on a dialogue request. The responder can respond with the same context, or can counter-propose another application context. The final application context is that picked by the responding program.

When your program invokes an OPEN statement, the APPLICATIONCONTEXT attribute contains the context that your program proposes in its dialogue request. If APPLICATIONCONTEXT is null (the default value), the application context defined by the National Institute of Science and Technology (NIST), "Nil Application Context," is proposed for your program, with the object ID {1 3 14 8 1 1}. When your program receives the response to the dialogue request, APPLICATIONCONTEXT contains the context proposed by the correspondent program. If APPLICATIONCONTEXT was changed in the response, the OPEN operation returns a WARNCONTEXTCHANGED (37) SUBFILEERROR.

When your program invokes an AWAITOPEN statement, the system sets APPLICATIONCONTEXT to a null string. If it was not already a null string, the SUBFILEERROR attribute is set to the WARNCONTEXTIGNORED (47) warning and the AWAITOPEN operation continues. When your program receives a dialogue request, APPLICATIONCONTEXT contains the context proposed by the correspondent program. If your program invoked the AWAITOPEN operation with PARTICIPATE set to TRUE, your program has the opportunity to counter-propose an application context by interrogating and setting the value of APPLICATIONCONTEXT while in the OPENRESPONSEPLEASE file state. This value is then sent by the service provider when your program invokes the RESPOND operation.

Examples

ALGOL

```
REPLACE PORTF(1).APPLICATIONCONTEXT
  BY "1 3 14 8 1 1.";
CASE OPEN(PORTF[SUBFILE 1]) OF
  BEGIN
    .
    .
    .
  WARNCONTEXTCHANGEDRSLT:
    IF PORTF(1).APPLICATIONCONTEXT
      NEQ "1 3 14 8 1 1." THEN
      CLOSE(PORTF[SUBFILE 1]);
    .
    .
    .
  END;
```

COBOL74

```
CHANGE ATTRIBUTE APPLICATIONCONTEXT OF PORTF(1)
     TO "1 3 14 8 1 1.".
MOVE 1 TO PORTF-SUB.
OPEN I-O PORTF.
IF PORTF-FS IS NOT EQUAL TO "00"
     IF ATTRIBUTE SUBFILEERROR OF PORT(1) IS EQUAL TO
                              VALUE (WARNCONTEXTCHANGED)
     IF ATTRIBUTE APPLICATIONCONTEXT OF PORTF(1) IS NOT
EQUAL TO
                                                    "1 3 14
8 1 1."
          MOVE 1 TO PORTF-SUBF.
CLOSE PORTF.
```

In this program example, the application can support only the application context defined by the object identifier "1 3 14 8 1 1.".

Exchanging Data Using OSINATIVESERVICE

When using an OSI network environment, the format of user messages is negotiated during dialogue establishment. The OSINATIVESERVICE provider negotiates an octet string format for the subfile.

On an OPEN operation, the provider proposes and must negotiate successfully the following values for OSI dialogue request parameters:

- Default context. This must be null.
- Presentation context set. This must contain:
  - ACSE-1 abstract syntax, object ID {2 2 1 0 1}.
  - NIST Octet String abstract syntax, object ID {1 3 14 8 2 1} (as defined by the National Institute of Science and Technology OSI Implementors' Agreements).

On an AWAITOPEN operation, the provider matches the OSINATIVESERVICE subfile only to an incoming call that proposes these values for these parameters.

Any associated data received from the correspondent is placed in the READ queue. When you invoke a READ statement, the result of the READ operation tells you if the data returned by the READ operation is associated data—for example, OPENDATA or OPENRESPONSEDATA.

The way a READ operation is handled by the port service depends upon the file state of the subfile. Table 23–1 describes the way a READ operation is handled on a subfile in each file state supported by OSINATIVESERVICE.

Table 23–1. Effects of File State on the READ Operation for OSINATIVESERVICE

File State	Action
CLOSED	Implicit OPEN
OFFERED	EOF
AWAITINGOFFER	EOF
AWAITINGHOST	EOF
OPENRESPONSEPLEASE	If the value of CENSUS is greater than 0, returns dialogue request user data; otherwise, returns EOF
OPENED	If the value of CENSUS is greater than 0, returns input data; otherwise depends on WAIT/DONTWAIT (NO WAIT in COBOL74)
SHUTTINGDOWN	If the value of CENSUS is greater than 0, returns input data; otherwise depends on WAIT/DONTWAIT (NO WAIT in COBOL74)

Table 23–1. Effects of File State on the READ Operation for OSINATIVESERVICE

File State	Action
CLOSEREQUESTED	If the value of CENSUS is greater than 0, returns input data; otherwise depends on WAIT/DONTWAIT (NO WAIT in COBOL74)
CLOSERESPONSEPLEASE	If the value of CENSUS is greater than 0, returns input data; otherwise EOF
CLOSECOLLRESPONSEPLEASE	If the value of CENSUS is greater than 0, returns input data; otherwise EOF
DEACTIVATIONPENDING	Returns input data; when READ queue is empty, subfile moves to DEACTIVATED
CLOSEDINPUTPENDING	Returns input data; when READ queue is empty, subfile moves to CLOSED
CLOSEPENDING	EOF
DEACTIVATED	EOF

The result of the WRITE operation depends upon the file state of the subfile you are writing to. Table 23–2 explains the results of the WRITE operation on a subfile in each of the file states supported by OSINATIVESERVICE.

Table 23–2. Effects of File State on the WRITE Operation for OSINATIVESERVICE

File State	Action
CLOSED	Implicit OPEN
OFFERED	EOF
AWAITINGOFFER	EOF
AWAITINGHOST	EOF
OPENRESPONSEPLEASE	EOF
OPENED	If OUTPUTEVENT is in the happened state, then WRITE (OK); otherwise, depends on WAIT/DONTWAIT (NO WAIT in COBOL74)
SHUTTINGDOWN	If OUTPUTEVENT is in the happened state, then WRITE (OK); otherwise, depends on WAIT/DONTWAIT (NO WAIT in COBOL74)
CLOSERESPONSEPLEASE	If OUTPUTEVENT is in the happened state, then WRITE (OK); otherwise, depends on WAIT/DONTWAIT (NO WAIT in COBOL74)
CLOSEREQUESTED	EOF

Table 23–2. Effects of File State on the WRITE Operation for OSINATIVESERVICE

File State	Action
CLOSECOLLRESPONSEPLEASE	EOF
DEACTIVATIONPENDING	EOF
DEACTIVATED	EOF
CLOSEDINPUTPENDING	EOF
CLOSEPENDING	EOF

Exchanging Large Messages Using OSINATIVESERVICE

If you need to exchange messages of unlimited size, you can send or receive data in segments. To use this functionality, set the PORTSEGMENTIO attribute to TRUE and set the BLOCKSTRUCTURE file attribute value to EXTERNAL. If BLOCKSTRUCTURE is set to FIXED, the segments are blank filled, which makes message reassembly difficult.

After you set PORTSEGMENTIO to TRUE, the following statements are true when your program is writing or reading a segmented message:

- The ACTUALMAXRECSIZE value is set to the value of the REQUESTEDMAXRECSIZE file attribute when the subport is opened.
- Message segmentation is allowed only for normal data transfer and is not allowed for associated data sent with OPEN, CLOSE, and RESPOND statements.
- The CENSUS attribute value indicates the number of segments that are in the queue to be read by the program.
- The MAXCENSUS attribute value indicates the maximum number of segments that can be queued before the service provider requests that the correspondent dialogue endpoint stop sending messages.

To write segmented messages, perform the following tasks:

- Define the largest possible segment by using the MAXRECSIZE file attribute. The value you specify cannot be greater than 63K.
- Indicate to the network provider that more segments are going to be written by using the MOREDATA option of the WRITE statement in ALGOL, COBOL74, or Pascal.
- Indicate to the network provider the length of the segment by using the normal length indicator of the WRITE statement.
- When you write the last segment of the message, omit the MOREDATA option from the WRITE statement.

To read segmented messages, perform the following tasks:

- Define the largest possible segment by using the MAXRECSIZE file attribute. The value you specify cannot be greater than 63K.
- Use the READ statement to receive all partial messages. Unisys recommends that the size of the buffer used in the READ statement and the length used in the READ statement should be equal to or greater than the ACTUALMAXRECSIZE value.

  If you do not follow this recommendation and the available message size is greater than the buffer size in the READ statement, a BUFFERLESSTHANASEGMENT (125)I/O error is returned and no data is returned in the buffer.
- Interrogate the [47:20] field of the STATE attribute to determine the length of the data read.
- Interrogate the [26:10] field of the STATE attribute to determine if more segments need to be read. If this field contains MOREDATA (78), then more segments of the messages must be read.

Closing a Dialogue Using OSINATIVESERVICE

In addition to ABORT dialogue termination, discussed in Section 20, "Closing a Dialogue," OSINATIVESERVICE supports orderly dialogue termination. OSINATIVESERVICE also allows for associated data to be sent with the CLOSE statement.

Example Syntax

ALGOL

```
CLOSE (<file name>[SUBFILE <subfile index>],<close
disposition>,
       <control option>,<associated data option>)
```

COBOL74

```
MOVE <subfile index> TO <file subfile control>.
CLOSE <file name> WITH <control option> USING CLOSE-
DISPOSITION OF
     <close disposition>
     <associated data>.
```

For a description of the orderly close disposition, see "Using Orderly Dialogue Termination with OSINATIVESERVICE" in this section. For a description of the associated data parameter, see "Sending Associated Data with a CLOSE Request" in this section.

Using Orderly Dialogue Termination with OSINATIVESERVICE

Orderly dialogue termination can be initiated by your program or by the correspondent endpoint. In both cases, the program can abort the termination procedure by invoking a CLOSE ABORT operation at any time during release. Receiving an ABORT during the termination procedure moves the subfile to the DEACTIVATIONPENDING or DEACTIVATED file state. The subfile moves to DEACTIVATIONPENDING if there is still data in the READ queue.

If you have set the PORTSEGMENTIO file attribute value to TRUE, all segments of the message must be written before your program attempts an orderly dialogue termination.

Understanding a Locally-Initiated Orderly CLOSE

You request orderly dialogue termination through the CLOSE statement. When a CLOSE ORDERLY operation is invoked on a subfile, the subfile moves to the CLOSEREQUESTED file state.

If an accept response is received from the correspondent endpoint for the CLOSE ORDERLY request, the subfile moves to CLOSEDINPUTPENDING as long as the READ queue of the subfile is not empty. When the READ queue is empty, the subfile moves to CLOSED. Note that rejection of orderly release requests is not yet supported.

If, instead of a response, a correspondent-initiated orderly termination request is received, a close collision has occurred. In this case, the subfile moves to the CLOSECOLLRESPONSEPLEASE file state. The action taken next depends on whether or not your subfile initiated the dialogue. Your program gets either a WARNCLOSECOLLINITIATOR (38) or a WARNCLOSECOLLRESPONDER (39) SUBFILEERROR.

If your program initiated the dialogue, the program must first respond with the RESPOND ACCEPTCLOSE statement to the CLOSE request of the other endpoint. Your subfile then returns to the CLOSEREQUESTED file state, and waits for the other endpoint to respond to the CLOSE request. When it receives this response, the subfile goes to either the CLOSED or CLOSEDINPUTPENDING file state.

If your program is the responder in the dialogue, the subfile does not go to CLOSECOLLRESPONSEPLEASE until the initiator of the dialogue has responded to the CLOSE request of your program. When the initiating program has responded and the subfile is in CLOSECOLLRESPONSEPLEASE, your program must then respond to the correspondent CLOSE request. Your subfile goes to either CLOSED or CLOSEDINPUTPENDING when your program performs a RESPOND ACCEPTCLOSE operation.

If the CLOSE control option is WAIT, control is returned to your program when the file state is either CLOSED, CLOSEDINPUTPENDING, or CLOSECOLLRESPONSEPLEASE. Control also returns to your program if it receives an ABORT during an orderly close and the subfile is in either the DEACTIVATED or DEACTIVATIONPENDING file state.

If the CLOSE control option is DONTWAIT (NO WAIT in COBOL74), control is returned to your program as soon as the CLOSE ORDERLY operation is checked for semantic correctness and the file state is CLOSEREQUESTED.

Example 1

ALGOL

```
RSLT := CLOSE(PORTF[SUBFILE 1],
              CLOSEDISPOSITION=ORDERLY);
        CASE RSLT OF
          BEGIN
          OKRSLT:
            DIALOG_COMPLETE(1);
          ELSE:
            DUMP_DIALOG(1);
            CLOSE(PORTF[SUBFILE 1],
                  CLOSEDISPOSITION=ABORT);
          END;
```

COBOL74

```
MOVE 1 TO PORTF-SUB.
CLOSE PORTF USING CLOSE-DISPOSITION OF ORDERLY.
IF PORTF-FS IS NOT EQUAL TO "00"
     PERFORM DUMP-DIALOG
     CLOSE PORTF USING CLOSE-DISPOSITION OF ABORT.
```

This program example initiates orderly release and waits for completion. For the protocol of these applications, only the dialogue initiator is allowed to request an orderly close. If the release could not complete because of a close collision or for any other reason, the subfile is closed with an ABORT.

Example 2

ALGOL

```
RSLT := CLOSE(PORTF[SUBFILE 1],
              CLOSEDISPOSITION=ORDERLY);
CASE RSLT OF
  BEGIN
  OKRSLT:
    DIALOG_COMPLETE(1);
  WARNINGCLOSECOLLINITIATORRSLT:
  WARNINGCLOSECOLLRESPONDERRSLT:
    IF RESPOND(PORT[SUBFILE 1],
               RESPONDTYPE=ACCEPTCLOSE)
       NEQ OKRSLT THEN
       DUMP_DIALOG(1)
    ELSE
       DIALOG_COMPLETE(1);
  ELSE:
    DUMP_DIALOG(1);
    CLOSE(PORTF[SUBFILE 1],
          CLOSEDISPOSITION=ABORT);
  END;
```

COBOL74

```
MOVE 1 TO PORTF-SUB.
CLOSE PORTF USING CLOSE-DISPOSITION OF ORDERLY.
IF PORTF-FS IS EQUAL TO "00"
     PERFORM DIALOG-COMPLETE
ELSE
     IF ATTRIBUTE SUBFILEERROR OF PORTF(1) IS EQUAL TO
                          VALUE (WARNCLOSECOLLINITIATOR)
     OR
     IF ATTRIBUTE SUBFILEERROR OF PORTF(1) IS EQUAL TO
                          VALUE (WARNCLOSECOLLRESPONDER)
          RESPOND PORTF WITH RESPONSE-TYPE OF ACCEPT-
CLOSE
               IF PORTF-FS IS EQUAL TO FS-SUBPORT-NOT-
RESPONDING
                  PERFORM DUMP-DIALOG
               ELSE
                  PERFORM DIALOG-COMPLETE
     ELSE
         PERFORM DUMP-DIALOG
         CLOSE PORTF USING CLOSE-DISPOSITION OF ABORT.
```

The program initiates orderly release in the previous example and waits for completion. If the release could not complete because of a close collision, the program accepts the correspondent's orderly close request and the orderly release continues.

Understanding a Correspondent-Initiated Orderly CLOSE

When your program receives a correspondent-initiated request for an orderly close, the subfile moves to the CLOSERESPONSEPLEASE file state. In this file state, your program can continue to write and to read any queued input that arrived before the CLOSE request. When it is ready to release the connection, your program invokes a RESPOND ACCEPTCLOSE operation, and the subfile moves to either the CLOSEDINPUTPENDING or CLOSED file state.

Sending Associated Data with a CLOSE Request

Your program can send user data with a CLOSE ORDERLY request, with an ABORT, or with a response to an CLOSE ORDERLY request when using OSINATIVESERVICE. It can do so through the ASSOCIATEDDATA parameter of the CLOSE and RESPOND statements.

Example 1

ALGOL

```
CLOSE(PORTF[SUBFILE 1],ASSOCIATEDDATA="SESSIONDATA");
```

COBOL74

```
MOVE 1 TO PORTF-SUB.
CLOSE PORTF USING
     ASSOCIATED-DATA OF "SESSIONDATA".
```

The program is sending SESSIONDATA with its abort message.

Example 2

ALGOL

```
CLOSE(PORTF[SUBFILE 1],
      CLOSEDISPOSITION=ORDERLY,
      DONTWAIT,
      ASSOCIATEDDATALENGTH=14,
      ASSOCIATEDDATA=PTR);
```

COBOL74

```
MOVE 1 TO PORTF-SUB.
CLOSE PORTF WITH NO WAIT USING
     CLOSE-DISPOSITION OF ORDERLY
     ASSOCIATED-DATA-LENGTH OF 14
     ASSOCIATED-DATA OF PORTF-MSG.
```

The program is sending 14 characters of user data, starting from PTR, with its orderly close request message. Control returns to the program as soon as possible.

For more information on associated data, see "Understanding the ASSOCIATEDDATA Parameter of OSINATIVESERVICE" earlier in this section.

Section 24
Using OSISESSIONSERVICE

OSISESSIONSERVICE is a subset of OSINATIVESERVICE, which includes service functions up to the Session layer only. Like OSINATIVESERVICE, OSISESSIONSERVICE currently offers basic full-duplex message transfer, except the APPLICATIONCONTEXT attribute is not supported, and default context and presentation context set do not apply.

OSISESSIONSERVICE supports the kernel and duplex functional units of the OSI Session Layer.

The fields on OSI protocol messages are made accessible to you through port statement parameters and attributes. Not all fields, however, are currently supported. Fields that are not available to you are sent without values when the OSI protocol message is transmitted. If these fields contain values when an OSI protocol message is received from a correspondent, the system discards the values.

The following text summarizes the file attributes and port statements supported by OSISESSIONSERVICE. The information about OSISESSIONSERVICE in this section describes the functions and features specific to this service, because they differ from the general functions and features discussed in the general sections.

File Attributes Supported by OSISESSIONSERVICE

The following file attributes are supported by OSISESSIONSERVICE. Table 15–1 contains the attributes supported by each network service.

ACTUALMAXRECSIZE	APPLICATIONGROUP	ATTERR
ATTVALUE	ATTYPE	AVAILABLE
AVAILABLEONLY	BLOCKEDTIMEOUT	BLOCKSTRUCTURE
CENSUS	CHANGEDSUBFILE	CHANGEEVENT
COMPRESSING	CURRENTRECORDLENGTH	FILEEQUATED
FILENAME	FILESTATE	FRAMESIZE
INPUTEVENT	INTERACTIVEFILE	INTNAME
KIND	LASTSUBFILE	LFILENAME
LTITLE	MAXCENSUS	MAXSUBFILES
MYHOST	MYNAME	OUTPUTEVENT
PATHNAME	PROVIDERGROUP	REINITIALIZE
REQUESTEDMAXRECSIZE	RESULTLIST	SECURITYTYPE
SERVICE	STATE	SUBFILEERROR
YOURHOST	YOURNAME	

In addition, the following file attributes are supported by OSISESSIONSERVICE:

PORTSEGMENTIO	YOURNSAPA	YOURSESSIONSEL
YOURTRANSPORTSEL		

If an attribute is invalid for this service, the only value of the attribute considered valid is the default value. Invalid values are handled as described under "Setting Proper Attribute Values" in Section 15. Be aware that the SECURITYTYPE attribute value must be PUBLIC and can default to PRIVATE if the default setting has not been changed by using the *NW NS SET MIGRATETOBASICSERVICE* system command.

If the attributes YOURHOST and YOURNAME are null strings on an OPEN operation, the operation fails with a BADATTRIBUTESFOROPEN (13) SUBFILEERROR. If the MYNAME attribute is a null string on an OPEN or AWAITOPEN operation, the operation fails with either a BADATTRIBUTESFOROPEN (13) or BADATTRIBUTESFORAWAITOPEN (34) SUBFILEERROR.

The ACTUALMAXRECSIZE attribute value can be 1 through 64512 for the BNA OSI network implementation.

Statements Supported by OSISESSIONSERVICE

Of the set of language statements pertaining to port files, the following are supported by OSISESSIONSERVICE:

- Attribute interrogation
- Attribute modification
- OPEN
  - Control option: WAIT/DONTWAIT (NO WAIT in COBOL74)/AVAILABLE
  - CONNECTTIMELIMIT
  - Associated data
  - Associated data length specification
- AWAITOPEN
  - Control option: WAIT/DONTWAIT (NO WAIT in COBOL74)/AVAILABLE
  - CONNECTTIMELIMIT
  - PARTICIPATE
- READ
  - Message
  - READ length specification
  - WAIT/DONTWAIT (NO WAIT in COBOL74)
- WRITE
  - Message
  - WRITE length specification
  - WAIT/DONTWAIT (NO WAIT in COBOL74)
  - MOREDATA
- RESPOND
  - Respond type: ACCEPTOPEN/REJECTOPEN/ACCEPTCLOSE
  - Associated data
  - Associated data length specification
- CLOSE
  - Close disposition: ABORT/ORDERLY
  - Control option: WAIT/DONTWAIT (NO WAIT in COBOL74)
  - Associated data
  - Associated data length specification

If you use a port statement not included in the preceding list, the system does not invoke the statement and an UNSUPPORTEDPRIMITIVERSLT (168) result and an UNSUPPORTEDPRIMITIVE (41) SUBFILEERROR are returned. If you use a statement parameter or a statement parameter value not included in the previous list, the system does not invoke the statement and an UNSUPPORTEDPARAMETERRSLT (128) result and an UNSUPPORTEDPARAMETER (18) SUBFILEERROR are returned.

Understanding the ASSOCIATEDDATA Parameter of OSISESSIONSERVICE

Your program can send user data with control messages exchanged during dialogue establishment and dialogue termination. You program can do this through the ASSOCIATEDDATA parameter of the OPEN, CLOSE, and RESPOND statements.

Your program receives associated data through the READ queue. The result of your READ statement tells you if the data returned by the READ operation is associated data.

The maximum length of ASSOCIATEDDATA that can be sent primarily depends upon the implementation restrictions of both network providers, as well as the number of bytes needed to encode certain dialogue parameters. The OSISESSIONSERVICE provider can guarantee support for 7K bytes of associated data. A correspondent OSI implementation might support a protocol version that can guarantee only up to 200 bytes of associated data and no data at all on an abort. The lesser restriction applies.

Associated data lengths greater than these limits can be sent and received by your program. However, they are not guaranteed by the MCP environment OSI provider. It is the responsibility of an application using OSI to specify its requirements for maximum associated data length, and hence its requirements on OSI network implementations, to other applications that want to communicate with it.

The preceding limits can be guaranteed only if the application context object ID has

- At most, 6 components
- Each component in the range 0 through 127 characters

If the associated data is too long, the OPEN, CLOSE ORDERLY, or RESPOND operation fails with an ASSOCIATEDDATATOOLONGRSLT (156) result and an ASSOCIATEDDATATOOLONG (32) SUBFILEERROR. If the associated data is too long on a CLOSE ABORT operation, the ABORT operation continues with the warning WARNABORTDATAIGNOREDRSLT (17) and a WARNABORTDATAIGNORED (48) SUBFILEERROR, and the ABORT is sent without associated data.

File States Supported by OSISESSIONSERVICE

The following is a list of all the possible file states a port file can have using OSISESSIONSERVICE:

CLOSED	CLOSEREQUESTED
OFFERED	CLOSERESPONSEPLEASE
AWAITINGHOST	CLOSEDINPUTPENDING
AWAITINGOFFER	CLOSECOLLRESPONSEPLEASE
OPENRESPONSEPLEASE	DEACTIVATIONPENDING
OPENED	DEACTIVATED
SHUTTINGDOWN	
CLOSEPENDING	

Figures 24–1 through 24–4 illustrate the file state transitions during the dialogue establishment phase, the data transfer phase, and the dialogue termination phase for OSISESSIONSERVICE. Note that a subfile can have multiple file state transitions between CHANGEEVENT and FILESTATE interrogations.

In the figures that follow, the following conventions apply:

- All user-initiated primitives, such as CLOSE ABORT, are in capital letters.
- Other events, such as dialogue not established, are enclosed in square brackets.

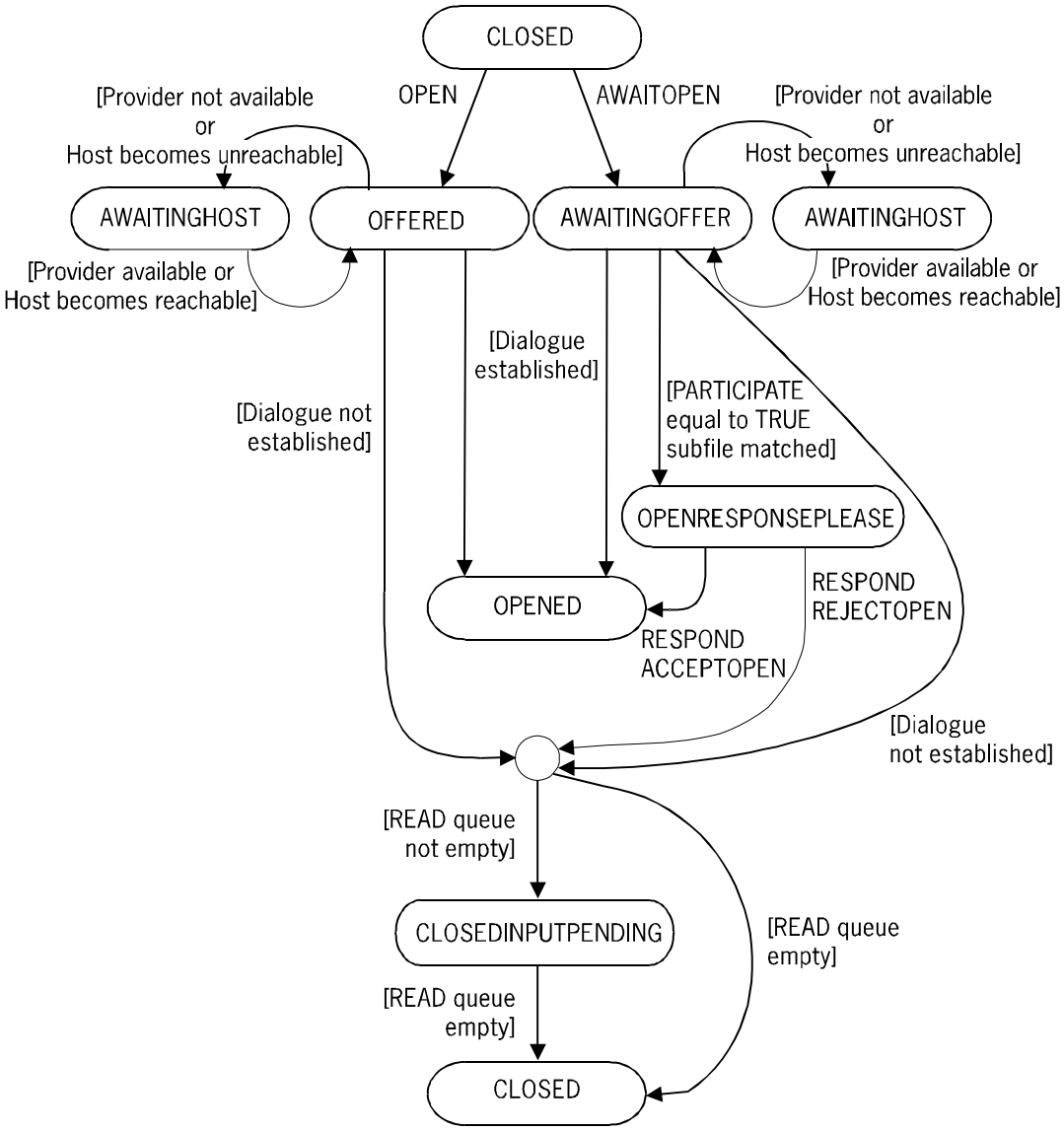


Figure 24–1. OSISESSIONSERVICE Dialogue Establishment File State Transitions

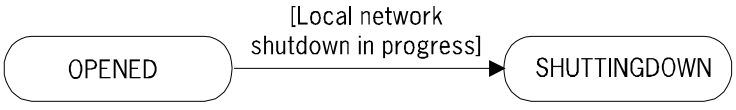


Figure 24–2. OSISESSIONSERVICE Probable File State Transitions during Data Transfer

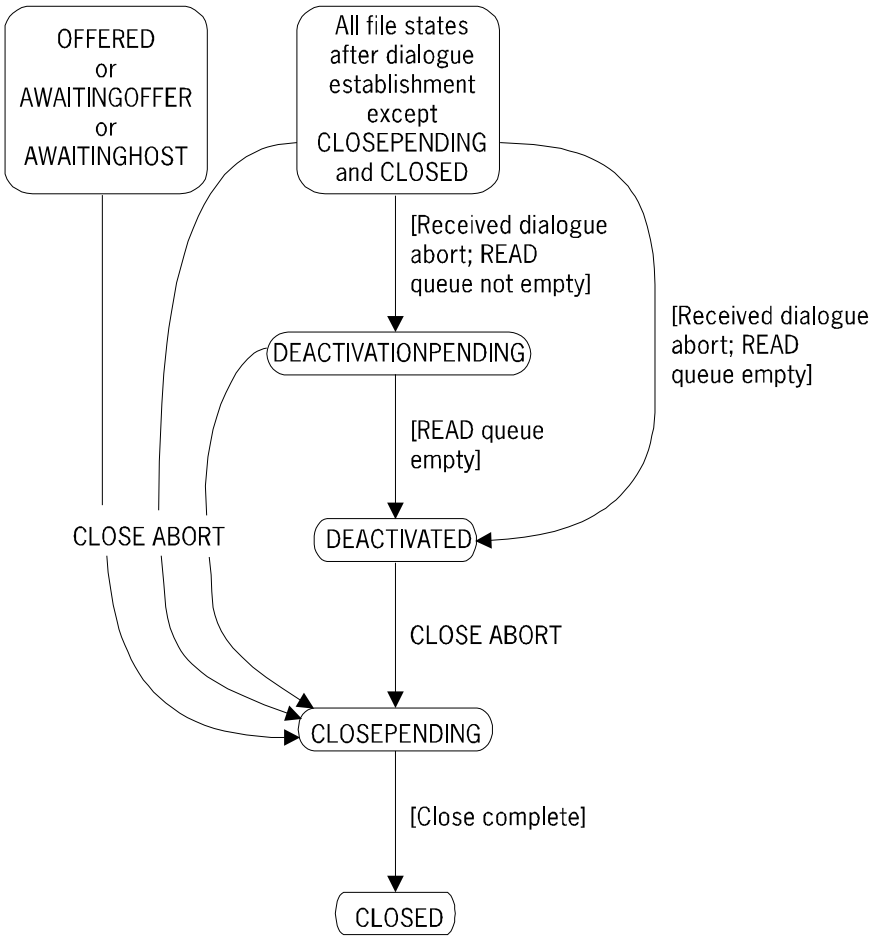


Figure 24–3. OSISESSIONSERVICE Dialogue Termination File State Transitions

If the remote host becomes unreachable, the file state is marked as BLOCKED. The file state becomes OPENED if the remote host becomes reachable within a specific time. If the remote host does not become reachable within that time period, the system initiates deactivation of the dialogue.

Use the BLOCKEDTIMEOUT attribute to control the length of time that the subport is to remain blocked before the system initiates deactivation of the dialogue.

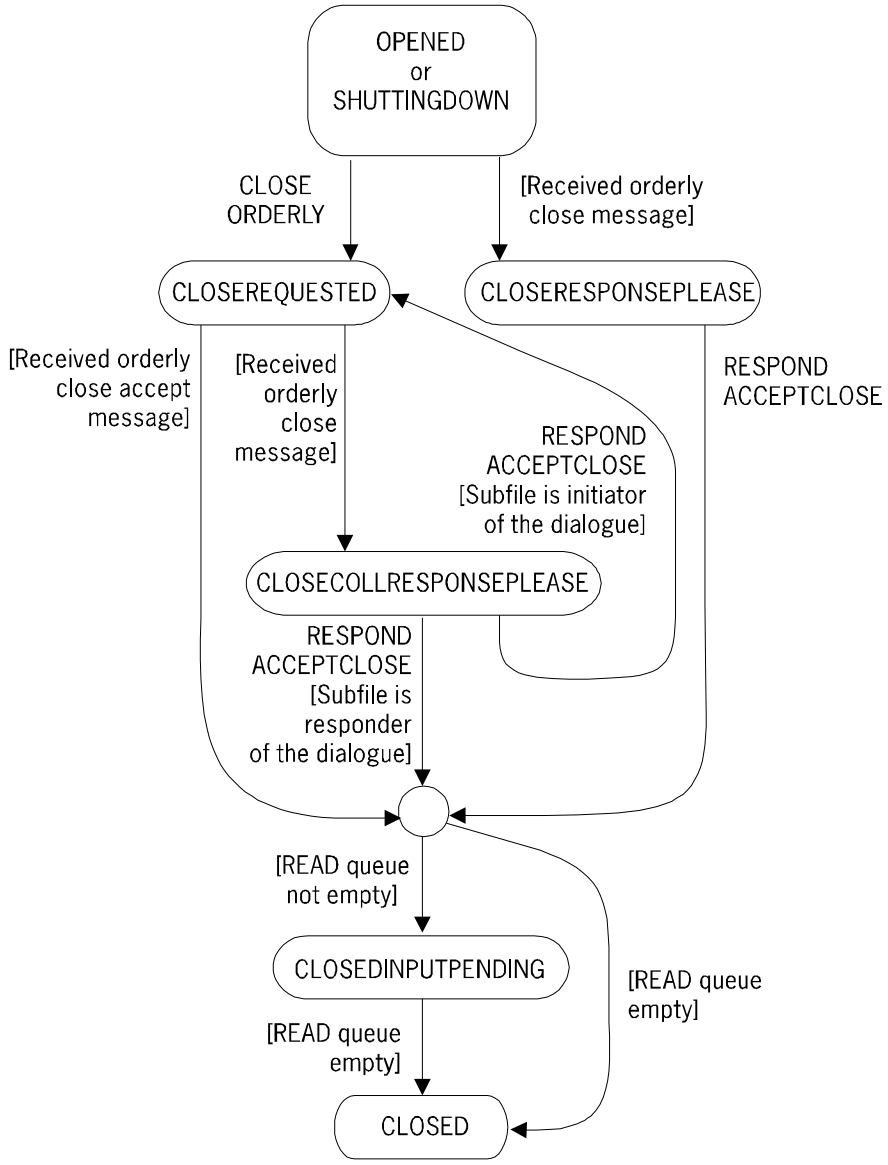


Figure 24–4. OSISESSIONSERVICE Orderly Termination File State Transitions

Preparing for Dialogue Establishment Using OSISESSIONSERVICE

The following OSI dialogue-establishment fields are used for matching:

OSI Dialogue Request/Response Fields	Attribute for OSI 1.2 or 3.0 Release	Attribute for OSI 3.1 and Later Releases
OSI address (your subfile)	MYNAME MYHOST FILENAME APPLICATIONGROUP	MYNAME MYHOST FILENAME APPLICATIONGROUP
OSI address (correspondent)	YOURNAME YOURHOST FILENAME APPLICATIONGROUP	YOURNAME YOURHOST FILENAME APPLICATIONGROUP YOURNSAPA YOURSESSIONSEL YOURTRANSPORTSEL

OSI endpoint addresses, both local and remote, are statically declared to the system through the *NW ADD ENDPOINTNAME* operator command. Through this command, an OSI endpoint address is declared and mapped into a NAME, HOST, FILENAME, and APPLICATIONGROUP attribute set. There is a current restriction that only direct Session Layer users can access OSISESSIONSERVICE. That is, the OSI addresses must be Session Layer user addresses. If a full OSI endpoint address is used, the OPEN or AWAITOPEN operation fails with an ENDPTINCOMPATIBLESERVICERSLT (166) result and an ENDPOINTINCOMPWITHSERVICE (40) SUBFILEERROR.

If your application program is using the OSI 3.1 or later release, you can use the YOURNSAPA, YOURSESSIONSEL, and YOURTRANSPORTSEL file attributes to directly identify the network end-point addresses of the correspondent application that might not be declared in the static network configuration.

If your application program is using the OSI 1.2 or 3.0 release, the network environment matches MYNAME, MYHOST, YOURNAME, and YOURHOST as described earlier in Section 17, "Preparing Your Subfile for Dialogue Establishment." The FILENAME and APPLICATIONGROUP values of your subfile must match the FILENAME and APPLICATIONGROUP values of the correspondent endpoint.

If your application program is using the OSI 3.1 or later release and you specify YOURSESSIONSEL, YOURTRANSPORTSEL, and YOURNSAPA values, these attributes are matched even if you specify values for the YOURNAME, YOURHOST, FILENAME, and APPLICATIONGROUP values. If you do not specify YOURSESSIONSEL, YOURTRANSPORTSEL, and YOURNSAPA values, YOURNAME, YOURHOST, FILENAME, and APPLICATIONGROUP values are matched.

Note that currently the BNA OSI network implementation associates only one OSI address for an endpoint for each NAME, HOST, FILENAME, and APPLICATIONGROUP attribute set. Endpoints that are reachable through more than one OSI address (because of multiple network addresses, for example) must have an attribute set for each address.

Establishing a Dialogue Using OSISESSIONSERVICE

An OSI dialogue request is transmitted when you invoke an OPEN operation on a subfile. When you invoke an AWAITOPEN operation, your subfile is made available to be matched to an incoming dialogue request. If both endpoints send dialogue requests, the colliding dialogue requests are considered as two separate requests for dialogue. That is, incoming dialogue requests are not matched to subfiles on which you have invoked an OPEN operation.

Using the OPEN Statement with OSISESSIONSERVICE

When you invoke an OPEN statement, a dialogue request is sent to the OSI endpoint address named by the YOURNAME, YOURHOST, FILENAME, and APPLICATIONGROUP attributes. You can send additional dialogue parameters with the request by using attributes or parameters of the OPEN statement. Some dialogue parameters are negotiated by the dialogue endpoints. The values of other dialogue parameters, like associated data, are merely exchanged. See "Understanding Negotiation during Dialogue Establishment" in Section 23, for more information on this topic.

Example Syntax

ALGOL

```
OPEN (<file name>[SUBFILE <subfile index>],<control
option>,
      <connect time limit option>,<associated data
option>)
```

COBOL74

```
MOVE <subfile index> TO <file subfile control>.
OPEN <control option> <file name> USING
     <connect time limit>
     <associated data option>
```

In addition to the file name, subfile index, control option, and connect time limit option, OSISESSIONSERVICE also supports the associated data option. This option consists of two elements:

- ASSOCIATEDDATALENGTH. This element specifies the length of the user data to be sent.
- ASSOCIATEDDATA. This specifies the user data to be sent.

Understanding the AVAILABLEONLY File Attribute Using OSISESSIONSERVICE

As discussed in "Understanding the AVAILABLEONLY File Attribute for OPEN," in Section 18, the AVAILABLEONLY attribute specifies whether the OPEN operation fails or suspends when a dialogue cannot currently be established with the correspondent endpoint.

If your port file is communicating with another port file on the local system, the following actions will occur depending on the value of the AVAILABLEONLY attribute:

- If AVAILABLEONLY is set to FALSE, the OPEN operation is suspended until a dialogue is established, or until the CONNECTTIMELIMIT value is exceeded.
- If AVAILABLEONLY is set to TRUE and no match is found, the OPEN operation fails, your program resumes, and a NOFILEFOUND (4) SUBFILEERROR is returned, and the subfile is not available for subsequent matching.

When your port file is communicating with another port file on a remote host, the AVAILABLEONLY attribute has no effect.

Understanding the OPEN Control Option Parameter for OSISESSIONSERVICE

OSISESSIONSERVICE allows the following values for the control option parameter of the OPEN statement:

- WAIT
- DONTWAIT (NO WAIT in COBOL74)
- AVAILABLE

These control options function as described earlier under "Understanding the OPEN Control Option Parameter" in Section 18.

Understanding the CONNECTTIMELIMIT Parameter for OSISESSIONSERVICE

This parameter functions the same as described earlier under "Understanding the OPEN CONNECTTIMELIMIT Parameter" in Section 18.

Understanding the OPEN ASSOCIATEDDATA Parameter for OSISESSIONSERVICE

You can send user data with a dialogue request when using OSISESSIONSERVICE. You can do so through the ASSOCIATEDDATA parameter of the OPEN statement.

No user data is sent if the associated data parameter is absent or if the associated data length is 0 (zero).

Example 1

ALGOL
```
OPEN(PORTF[SUBFILE 1],ASSOCIATEDDATA="MYDATA");
```

COBOL74
```
MOVE 1 TO PORTF-SUB.
OPEN I-O PORTF USING ASSOCIATED-DATA OF "MYDATA".
```

This example opens subfile 1 of PORTF. When the dialogue request is sent, the information "MYDATA" is sent to the correspondent process as user data.

Example 2

ALGOL
```
OPEN(PORTF[SUBFILE 1],
     ASSOCIATEDDATALENGTH=14,
     ASSOCIATEDDATA=PTR);
```

COBOL74
```
MOVE 1 TO PORTF-SUB.
OPEN I-O PORTF USING
     ASSOCIATED-DATA-LENGTH OF 14
     ASSOCIATED-DATA OF PORTF-MSG.
```

This example opens subfile 1 of PORTF. When the dialogue request is sent, the data pointed at by PTR for 14 characters is sent to the correspondent process as user data.

Using the AWAITOPEN Statement with OSISESSIONSERVICE

Invoke an AWAITOPEN statement to make a subfile available to accept dialogue requests from other programs. When an AWAITOPEN is invoked, the local system is searched for an outstanding dialogue request that matches the subfile. If no matching dialogue request has already been received, the AWAITOPEN fails or is saved for future matching, depending upon the value of the AVAILABLEONLY attribute.

When the system receives a dialogue request, it matches the request to a subfile that invoked an AWAITOPEN statement. If the system cannot match the request, it keeps the request for future matching.

The following fields from the dialogue request received from the correspondent endpoint are used by the OSISESSIONSERVICE provider when it matches the request to a subfile:

- The calling endpoint address
- The called endpoint address

The addresses are looked up on the local system to retrieve matching values for the NAME, HOST, FILENAME, and APPLICATIONGROUP attributes. These values are used to match to the attributes of the subfile. The subfile is matched as described in "Preparing for Dialogue Establishment Using OSISESSIONSERVICE" in this section.

Example Syntax

ALGOL

```
AWAITOPEN (<file name>[SUBFILE <subfile index>],
           <control option>, <participate option>,
           <connect time limit option>)
```

COBOL74

```
MOVE <subfile index> TO <file subfile control>.
AWAIT-OPEN <control option> <file name> USING
     <participate option>
     <connect time limit option>
```

In addition to the file name, subfile index, control option, and connect time limit parameters of the general port AWAITOPEN statement, OSISESSIONSERVICE also supports the PARTICIPATE option. This parameter allows you to conditionally or unconditionally establish a dialogue when the subfile is matched to an incoming dialogue request. See "Understanding the AWAITOPENPARTICIPATE Parameter for OSISESSIONSERVICE" later in this section for more information.

Understanding the AWAITOPEN Control Option Parameter for OSISESSIONSERVICE

OSISESSIONSERVICE allows the following values for the AWAITOPEN control:

- WAIT
- DONTWAIT (NO WAIT in COBOL74)
- AVAILABLE

These options function as described earlier under "Understanding the AWAITOPEN Control Option Parameter" in Section 18.

Understanding the AWAITOPEN PARTICIPATE Parameter for OSISESSIONSERVICE

OSISESSIONSERVICE provides you with the ability to conditionally establish a dialogue on an AWAITOPEN operation. The PARTICIPATE option of the AWAITOPEN statement specifies if the program uses conditional or unconditional dialogue establishment when a dialogue request is received. The PARTICIPATE option can be either TRUE or FALSE.

If PARTICIPATE is FALSE when a request for dialogue establishment is received, the request is accepted and the subfile is moved to an OPENED file state. The values of negotiable attributes are unconditionally accepted and no user data is sent with the response.

If PARTICIPATE is TRUE when a request for dialogue establishment is received, the subfile is moved to the OPENRESPONSEPLEASE file state. In this state, the program can interrogate or modify certain attributes for negotiation or for transmission on the response. The program accepts or rejects the call through the RESPOND ACCEPTOPEN/REJECTOPEN statement.

If there is user data in the READ queue that came with the dialogue request (OPENDATA), the program can read the data before it responds. If you want to send user data with the response, use the associated data parameter of the RESPOND statement. If the program accepts the request, the subfile moves to the OPENED file state. If the program rejects the request, the subfile moves to the CLOSED or CLOSEDINPUTPENDING file state.

PARTICIPATE defaults to FALSE.

Example

ALGOL
```
AWAITOPEN(PORTF[SUBFILE 1],PARTICIPATE=TRUE);
```

COBOL74
```
MOVE 1 TO PORTF-SUBF.
AWAIT-OPEN PORTF USING PARTICIPATE.
```

When subfile 1 of PORTF is matched to an incoming dialogue request, the subfile transits to the OPENRESPONSEPLEASE file state. In this file state, the program can interrogate attributes of the matching dialogue request, set negotiable attributes, and READ any associated data that might have arrived with the request.

Using the RESPOND Statement with OSISESSIONSERVICE

OSISESSIONSERVICE allows a program to respond positively or negatively to a request indication received from a correspondent endpoint. The nature of the request is contained in the FILESTATE attribute. The program uses the RESPOND statement to respond to dialogue establishment requests (OPENRESPONSEPLEASE file state) and dialogue termination requests (CLOSERESPONSEPLEASE and CLOSECOLLRESPONSEPLEASE file states) from the correspondent endpoint. The RESPOND statement returns a result value.

Example Syntax

ALGOL
```
RESPOND (<file identifier>[SUBFILE<subfile index>],
        RESPONDTYPE=<response type>,<associated data
option>)
```

COBOL74
```
MOVE <subfile index> TO <file subfile control>
RESPOND <file name> WITH RESPONSE-TYPE OF <response type>
     USING <associated data option>.
```

The syntax elements are as follows:

Element	Purpose
File identifier	Specifies the name of the port file on which the RESPOND operation is to be performed.
Subfile index	Indicates the subfile that the response is for. A subfile index of 0 (zero) is currently not allowed.
Response type	Specifies the type of response by using one the following values: ACCEPTOPEN, REJECTOPEN, or ACCEPTCLOSE. The response type is described under "Understanding the Response Type Parameter" later in this section.
Associated data option	Identifies data to be sent with the RESPOND statement. This option consists of two elements: ASSOCIATEDDATALENGTH. This element specifies the length of user data to be sent. ASSOCIATEDDATA. This element specifies the user data to be sent. See "Understanding the ASSOCIATEDDATA Parameter of OSISESSIONSERVICE" earlier in this section for more information.

Example 1

ALGOL
```
RESPOND(PORTF[SUBFILE 1], RESPONDTYPE=ACCEPTOPEN);
```

COBOL74
```
MOVE 1 TO PORTF-SUB.
RESPOND PORTF WITH RESPONSE-TYPE OF ACCEPT-OPEN.
```

In this example, the program is responding to an offer for subfile 1 on port file PORTF by accepting the offer.

Example 2

ALGOL
```
RESPOND(PORTF[SUBFILE 1],
        RESPONDTYPE=ACCEPTOPEN,
        ASSOCIATEDDATA="INFO");
```

COBOL74
```
MOVE 1 TO PORTF-SUB.
RESPOND PORTF WITH RESPONSE-TYPE OF ACCEPT-OPEN
     USING ASSOCIATED-DATA OF "INFO".
```

The program is responding to an offer for subfile 1 of port file PORTF by accepting the offer. The information INFO is sent as user data with the positive response to the dialogue request.

Understanding the Response Type Parameter

You can assign the response type parameter the values ACCEPTOPEN, REJECTOPEN, and ACCEPTCLOSE.

Your program can respond to a dialogue establishment request if the program invoked an AWAITOPEN operation with PARTICIPATE set to TRUE. When a dialogue establishment request is received, the file state moves to OPENRESPONSEPLEASE. Your program can agree to the dialogue by assigning the value ACCEPTOPEN to the response type parameter of the RESPOND statement. Your program can reject the dialogue request by assigning the value REJECTOPEN to the RESPOND response type parameter.

Your program can respond to a dialogue orderly close request when the subfile file state is CLOSERESPONSEPLEASE or CLOSECOLLRESPONSEPLEASE. Your program can agree to the close request by assigning the value ACCEPTCLOSE to the response type. Note that a response type of REJECTCLOSE is not yet supported.

Example 1

ALGOL
```
RESPOND(PORTF[SUBFILE 1],RESPONDTYPE=ACCEPTOPEN);
```

COBOL74
```
MOVE 1 TO PORTF-SUB.
RESPOND PORTF WITH RESPONSE-TYPE OF ACCEPT-OPEN.
```

In this example, subfile 1 of PORTF is in the file state OPENRESPONSEPLEASE. The program is responding to the OPEN request by accepting the offer.

Example 2

ALGOL
```
RESPOND(PORTF[SUBFILE 1],RESPONDTYPE=REJECTOPEN);
```

COBOL74
```
MOVE 1 TO PORTF-SUB.
RESPOND PORTF WITH RESPONSE-TYPE OF REJECT-OPEN.
```

In this example, subfile 1 of PORTF is in the file state OPENRESPONSEPLEASE. The program is responding to the OPEN request by rejecting the offer.

Example 3

ALGOL
```
RESPOND(PORTF[SUBFILE 1],RESPONDTYPE=ACCEPTCLOSE);
```

COBOL74
```
MOVE 1 TO PORTF-SUB.
RESPOND PORTF WITH RESPONSE-TYPE OF ACCEPT-CLOSE.
```

In this example, subfile 1 of PORTF is in the file state CLOSERESPONSEPLEASE or CLOSECOLLRESPONSEPLEASE. The program is positively responding to the request for orderly close.

Exchanging Data Using OSISESSIONSERVICE

Any associated data received from the correspondent is placed in the READ queue. When you invoke a READ statement, the result of the READ operation tells you if the data returned by the READ operation is associated data (for example, OPENDATA, OPENRESPONSEDATA).

The way a READ operation is handled by the port service depends upon the file state of the subfile. Table 24–1 describes the way a READ operation is handled on a subfile in each file state supported by OSISESSIONSERVICE.

Table 24–1. Effects of File State on the READ Operation for OSISESSIONSERVICE

File State	Action
CLOSED	Implicit OPEN
OFFERED	EOF
AWAITINGOFFER	EOF
AWAITINGHOST	EOF
OPENRESPONSEPLEASE	If the value of CENSUS is greater than 0, returns dialogue request user data; otherwise, returns EOF
OPENED	If the value of CENSUS is greater than 0, returns input data; otherwise depends on WAIT/DONTWAIT (NO WAIT in COBOL74)
SHUTTINGDOWN	If the value of CENSUS is greater than 0, returns input data; otherwise depends on WAIT/DONTWAIT (NO WAIT in COBOL74)
CLOSEREQUESTED	If the value of CENSUS is greater than 0, returns input data; otherwise depends on WAIT/DONTWAIT (NO WAIT in COBOL74)
CLOSERESPONSEPLEASE	If the value of CENSUS is greater than 0, returns input data; otherwise EOF
CLOSECOLLRESPONSEPLEASE	If the value of CENSUS is greater than 0, returns input data; otherwise EOF
DEACTIVATIONPENDING	Returns input data; when READ queue is empty, subfile moves to DEACTIVATED
CLOSEDINPUTPENDING	Returns input data; when READ queue is empty, subfile moves to CLOSED
CLOSEPENDING	EOF
DEACTIVATED	EOF

The result of the WRITE operation depends upon the file state of the subfile you are writing to. Table 24–2 explains the results of the WRITE operation on a subfile in each of the file states supported by OSISESSIONSERVICE.

Table 24–2. Effects of File State on the WRITE Operation for OSISESSIONSERVICE

File State	Action
CLOSED	Implicit OPEN
OFFERED	EOF
AWAITINGOFFER	EOF
AWAITINGHOST	EOF
OPENRESPONSEPLEASE	EOF
OPENED	If OUTPUTEVENT is in the happened state, then WRITE (OK); otherwise, depends on WAIT/DONTWAIT (NO WAIT in COBOL74)
SHUTTINGDOWN	If OUTPUTEVENT is in the happened state, then WRITE (OK); otherwise, depends on WAIT/DONTWAIT (NO WAIT in COBOL74)
CLOSERESPONSEPLEASE	If OUTPUTEVENT is in the happened state, then WRITE (OK); otherwise, depends on WAIT/DONTWAIT (NO WAIT in COBOL74)
CLOSEREQUESTED	EOF
CLOSECOLLRESPONSEPLEASE	EOF
DEACTIVATIONPENDING	EOF
DEACTIVATED	EOF
CLOSEDINPUTPENDING	EOF
CLOSEPENDING	EOF

Exchanging Large Messages Using OSISESSIONSERVICE

If you need to exchange messages of unlimited size, you can send or receive data in segments. To use this functionality, set the PORTSEGMENTIO attribute to TRUE and set the BLOCKSTRUCTURE file attribute value to EXTERNAL. If BLOCKSTRUCTURE is set to FIXED, the segments are blank filled, which makes message reassembly difficult.

After you set PORTSEGMENTIO to TRUE, the following statements are true when your program is writing or reading a segmented message:

- The ACTUALMAXRECSIZE value is set to the value of the REQUESTEDMAXRECSIZE file attribute when the subport is opened.
- Message segmentation is allowed only for normal data transfer and is not allowed for associated data sent with OPEN, CLOSE, and RESPOND statements.
- The CENSUS attribute value indicates the number of segments that are in the queue to be read by the program.
- The MAXCENSUS attribute value indicates the maximum number of segments that can be queued before the service provider requests that the correspondent dialogue endpoint stop sending messages.

To write segmented messages, perform the following tasks:

- Define the largest possible segment by using the MAXRECSIZE file attribute. The value you specify cannot be greater than 63K.
- Indicate to the network provider that more segments are going to be written by using the MOREDATA option of the WRITE statement in ALGOL, COBOL74, or Pascal.
- Indicate to the network provider the length of the segment by using the normal length indicator of the WRITE statement.
- When you write the last segment of the message, omit the MOREDATA option from the WRITE statement.

To read segmented messages, perform the following tasks:

- Define the largest possible segment by using the MAXRECSIZE file attribute. The value you specify cannot be greater than 63K.
- Use the READ statement to receive all partial messages. Unisys recommends that the size of the buffer used in the READ statement and the length used in the READ statement should be equal to or greater than the ACTUALMAXRECSIZE value.

  If you do not follow this recommendation and the available message size is greater than the buffer size in the READ statement, a BUFFERLESSTHANASEGMENT (125) I/O error is returned and no data is returned in the buffer.

- Interrogate the [47:20] field of the STATE attribute to determine the length of the data read.
- Interrogate the [26:10] field of the STATE attribute to determine if more segments need to be read. If this field contains MOREDATA (78), then more segments of the messages must be read.

Closing a Dialogue Using OSISESSIONSERVICE

In addition to ABORT dialogue termination, discussed in Section 20, "Closing a Dialogue," OSISESSIONSERVICE supports orderly dialogue termination. OSISESSIONSERVICE also allows for associated data to be sent with the CLOSE statement.

Example Syntax

ALGOL

```
CLOSE (<file name>[SUBFILE <subfile index>],<close
disposition>,
       <control option>,<associated data option>)
```

COBOL74

```
MOVE <subfile index> TO <file subfile control>.
CLOSE <file name> WITH <control option> USING CLOSE-
DISPOSITION OF
     <close disposition>
     <associated data>.
```

For a description of the orderly close disposition, see "Using Orderly Dialogue Termination with OSISESSIONSERVICE" in this section. For a description of the associated data parameter, see "Sending Associated Data with a CLOSE Request" later in this section.

Using Orderly Dialogue Termination with OSISESSIONSERVICE

Orderly dialogue termination can be initiated by your program or by the correspondent endpoint. In both cases, the program can abort the termination procedure by invoking a CLOSE ABORT operation at any time during release. Receiving an ABORT during the termination procedure moves the subfile to the DEACTIVATIONPENDING or DEACTIVATED file state. The subfile moves to DEACTIVATIONPENDING if there is still data in the READ queue.

If you have set the PORTSEGMENTIO file attribute value to TRUE, all segments of the message must be written before your program attempts an orderly dialogue termination.

Understanding a Locally-Initiated Orderly CLOSE

You request orderly dialogue termination through the CLOSE statement. When a CLOSE ORDERLY operation is invoked on a subfile, the subfile moves to the CLOSEREQUESTED file state.

If an accept response is received from the correspondent endpoint for the CLOSE ORDERLY request, the subfile moves to CLOSEDINPUTPENDING as long as the READ queue of the subfile is not empty. When the READ queue is empty, the subfile moves to CLOSED. Note that rejection of ORDERLY release requests is not yet supported.

If, instead of a response, a correspondent-initiated orderly termination request is received, a close collision has occurred. In this case, the subfile moves to the CLOSECOLLRESPONSEPLEASE file state. The action taken next depends on whether or not your subfile initiated the dialogue. Your program gets either a WARNCLOSECOLLINITIATOR (38) or a WARNCLOSECOLLRESPONDER (39) SUBFILEERROR.

If your program initiated the dialogue, the program must first respond with the RESPOND ACCEPTCLOSE statement to the CLOSE request of the other endpoint. Your subfile then returns to the CLOSEREQUESTED file state, and waits for the other endpoint to respond to the CLOSE request. When it receives this response, the subfile goes to either the CLOSED or CLOSEDINPUTPENDING file state.

If your program is the responder in the dialogue, the subfile does not go to CLOSECOLLRESPONSEPLEASE until the initiator of the dialogue has responded to the CLOSE request of your program. When the initiating program has responded and the subfile is in CLOSECOLLRESPONSEPLEASE, your program must then respond to the correspondent CLOSE request. Your subfile goes to either CLOSED or CLOSEDINPUTPENDING when your program performs a RESPOND ACCEPTCLOSE operation.

If the CLOSE control option is WAIT, control is returned to your program when the file state is either CLOSED, CLOSEDINPUTPENDING, or CLOSECOLLRESPONSEPLEASE. Control also returns to your program if it receives an ABORT during an orderly close and the subfile is in either the DEACTIVATED or DEACTIVATIONPENDING file state.

If the CLOSE control option is DONTWAIT (NO WAIT in COBOL74), control is returned to your program as soon as the CLOSE ORDERLY operation is checked for semantic correctness and the file state is CLOSEREQUESTED.

Example 1

ALGOL

```
RSLT := CLOSE(PORTF[SUBFILE 1],
              CLOSEDISPOSITION=ORDERLY);
        CASE RSLT OF
          BEGIN
          OKRSLT:
            DIALOG_COMPLETE(1);
          ELSE:
            DUMP_DIALOG(1);
            CLOSE(PORTF[SUBFILE 1],
                  CLOSEDISPOSITION=ABORT);
          END;
```

COBOL74

```
MOVE 1 TO PORTF-SUB.
CLOSE PORTF USING CLOSE-DISPOSITION OF ORDERLY.
IF PORTF-FS IS NOT EQUAL TO "00"
     PERFORM DUMP-DIALOG
     CLOSE PORTF USING CLOSE-DISPOSITION OF ABORT.
```

This program example initiates orderly release and waits for completion. For the protocol of these applications, only the dialogue initiator is allowed to request orderly CLOSE. If the release could not complete because of a close collision or for any other reason, the subfile is closed with an ABORT.

Example 2

ALGOL

```
RSLT := CLOSE(PORTF[SUBFILE 1],
              CLOSEDISPOSITION=ORDERLY);
CASE RSLT OF
  BEGIN
  OKRSLT:
    DIALOG_COMPLETE(1);
  WARNINGCLOSECOLLINITIATORRSLT:
  WARNINGCLOSECOLLRESPONDERRSLT:
    IF RESPOND(PORT[SUBFILE 1],
               RESPONDTYPE=ACCEPTCLOSE)
       NEQ OKRSLT THEN
       DUMP_DIALOG(1)
    ELSE
       DIALOG_COMPLETE(1);
  ELSE:
    DUMP_DIALOG(1);
    CLOSE(PORTF[SUBFILE 1],
          CLOSEDISPOSITION=ABORT);
  END;
```

COBOL74

```
MOVE 1 TO PORTF-SUB.
CLOSE PORTF USING CLOSE-DISPOSITION OF ORDERLY.
IF PORTF-FS IS EQUAL TO "00"
     PERFORM DIALOG-COMPLETE
ELSE
     IF ATTRIBUTE SUBFILEERROR OF PORTF(1) IS EQUAL TO
                          VALUE (WARNCLOSECOLLINITIATOR)
     OR
     IF ATTRIBUTE SUBFILEERROR OF PORTF(1) IS EQUAL TO
                          VALUE (WARNCLOSECOLLRESPONDER)
          RESPOND PORTF WITH RESPONSE-TYPE OF ACCEPT-
CLOSE
          IF PORTF-FS IS EQUAL TO FS-SUBPORT-NOT-
RESPONDING
               PERFORM DUMP-DIALOG
          ELSE
               PERFORM DIALOG-COMPLETE
     ELSE
          PERFORM DUMP-DIALOG
          CLOSE PORTF USING CLOSE-DISPOSITION OF ABORT.
```

The program initiates orderly release in the example above and waits for completion. If the release could not complete because of a close collision, the program accepts the correspondent's ORDERLY CLOSE request and the orderly release continues.

Understanding a Correspondent-Initiated Orderly CLOSE

When your program receives a correspondent-initiated request for an orderly close, the subfile moves to the CLOSERESPONSEPLEASE file state. In this file state, your program can continue to write and to read any queued input that arrived before the CLOSE request. When it is ready to release the connection, your program invokes a RESPOND ACCEPTCLOSE operation, and the subfile moves to either the CLOSEDINPUTPENDING or CLOSED file state.

Sending Associated Data with a CLOSE Request

Your program can send user data with a CLOSE ORDERLY request, with an ABORT, or with a response to an CLOSE ORDERLY request when using OSISESSIONSERVICE. It can do so through the ASSOCIATEDDATA parameter of the CLOSE and RESPOND statements.

Example 1

ALGOL

```
CLOSE(PORTF[SUBFILE 1],ASSOCIATEDDATA="SESSIONDATA");
```

COBOL74

```
MOVE 1 TO PORTF-SUB.
CLOSE PORTF USING
     ASSOCIATED-DATA OF "SESSIONDATA".
```

The program is sending "SESSIONDATA" with its abort message.

Example 2

ALGOL

```
CLOSE(PORTF[SUBFILE 1],
      CLOSEDISPOSITION=ORDERLY,
      DONTWAIT,
      ASSOCIATEDDATALENGTH=14,
      ASSOCIATEDDATA=PTR);
```

COBOL74

```
MOVE 1 TO PORTF-SUB.
CLOSE PORTF WITH NO WAIT USING
     CLOSE-DISPOSITION OF ORDERLY
     ASSOCIATED-DATA-LENGTH OF 14
     ASSOCIATED-DATA OF PORTF-MSG.
```

The program is sending 14 characters of user data, starting from PTR, with its orderly close request message. Control returns to the program as soon as possible.

For more information on associated data, see "Understanding the ASSOCIATEDDATA Parameter of OSISESSIONSERVICE" earlier in this section

Section 25
Using BNANATIVESERVICE

BNANATIVESERVICE is the port service native to the BNA network and is provided by the BNA version 2, and Local Port Provider implementations. This port service offers a message-oriented service.

File Attributes Supported by BNANATIVESERVICE

The following file attributes are supported by BNANATIVESERVICE. Table 15–1 contains the attributes supported by each port service.

ACTUALMAXRECSIZE	APPLICATIONGROUP	ATTERR
ATTYPE	ATTVALUE	AVAILABLE
AVAILABLEONLY	BLANK	BLOCKEDTIMEOUT
BLOCKSTRUCTURE	BUFFERS	CENSUS
CHANGEDSUBFILE	CHANGEEVENT	COMPRESSING
COMPRESSION	COMPRESSIONCONTROL	COMPRESSIONREQUESTED
CURRENTRECORDLENGTH	DIALOGCHECKINTERVAL	DIALOGPRIORITY
DONOTSEARCHNETWORK	EXTMODE	FILEEQUATED
FILENAME	FILESTATE	FRAMESIZE
INPUTEVENT	INTERACTIVEFILE	INTMODE
INTNAME	KIND	LASTSUBFILE
LFILENAME	LTITLE	MAXCENSUS
MAXSUBFILES	MYHOST	MYHOSTGROUP
MYNAME	OUTPUTEVENT	PATHNAME
PROVIDERGROUP	REINITIALIZE	REQUESTEDMAXRECSIZE
RESULTLIST	SECURITYTYPE	SERVICE
STATE	SUBFILEERROR	TRANSLATE
TRANSLATING	YOURHOST	YOURHOSTGROUP
YOURNAME	YOURUSERCODE	

If an attribute is invalid for this service, the only value of the attribute considered valid is the default value. Invalid values are as described under "Setting Proper Attribute Values" in Section 15.

BNANATIVESERVICE attributes that do not apply to a provider are ignored by that provider. Refer to Table 15–1 to identify the attributes that are not supported by specific providers.

The attribute ACTUALMAXRECSIZE has the following restrictions for each provider implementation:

Provider	Range Allowed
BNA Version 2 remote dialogue	1 through 20000
BNA Version 2 local dialogue	1 through 65513
LPP	1 through 65513

Statements Supported by BNANATIVESERVICE

Of the set of language statements pertaining to port files, the following are supported by BNANATIVESERVICE:

- Attribute interrogation
- Attribute modification
- OPEN
  - Control option: WAIT/DONTWAIT (NO WAIT in COBOL74) /OFFER/AVAILABLE
  - CONNECTTIMELIMIT
- AWAITOPEN
  - Control option: WAIT/DONTWAIT (NO WAIT in COBOL74)/AVAILABLE
  - CONNECTTIMELIMIT
- READ
  - Message
  - READ length specification
  - WAIT/DONTWAIT (NO WAIT in COBOL74)
- WRITE
  - Message
  - WRITE length specification
  - WAIT/DONTWAIT (NO WAIT in COBOL74)

- CLOSE
  - Control option: WAIT/DONTWAIT (NO WAIT in COBOL74)
  - Close disposition: ABORT
- WAIT

If you use a port statement not included in the preceding list, the system does not invoke the statement and an UNSUPPORTEDPRIMITIVERSLT (168) result and an UNSUPPORTEDPRIMITIVE (41) SUBFILEERROR are returned. If you use a statement parameter or a statement parameter value not included in the preceding list, the system does not invoke the statement and an UNSUPPORTEDPARAMETERRSLT (128) result and an UNSUPPORTEDPARAMETER (18) SUBFILEERROR are returned.

File States Supported by BNANATIVESERVICE

The following is a list of all the possible file states a port file can have using BNANATIVESERVICE:

CLOSED	OFFERED
AWAITINGHOST	AWAITINGOFFER
OPENED	BLOCKED
SHUTTINGDOWN	CLOSEPENDING
DEACTIVATIONPENDING	DEACTIVATED

Figures 25–1 through 25–3 illustrate the file state transitions during the dialogue establishment phase, the data transfer phase, and the dialogue termination phase for BNANATIVESERVICE. Note that a subfile can have multiple file state transitions between CHANGEEVENT and FILESTATE interrogations.

In the figures that follow, the following conventions apply:

- All user-initiated primitives, such as CLOSE ABORT, are in capital letters.
- Other events, such as dialogue not established, are enclosed in square brackets.

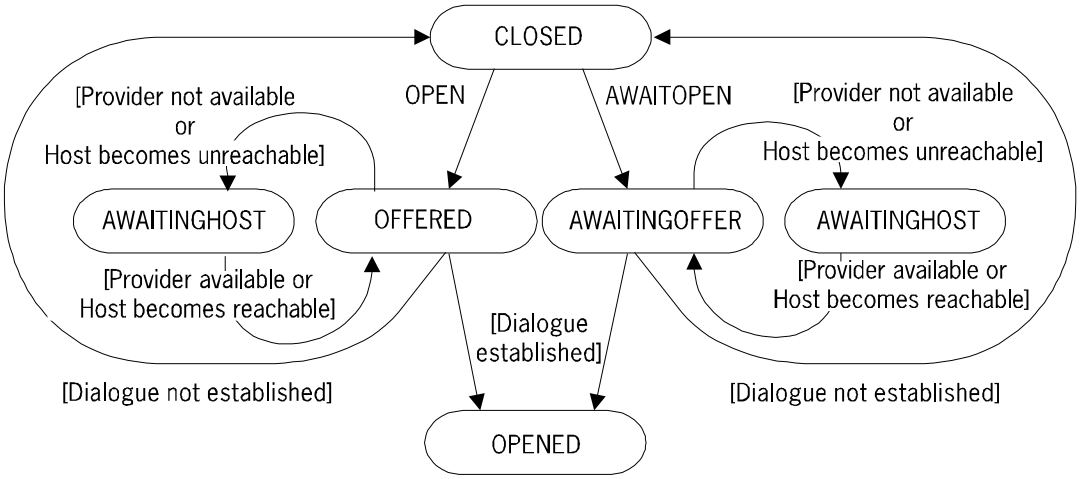


Figure 25–1. BNANATIVESERVICE Dialogue Establishment File State Transitions

If the provider is BNAv2 and the BNAv2 command NW VALIDATE has been set to HOST STRICT (see the *Networking Operations Reference Manual*), and if the remote host has not been declared to BNAv2 with a NW ADD HOST command, then the FILESTA will not stay in AWAITINGHOST state, but will transit back to closed instead (via OFFERED).

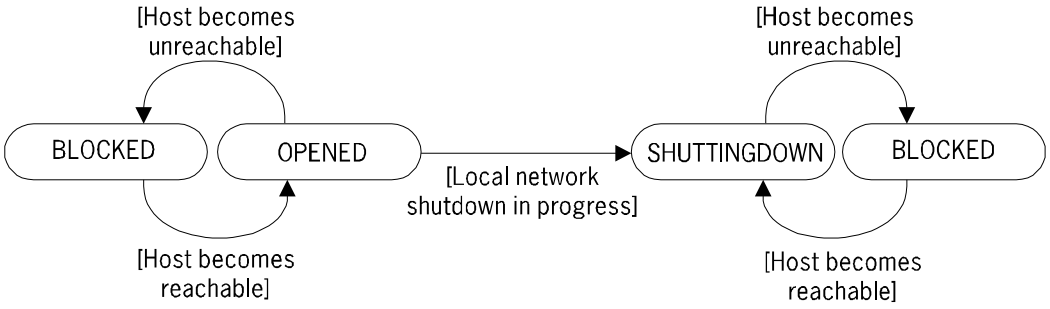


Figure 25–2. BNANATIVESERVICE Probable File State Transitions during Data Transfer

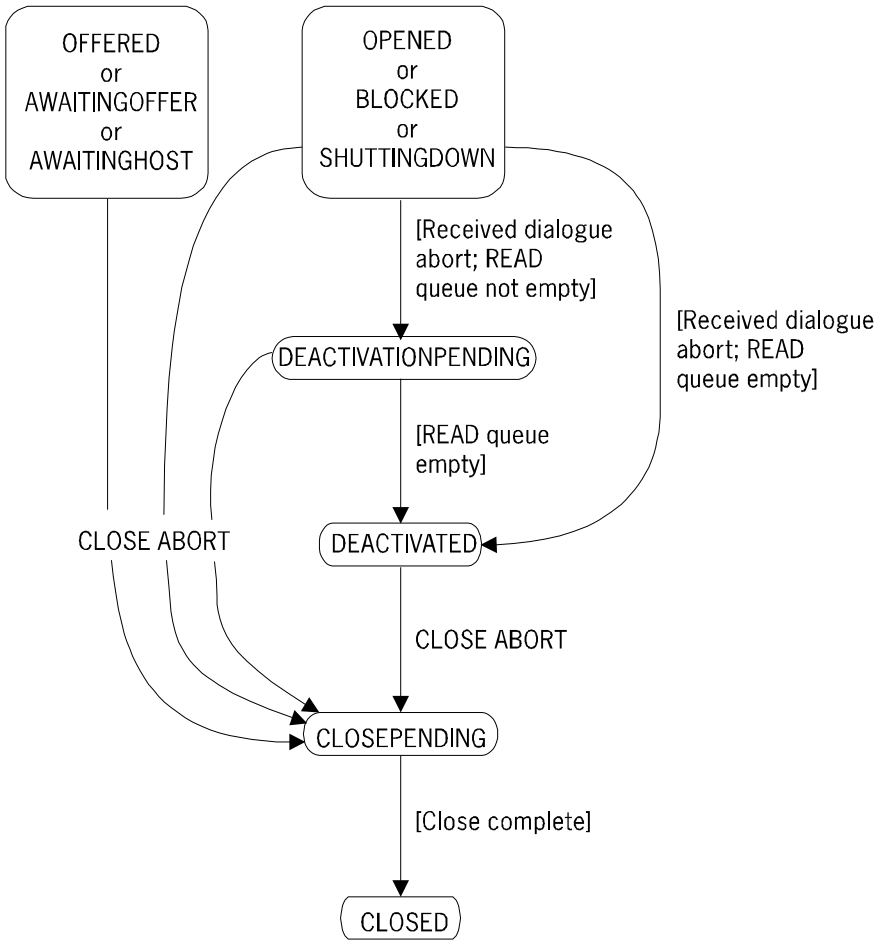


Figure 25–3. BNANATIVESERVICE Dialogue Termination File State Transitions

If the remote host becomes unreachable, the file state is marked as BLOCKED. The file state becomes OPENED if the remote host becomes reachable within a specific time. If the remote host does not become reachable within that time period, the system initiates deactivation of the dialogue.

Use the BLOCKEDTIMEOUT attribute to control the length of time that the subport is to remain blocked before the system initiates deactivation of the dialogue.

When a program invokes an OPEN operation, a dialogue request is issued to the endpoint addressed by YOURNAME and YOURHOST attribute values of the program. When a dialogue request is received by your system, it is matched by the BNANATIVESERVICE provider to a subfile that has invoked an OPEN or AWAITOPEN statement.

If the incoming dialogue request cannot be matched, and the AVAILABLEONLY attribute is FALSE for the request, the system keeps the request for future matching. If the AVAILABLEONLY attribute is TRUE for the incoming request, and the request cannot be immediately matched, it is rejected.

BNANATIVESERVICE uses the following matching attributes to match dialogue requests:

Matching Attribute	Description
MYNAME	The name your program uses for the dialogue.
YOURNAME	The name the correspondent endpoint uses for the dialogue.
MYHOST	The name of the host on which your program is running. This is a read-only attribute.
YOURHOST	The name of the host where the correspondent endpoint is located. A null YOURHOST value matches any incoming MYHOST value.
MYHOSTGROUP	The name of the group of hosts to which the host executing your program belongs. This is a read-only attribute.
YOURHOSTGROUP	The name identifying the host group of the correspondent endpoint.
FILENAME	This is the name of the port file of your program. FILENAME cannot be null, and must match the FILENAME of the correspondent endpoint.
APPLICATIONGROUP	This identifies the user community to which your port belongs. The value of APPLICATIONGROUP must match that of the correspondent endpoint. A null APPLICATIONGROUP matches only other null APPLICATIONGROUP values.
INTMODE	Specifies the internal character size of records used in the port file. For a match to occur, it must be possible for the two endpoints to agree on a character set for the dialogue. If the INTMODE values of the endpoints do not match, they must not both have the attribute TRANSLATE set equal to NOTRANS.
YOURUSERCODE	Specifies the usercode under which the correspondent endpoint must be running when the SECURITYTYPE attribute value for your subfile is PRIVATE. If the SECURITYTYPE value for your subfile is PUBLIC, YOURUSERCODE is not used.

The BNANATIVESERVICE provider obtains the value of the matching attributes for the correspondent endpoint through protocol exchanged during dialogue establishment.

The following information explains the matching criteria for the MYNAME/YOURNAME and the MYHOST/YOURHOST attribute couples, and the YOURUSERCODE attribute.

MYNAME/YOURNAME

In order to match, the YOURNAME attribute value for your endpoint must match the MYNAME attribute value of the correspondent endpoint. If you set the YOURNAME value of your subfile to null, it matches the MYNAME value of any correspondent endpoint. The MYNAME value of your subfile must also match the YOURNAME value of the correspondent endpoint. If the MYNAME value of your subfile is null, it can match only null YOURNAME values for correspondent endpoints.

MYHOST/YOURHOST

The YOURHOST value of your subfile must match the MYHOST value of the correspondent endpoint. A null YOURHOST value matches any MYHOST value.

Two port subfiles that reside on separate hosts and have null YOURHOST values do not match unless the application is using the host independent matching (HIM) feature.

The Local Port Provider (LPP) does not service dialogues when YOURHOST is set to foreign hosts. In this case, LPP returns an UNSUPPORTEDFUNCTION subfile error.

BNA does not service dialogues when YOURHOST is set to local host (for example, local ports). In this case, BNA returns an UNSUPPORTEDFUNCTION subfile error.

YOURUSERCODE

Whether or not YOURUSERCODE is used in dialogue matching depends upon the value of the SECURITYTYPE attribute.

SECURITYTYPE restricts the correspondent endpoints that can match with your subfile. Each host performs security checking for its own subfile as follows:

- If the SECURITYTYPE value of your port file is PUBLIC, security checking is immediately successful, and YOURUSERCODE does not influence matching. After the open action, YOURUSERCODE holds the usercode of the task that declared the correspondent subport so that the application can perform its own security checking.
- If the SECURITYTYPE value of your port file is PRIVATE, the YOURUSERCODE value of the correspondent port must match the usercode of the program offering the correspondent endpoint, or the dialogue is not matched.

Using Host Independent Matching (HIM)

Host independent matching enables a BNA subport to find the endpoint of the complementary subport without the initiating application knowing the name of the other host.

HIM applies only to subports on an enterprise server host with BNA service and requires one additional code module that must be entered into the system with the SL (Support Library) system command. An example of the code module follows:

```
SL HIMSUPPORT = SYSTEM/HIM/SUPPORT : ONEONLY, LINKCLASS = 1
```

The FILENAME attribute of a subport is defined as the RESOURCENAME. The ADD RESOURCE command maps a resource to a list of possible BNA hosts. An application invokes the HIM feature by setting the subport file attribute APPLICATIONCONTEXT to the string "him". When the application opens the subport, BNA checks if HIM is running and proceeds as follows:

- If HIM is not running, the OPEN operation continues with the normal matching methods.
- If HIM is running, the FILENAME value is passed to the Service Resolver library (HIMSUPPORT).

The library returns a list of resource or host names previously defined in the ADD RESOURCE command. The Port Level Manager (PLM) directs an offer to the first host on the list and proceeds as follows:

- If the OPEN operation fails, or no matching subport exists at the other host, then the next host is tried.
- If the subport fails to open after trying all the hosts on the list, the OPEN operation continues with the normal matching methods.

Operations Interface Networking Commands

The following Operations Interface (OI) commands support the HIM feature.

NW ADD RESOURCE

Builds a library of the host names currently providing specific resources or applications. BNA uses this library during subport matching when the port file declaration specifies the HIM feature.

NW MODIFY RESOURCE

Updates a library of the host names currently providing specific resources or applications. A specified resource must already exist in the library. The hosts listed in the command replace those in the library.

NW DELETE RESOURCE

Deletes a resource from the table maintained by the library.

NW RESOURCE

Enables you to inquire on all resource names in the library or on a specific resource.

Refer to the *Networking Operations Reference Manual* for more information.

Example

The following command is entered before running the program:

```
NW ADD RESOURCE PORTTEST AT (HOSTA, HOSTB)
```

The following is an example of a port file declaration in a program:

```
FILE PORTF  ( KIND = PORT

        ,FILENAME = "PORTTEST."
        ,MYNAME = "LOCAL."
        ,YOURNAME = "REMOTE."
        ,APPLICATIONCONTEXT = "HIM."
        ,YOURHOST = "." )
        ;
```

When the program executes an OPEN operation statement for PORTF, BNA determines whether HIM is running and proceeds as follows:

- If HIM is running, an offer for this subport is directed to the first host in the ADDRESOURCE list, HOSTA.
  - If HOSTA does not have a matching candidate available, the offer is refused and the OPEN operation is sent to the second host, HOSTB.
  - If the OPEN operation is also refused at HOSTB, then the OPEN operation is added to the local host's candidate list with the YOURHOST attribute set to null.
- If HIM is not running at the time the OPEN operation is executed, the offer is immediately put on the local candidate list with the YOURHOST attribute set to null.

Establishing a Dialogue Using BNANATIVESERVICE

Dialogue establishment is initiated through the OPEN statement. An OPEN operation causes a dialogue request to be issued. When an AWAITOPEN operation is invoked, the subfile waits for a matching dialogue request to be received. Colliding OPEN requests are resolved into one dialogue in BNANATIVESERVICE.

Dialogue establishment works for BNANATIVESERVICE as described in Section 18, "Establishing a Subfile Dialogue." Additional functionality provided by BNANATIVESERVICE is described in the following subsections.

Using the OPEN Statement with BNANATIVESERVICE

The OPEN statement functions for BNANATIVESERVICE as described in Section 18, "Establishing a Subfile Dialogue." In addition to the functionality described in Section 18, BNANATIVESERVICE also allows the OFFER control option, which is described in this subsection. The OPEN statement syntax is included again here for easy reference.

Example Syntax

ALGOL

```
OPEN (<file name>[SUBFILE <subfile index>],<control
option>,
      <connect time limit option>);
```

COBOL74

```
MOVE <subfile index> TO <file subfile control>.
OPEN <control option> <file name> USING
     <connect time limit>.
```

In addition to the WAIT, DONTWAIT (NO WAIT in COBOL74), and AVAILABLE control options, BNANATIVESERVICE allows the OFFER control option. OFFER indicates that control is returned to your program after the availability of the host specified by the YOURHOST attribute has been determined. Dialogue establishment then continues in parallel with the execution of your program. When an OPEN operation specifies all subfiles, control is returned only after host availability has been determined for all affected subfiles.

Note that determining host availability can cause a significant delay. If your program cannot afford such a delay, you should use the DONTWAIT control option or NO WAIT control option in COBOL74.

The result of the offer is shown through changes in the FILESTATE and SUBFILEERROR attributes for each subfile being opened. If the specified host is available, FILESTATE is set to OFFERED before control is returned to your program. If the specified host is unavailable, the action taken depends upon the value of the AVAILABLEONLY attribute. If AVAILABLEONLY is FALSE, FILESTATE is set to AWAITINGHOST before control is returned.

If AVAILABLEONLY is TRUE and the host specified by YOURHOST is unreachable, then the OPEN operation fails, your program resumes, and an UNREACHABLEHOST (5) SUBFILEERROR is returned. If AVAILABLEONLY is TRUE and no match is found, the OPEN operation fails, your program resumes, and a NOFILEFOUND (4) SUBFILEERROR is returned.

Example

ALGOL

```
RSLT := OPEN ( PORTF[SUBFILE 1], OFFER );
```

COBOL74

```
MOVE 1 TO PORTFX.
OPEN OFFER PORTFX.
IF ATTRIBUTE FILESTATE OF PORTF(1) IS EQUAL TO VALUE
OPENED...
```

Using the AWAITOPEN Statement with BNANATIVESERVICE

Refer to "Using the AWAITOPEN Statement" in Section 18 for an explanation of the AWAITOPEN statement.

Understanding Negotiation during Dialogue Establishment with BNANATIVESERVICE

BNANATIVESERVICE negotiates the following file attributes during dialogue establishment:

Attribute	Description
ACTUALMAXRECSIZE	This attribute determines the maximum data length in FRAMESIZE units that can be handled on READ and WRITE operations.
EXTMODE	This attribute determines the physical character encoding of the records in the port file.
COMPRESSING	This attribute determines whether compression is currently being performed by the dialogue.

ACTUALMAXRECSIZE

ACTUALMAXRECSIZE is negotiated to be the lesser of the REQUESTEDMAXRECSIZE values of the two endpoints.

EXTMODE

EXTMODE is negotiated based upon the values of the INTMODE and TRANSLATE attributes for the correspondent endpoints. If the two values of INTMODE are the same, EXTMODE takes the value of INTMODE and TRANSLATING becomes FALSE. If the two values of INTMODE are different, the result depends upon the values of the TRANSLATE attribute.

If both endpoints specify the NOTRANS value, no match is made and the OPEN operation fails. If either endpoint specifies the USERTRANS or FULLTRANS values, the dialogue match is made. The provider of this service assigns translation responsibility to the endpoint or endpoints that it deems appropriate. For instance, the provider might choose the endpoint with the greatest translation capability. FULLTRANS is assumed to be more capable than USERTRANS, which in turn is assumed to be more capable than NOTRANS.

If one of the endpoints specifies FULLTRANS, TRANSLATING becomes TRUE for that endpoint and FALSE for the correspondent endpoint, and the value of EXTMODE for that endpoint becomes the value of INTMODE for the correspondent endpoint. In this case, the endpoint that has TRANSLATING set to TRUE becomes responsible for the translation. If both subfiles specify FULLTRANS, the assignment of translation responsibility is arbitrary.

If neither endpoint specifies FULLTRANS, but one of the endpoints specifies USERTRANS, TRANSLATING becomes FALSE, and the value of EXTMODE for that endpoint becomes the value of the INTMODE attribute for the correspondent endpoint. In this case, the subfile in which INTMODE is not equal to EXTMODE (that is, the subfile that specifies USERTRANS) is responsible for performing user-level translation of any data read or written. If both subfiles specify USERTRANS, the assignment of translation responsibility is arbitrary.

COMPRESSING

Whether compression of data is possible or not is dependent on the port provider. The COMPRESSING file attribute can be interrogated to determine whether compression is currently in effect for the dialogue.

If compression of data is possible for a provider, use the COMPRESSIONCONTROL attribute to specify whether your program or the provider controls stripping of contiguous characters. If the COMPRESSIONCONTROL value is USER, the default, compression is controlled by the value of the COMPRESSIONREQUESTED file attribute. In this case, the COMPRESSING attribute value reflects the COMPRESSIONREQUESTED value.

The Local Port Provider (LPP) does not compress data. The COMPRESSING attribute is always FALSE for this provider.

For providers that use the BNA network, data compression can be performed on the local host if the network characteristic Port Compression Allowed is TRUE. Port Compression Allowed defaults to TRUE, but can be set to FALSE by using the NW PCA operator command. If Port Compression Allowed is TRUE, the final decision to compress data is made at dialogue establishment time.

For providers that use the BNA network, if the COMPRESSIONCONTROL value is SYSTEM and compression is supported on both hosts, the router manager controls whether compression is performed. The router manager determines if there is to be compression by considering the resistance factors of the network against the compression threshold. The compression threshold can be manipulated by using the *NW COMPRESSION THRESHOLD* operator command. The default of the threshold is 50. In general, compression does not occur on fast connections, but does occur on slower connections.

Exchanging Data Using BNANATIVESERVICE

BNANATIVESERVICE follows the procedures for READ and WRITE operations described in Section 19, "Exchanging Data."

If your program is using BNA Version 2, use the DIALOGPRIORITY file attribute to specify the priority of transmissions from the port subfile relative to the other port subfiles. Your program should specify the 0 (zero) default value for the DIALOGPRIORITY attribute, because the value 1 or 2 causes competition with the BNA Version 2 system software and network management functions.

Tables 25–1 and 25–2 contain the results of the READ or WRITE operation as related to the file state of the subfile on which the operation is being performed.

The way a READ operation is handled by the port service depends upon the file state of the subfile. Table 25–1 describes the way a READ operation is handled on a subfile in each file state supported by BNANATIVESERVICE.

Table 25–1. Effects of File State on the READ Operation for BNANATIVESERVICE

File State	Action
CLOSED	Implicit OPEN
OFFERED	EOF
AWAITINGHOST	EOF
OPENED	If the value of CENSUS is greater than 0, returns input data; otherwise depends on WAIT/DONTWAIT (NO WAIT in COBOL74)
BLOCKED	If the value of CENSUS is greater than 0, returns input data; otherwise depends on WAIT/DONTWAIT (NO WAIT in COBOL74)
SHUTTINGDOWN	If the value of CENSUS is greater than 0, returns input data; otherwise depends on WAIT/DONTWAIT (NO WAIT in COBOL74)
CLOSEPENDING	EOF
DEACTIVATIONPENDING	Returns input data; when READ queue is empty, file state moves to DEACTIVATED
DEACTIVATED	EOF

The result of the WRITE operation depends upon the file state of the subfile you are writing to. Table 25–2 explains the results of the WRITE operation on a subfile in each of the file states supported by BNANATIVESERVICE.

Table 25–2. Effects of File State on the WRITE Operation for BNANATIVESERVICE

File State	Action
CLOSED	Implicit OPEN
OFFERED	EOF
AWAITINGHOST	EOF
OPENED	If OUTPUTEVENT is in the happened state, then WRITE (OK); otherwise, depends on WAIT/DONTWAIT (NO WAIT in COBOL74)
BLOCKED	If OUTPUTEVENT is in the happened state, then WRITE (OK); otherwise, depends on WAIT/DONTWAIT (NO WAIT in COBOL74)
SHUTTINGDOWN	If OUTPUTEVENT is in the happened state, then WRITE (OK); otherwise, depends on WAIT/DONTWAIT (NO WAIT in COBOL74)
CLOSEPENDING	EOF
DEACTIVATIONPENDING	EOF
DEACTIVATED	EOF

Closing a Dialogue Using BNANATIVESERVICE

BNANATIVESERVICE provides ABORT dialogue termination, described in Section 20, "Closing a Dialogue." BNANATIVESERVICE does not support orderly dialogue termination; see "Using ABORT Termination for Orderly Release" in Section 20.

Section 26
Using TCPIPNATIVESERVICE

TCPIPNATIVESERVICE is a data-stream-oriented service offered over the TCP network environment and is provided by the TCP/IP network environment.

File Attributes Supported by TCPIPNATIVESERVICE

The following file attributes are supported by the TCPIPNATIVESERVICE. Table 15–1 contains the attributes supported by each service.

ACTUALMAXRECSIZE	ATTERR	ATTVALUE
ATTYPE	AVAILABLE	BLOCKEDTIMEOUT
CHANGEDSUBFILE	CHANGEEVENT	CURRENTRECORDLENGTH
DIALOGCHECKINTERVAL	DIALOGPRIORITY	FILEEQUATED
FILESTATE	FRAMESIZE	FRAMESIZECENSUS
INPUTEVENT	INTNAME	KIND
LASTSUBFILE	MAXFRAMESIZECENSUS	MAXSUBFILES
MYDOMAINNAME	MYHOST	MYIPADDRESS
MYNAME	OUTPUTEVENT	PASSIVEOPEN
PROVIDERGROUP	RESULTLIST	SERVICE
STATE	SUBFILEERROR	YOURDOMAINNAME
YOURHOST	YOURIPADDRESS	YOURNAME

Note: *Using the AWAITOPEN statement is preferable to setting the PASSIVEOPEN attribute to TRUE. Refer to "Using the AWAITOPEN Statement" in Section 18 for more information.*

If an attribute is not supported by this service, the only value of the attribute considered valid is the default value. An exception is the attribute FILENAME, which can have the value null string.

The following attributes have restrictions on the range of valid values for this service:

Attribute	Valid Range
FRAMESIZE	8. Note that FRAMESIZE defaults to 48.
MYNAME	Null or "0." through "65535."
YOURNAME	Null or "0." through "65535."

Setting attributes to values invalid for this service is handled as described earlier under "Setting Proper Attribute Values" in Section 15.

The ACTUALMAXRECSIZE value is the optimum segment size, in FRAMESIZE units, transmitted by the TCP provider to its peer. Your program can use this information to maximize the utilization of the underlying transmission mechanisms. If your program delivers data to the provider in multiples of ACTUALMAXRECSIZE bytes, the performance of your program might increase.

Port Support for TCPIPNATIVESERVICE

In a TCP/IP network, a port number and an IP address are used to uniquely identify an endpoint. Port numbers are 16-bit positive integers and are partitioned into several ranges. The first range is for reserved ports—numbers less than 1024 are typically used by servers for listening. The reserved ports are further broken down so that the first 255 are used by well-known services. The TCP protocol uses source and destination port numbers to create a TCP dialogue. The source and the destination port numbers must be unique for a new dialogue to be created.

The following combinations are used to define the IP address and port number on MCP systems:

- MYNAME port file attribute and MYIPADDRESS subport file attribute, MYDOMAINNAME port file attribute, or MYHOST port file attribute
- YOURNAME subport file attribute and YOURIPADDRESS subport file attribute, YOURDOMAINNAME subport file attribute, or YOURHOST subport file attribute

For other restrictions on the specification of these attributes for opening a TCP dialogue, refer to "Preparing for Dialogue Establishment Using TCPIPNATIVESERVICE" later in this section. You can also refer to the MYDOMAINNAME, MYHOST, MYIPADDRESS, MYNAME, YOURDOMAINNAME, YOURHOST, YOURIPADDRESS, and YOURNAME attributes in the *File Attributes Reference Manual*.

Statements Supported by TCPIPNATIVESERVICE

Of the set of language statements pertaining to port files, the following are supported by TCPIPNATIVESERVICE:

- GETATTRIBUTE
- SETATTRIBUTE
- OPEN
  - Control option: WAIT/DONTWAIT (NO WAIT in COBOL74)
  - CONNECTIMELIMIT
- AWAITOPEN
  - Control option: WAIT/DONTWAIT (NO WAIT in COBOL74)
  - CONNECTTIMELIMIT
- READ
  - Data
  - READ length specification
  - WAIT/DONTWAIT (NO WAIT in COBOL74)
- WRITE
  - Data
  - WRITE length specification
  - WAIT/DONTWAIT (NO WAIT in COBOL74)
  - URGENT
- CLOSE
  - Close disposition: ABORT/ORDERLY
  - Control option: WAIT/DONTWAIT (NO WAIT in COBOL74)

If you use a port file statement not included in the preceding list, the system does not invoke the statement and an UNSUPPORTEDPRIMITIVERSLT (168) result is returned. If you use a statement parameter or a statement parameter value not included in the preceding list, the system does not invoke the statement and an UNSUPPORTEDPARAMETERRSLT (128) result is returned.

File States Supported by TCPIPNATIVESERVICE

The following table lists all the possible file states a port file can have when it is using TCPIPNATIVESERVICE:

AWAITINGHOST	AWAITINGOFFER
BLOCKED	CLOSED
CLOSEDINPUTPENDING	CLOSEPENDING
CLOSEREQUESTED	CLOSEREQUESTRECEIVED
DEACTIVATED	DEACTIVATIONPENDING
OFFERED	OPENED
URGENTDATAWAITING	

Figures 26–1 through 26–5 illustrate the file state transitions of TCPIPNATIVESERVICE.

In the figures that follow, the following conventions apply:

- All user-initiated primitives, such as CLOSE ABORT, appear in uppercase letters.
- Other events, such as dialogue not established, are enclosed in square brackets.

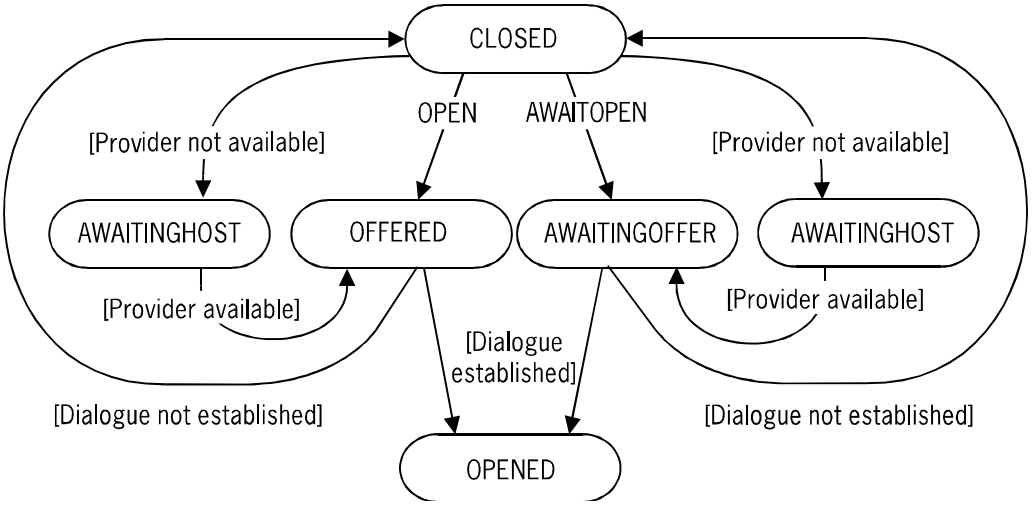


Figure 26–1. TCPIPNATIVESERVICE Dialogue Establishment File State Transitions

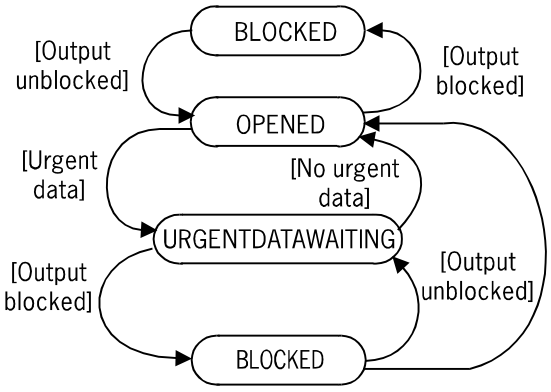


Figure 26–2. TCPIPNATIVESERVICE Probable File State Transitions during Data Transfer

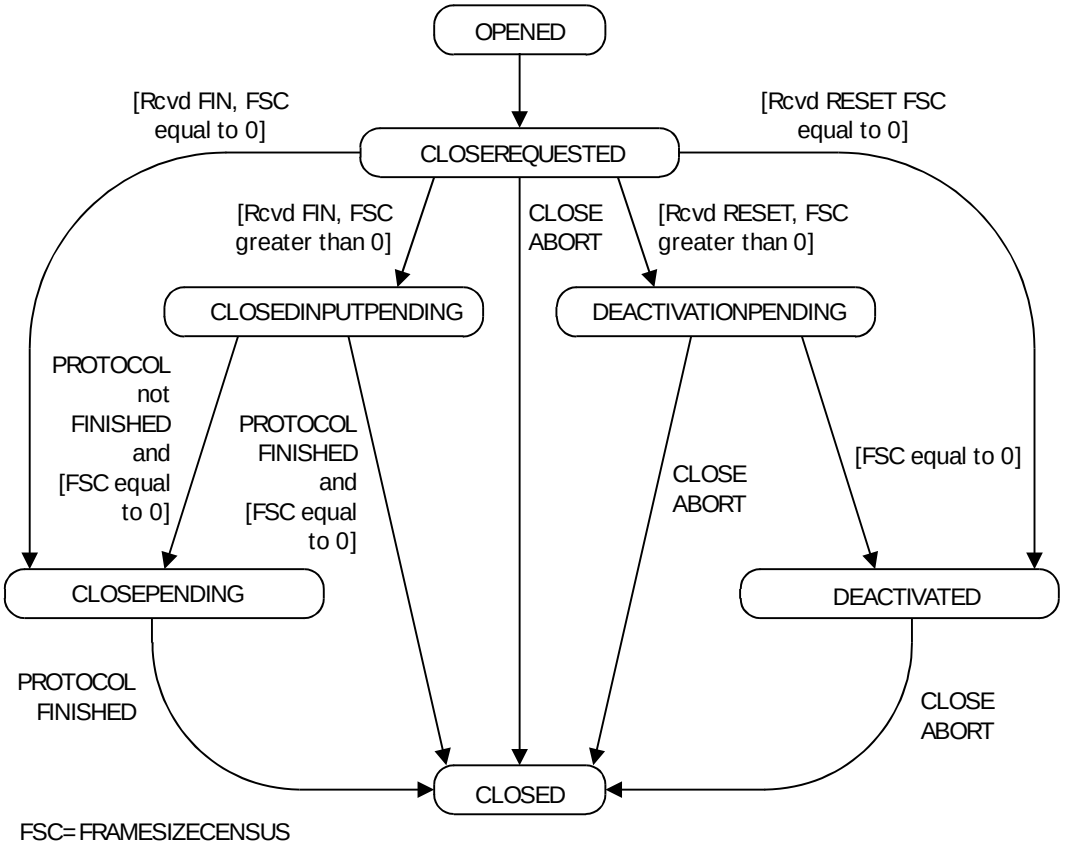


Figure 26–3. TCPIPNATIVESERVICE Dialogue Termination File State Transitions for CLOSEREQUESTED

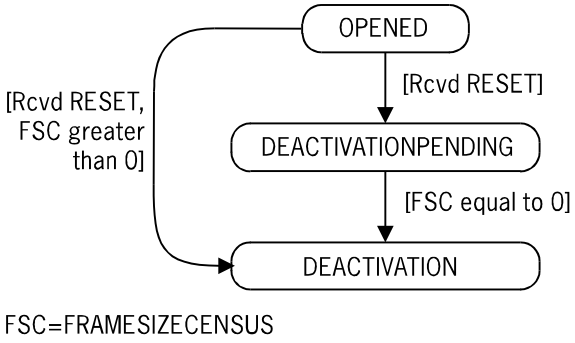


Figure 26–4. TCPIPNATIVESERVICE Dialogue Abnormal Termination File State Transitions for OPEN

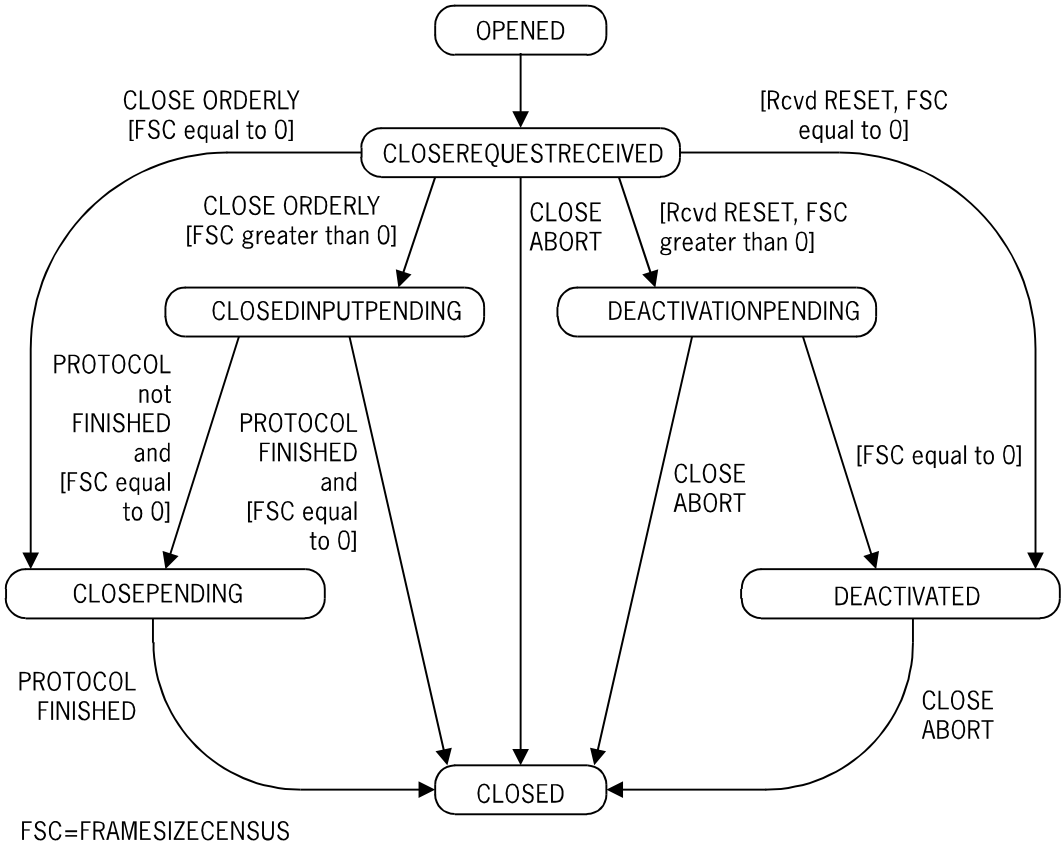


Figure 26–5. TCPIPNATIVESERVICE Dialogue Termination File State Transitions for CLOSEREQUESTRECEIVED

If no data is being exchanged over a connection, you can specify that a "keep-alive" packet be sent to verify that the connection is still open by setting the DIALOGCHECKINTERVAL file attribute value to a nonzero value. If no response is received from the keep-alive packet, the connection goes to the blocked state and remains in that state for the time defined by the BLOCKEDTIMEOUT file attribute value.

The connection terminates after no response to the keep-alive packet is received in the time defined by the BLOCKEDTIMEOUT value. If a response to the keep-alive packet is received, the connection returns to an open state.

Keep-alive packets are sent only when the dialogue is in the open and blocked states.

Preparing for Dialogue Establishment Using TCPIPNATIVESERVICE

TCPIPNATIVESERVICE uses the following matching attributes:

Matching Attribute	Description
MYNAME	The name you are using for the dialogue, which is mapped into the TCP source port address.
MYIPADDRESS	The IP address of the subfile. This must be one of the valid IP addresses for the local host or null.
MYDOMAINNAME	The domain name of the host on which the program is running. This is a read-only attribute. Associated with this name are statically-defined IP addresses.
MYHOST	The name of the host on which your program is running. This is a read-only attribute. Associated with this name are statically-defined IP addresses.
YOURNAME	The name of the correspondent port, which is mapped into the TCP destination port address.
YOURIPADDRESS	The IP address of the correspondent endpoint of the subfile.
YOURDOMAINNAME	The domain name of the host on which the correspondent endpoint is located. This name is used by the MCP environment Resolver to determine the actual location of the host in the network.
YOURHOST	The name of the host on which the correspondent endpoint is located, which is mapped into the destination IP address.

During an OPEN operation, the YOURNAME and MYNAME values of the two endpoints must match. In addition, one of the following values of the two endpoints must match:

- YOURHOST and MYHOST
- YOURIPADDRESS and MYIPADDRESS

  If you specify a YOURDOMAINNAME or a YOURHOST value, the IP address associated with the YOURDOMAINNAME value is obtained as follows:

  TCPIPSUPPORT inquires on its mapping table for an IP address that is mapped to that DOMAINNAME. If a match does not exist in the mapping table, then TCPIPSUPPORT invokes the Resolver to obtain the IP address associated with the specified DOMAINNAME.

  When a TCPIP connection is opened, the TCPIPSUPPORT maps a hostname and domain name to the remote IP address by default. If the IP address is not in the TCPIPSUPPORT internal table, it calls into the Resolver to find the hostname and domain name associated with the IP address. It sends requests to the configured Domain Name Server(s). If a hostname is found before the open completes, YOURHOSTNAME and YOURDOMAINNAME subfile attributes are updated. If the application does not have these attributes updated, you can suppress them by setting DoNotSearchNetwork to TRUE.

When you open the file with an OPEN statement (or PASSIVEOPEN = true), you must at a minimum specify a value for the YOURNAME attribute, and specify a value for one of the following attributes: YOURIPADDRESS, YOURDOMAINNAME, or YOURHOST. If you specify a YOURIPADDRESS value, the YOURDOMAINNAME and YOURHOST values might still be null after the OPEN operation is complete. If you specify a YOURDOMAINNAME or a YOURHOST value, the YOURIPADDRESS value contains the appropriate information after the OPEN operation is complete.

Note: *If your application has a port file open, and the YOURHOST, YOURIPADDRESS, and YOURDOMAINNAME values are all specified, or two are specified, the TCP provider uses the values in the following order: YOURIPADDRESS, YOURDOMAINNAME, and YOURHOST.*

When you open the file with an AWAITOPEN statement, you can assign a null string to the YOURHOST, YOURDOMAINNAME, or YOURIPADDRESS attribute and the YOURNAME attribute. In this case, your port file matches any calling endpoint address when the YOURNAME value of the calling endpoint matches the MYNAME attribute value of your port file.

When a file is opened actively, the following results are true:

Situation	Result
YOURHOST, YOURIPADDRESS, and YOURDOMAINNAME values are null strings.	The OPEN operation fails and the BADATTRIBUTESFOROPEN (13) SUBFILEERROR is returned.
YOURNAME is a null string.	The OPEN operation fails and the BADATTRIBUTESFOROPEN (13) SUBFILEERROR is returned.

Situation	Result
YOURHOST value equals the MYHOST value.	The OPEN operation is successful if the YOURDOMAINNAME and YOURIPADDRESS values are null. If either of those values are specified, YOURHOST is not used for matching.
The MYNAME attribute value is a null string.	The OPEN operation continues and TCP/IP software assigns a port address.
The OPEN operation is requested with values for MYNAME, YOURNAME, and YOURIPADDRESS that are already in use by a connection in use by an existing application on the local host.	The OPEN operation fails and the CONNECTIONINUSE (59) SUBFILEERROR is returned.

When a file is opened passively, the following results are true:

Situation	Result
The YOURHOST attribute value is not a null string, and the YOURNAME attribute value is a null string.	The OPEN operation continues.
The YOURHOST attribute value is a null string, and the YOURNAME attribute value is not a null string.	The OPEN operation continues.
The MYNAME attribute value is a null string.	The OPEN operation fails and the BADATTRIBUTESFOROPEN (13) SUBFILEERROR is returned.

Establishing a Dialogue Using TCPIPNATIVESERVICE

A dialogue is established in the same manner as described in Section 18, "Establishing a Subfile Dialogue," except that the AVAILABLEONLY attribute does not affect TCP.

Using the OPEN Statement with TCPIPNATIVESERVICE

The OPEN statement functions for TCPIPNATIVESERVICE as described in Section 18, under "Using the OPEN Statement," except that the AVAILABLEONLY file attribute is ignored.

Using the AWAITOPEN Statement

TCPIPNATIVESERVICE supports the AWAITOPEN statement.

The AWAITOPEN statement functions for TCPIPNATIVESERVICE as described in Section 18, under "Using the AWAITOPEN Statement," except that the AVAILABLEONLY file attribute value is ignored and TCPIPNATIVESERVICE functions as though AVAILABLEONLY is FALSE.

Exchanging Data Using TCPIPNATIVESERVICE

The methods used to exchange data using TCPIPNATIVESERVICE are the same as those described earlier for data-stream-oriented services in Section 19, "Exchanging Data."

Use the DIALOGPRIORITY attribute to specify the priority of transmissions from the current port subfile relative to other port subfiles. User programs can use the values 0 (zero) through 2. These values have the following significance:

- 0—Routine
- 1—Priority
- 2—Immediate

Some additional considerations are described in this subsection. These considerations involve dealing with data-stream information, using the MAXFRAMESIZECENSUS and FRAMESIZECENSUS attributes, and using the urgent message capability of TCPIPNATIVESERVICE. Note also that the CENSUS and MAXCENSUS attributes do not apply to TCPIPNATIVESERVICE.

Tables 26–1 and 26–2 contain the results of the READ or WRITE operation as related to the file state of the port file on which the operation is being performed.

The way a READ operation is handled by the port service depends upon the file state of the port file. Table 26–1 describes the way a READ operation is handled in each file state supported by TCPIPNATIVESERVICE.

Table 26–1. Effects of File State on the READ Operation for TCPIPNATIVESERVICE

File State	Action
CLOSED	Implicit OPEN
OFFERED	EOF
AWAITINGHOST	EOF
AWAITINGOFFER	EOF
OPENED	If the value of FRAMESIZECENSUS is greater than 0, returns input data; otherwise depends on WAIT/DONTWAIT (NO WAIT in COBOL74)
BLOCKED	If the value of FRAMESIZECENSUS is greater than 0, returns input data; otherwise depends on WAIT/DONTWAIT (NO WAIT in COBOL74)
URGENTDATAWAITING	If the value of FRAMESIZECENSUS is greater than 0, returns input data; otherwise depends on WAIT/DONTWAIT (NO WAIT in COBOL74)
CLOSEREQUESTRECEIVED	Returns input data or EOF
CLOSEREQUESTED	Returns input data or EOF
CLOSEPENDING	EOF
DEACTIVATIONPENDING	Returns input data; when READ queue is empty, port file moves to DEACTIVATED
CLOSEDINPUTPENDING	Returns data; when READ queue is empty, port file is CLOSED or CLOSEPENDING
DEACTIVATED	EOF

The result of the WRITE operation depends upon the file state of the port file you are writing to. Table 26–2 explains the results of the WRITE operation on a port file in each of the file states supported by TCPIPNATIVESERVICE.

Table 26–2. Effects of File State on the WRITE Operation for TCPIPNATIVESERVICE

File State	Action
CLOSED	Implicit OPEN
OFFERED	EOF
AWAITINGHOST	EOF
AWAITINGOFFER	EOF
OPENED	If OUTPUTEVENT is in the happened state, then WRITE (OK); otherwise, depends on WAIT/DONTWAIT (NO WAIT in COBOL74)
BLOCKED	If OUTPUTEVENT is in the happened state, then WRITE (OK); otherwise, depends on WAIT/DONTWAIT (NO WAIT in COBOL74)
URGENTDATAWAITING	If OUTPUTEVENT is in the happened state, then WRITE (OK);otherwise, depends on WAIT/DONTWAIT (NO WAIT in COBOL74)
CLOSEREQUESTRECEIVED	If OUTPUTEVENT is in the happened state, then WRITE (OK); otherwise, depends on WAIT/DONTWAIT (NO WAIT in COBOL74)
CLOSEREQUESTED	EOF
CLOSEPENDING	EOF
DEACTIVATIONPENDING	EOF
CLOSEDINPUTPENDING	EOF
DEACTIVATED	EOF

Understanding Data-Stream-Oriented Data Transfer Using TCPIPNATIVESERVICE

As mentioned before, data-stream-oriented data transfer uses the same methods as message-oriented data transfer. With data-stream-oriented transfer, however, the data is not sent and received in message-size chunks. The amount of data read or written is specified by the data length parameter in the READ or WRITE statement.

Also, TCPIPNATIVESERVICE uses the MAXFRAMESIZECENSUS and the FRAMESIZECENSUS attributes instead of the MAXCENSUS and CENSUS attributes.

For TCPIPNATIVESERVICE, FRAMESIZE must be set to 8; otherwise, the BADATTRIBUTESFOROPEN (13) SUBFILEERROR is returned.

Unlike message-oriented port services, if a TCPIPNATIVESERVICE READ operation does not read the entire data buffer, the data buffer is not deallocated, and the next READ operation starts where the previous one left off. The data buffer is deallocated once all its data has been read.

If a READ operation requests more data than the current data buffer contains, the operation attempts to satisfy the READ request from subsequent data buffers of the port file until the request is satisfied. If the READ operation runs out of data before the request has been fulfilled, the operation is terminated with the data read to that point. It is the responsibility of the program to track the amount of data read.

The following actions occur if a WRITE DONTWAIT operation requests that more data be sent than can be sent by the provider at the moment:

- The number of bytes of data that can be handled are sent.
- The INSUFFICIENTBUFFERS (132) I/O result is returned.
- The [47:20] Field of the STATE file attribute and the CURRENTRECORDLENGTH file attribute value are set to the number of bytes sent by the provider.

To ensure that all data is sent, your program should determine if all the data has been sent. If all the data was not sent, your program must issue another WRITE statement that points to the remaining data to be sent.

Using Urgent Data with TCPIPNATIVESERVICE

TCPIPNATIVESERVICE gives you the ability to indicate that a port file has urgent data that needs to be read from the file.

The value of the FILESTATE attribute of the port file remains URGENTDATAWAITING until all of the urgent data has been read. When all of the urgent data has been read, the FILESTATE value returns to the appropriate state.

Reading Urgent Data with TCPIPNATIVESERVICE

The READ result informs your program that the data received in a READ operation contains urgent data. The READ result is a copy of the STATE file attribute value. If the READ result contains a zero (0) in the [01:01] field and a 42 (URGENT flag) in the [26:10] field, urgent data has been read. The URGENT flag remains on as long as there is urgent data in the user's buffer. When the last byte of urgent data has been read, the READ operation returns to the program without appending nonurgent data even though more data might have been requested with the READ operation.

Data marked as urgent is not given special treatment by TCPIPNATIVESERVICE; it is the responsibility of your program to handle urgent data.

Example 1

ALGOL

```
DEFINE URGENTDATAREAD(RSLT)=
   (NOT RSLT.[01:01] AND (REAL(RSLT.[26:10])=42))#;
RSLT := READ(PORTF,72,IOBUF[*]);
IF URGENTDATAREAD (RSLT) THEN
   PROCESS_URGENT_DATA
ELSE
   .
   .
   .
```

COBOL74

```
01 TEMP1                              USAGE REAL.
01 TEMP2                              USAGE REAL.
01 URGENT-DATA                        USAGE REAL.
.
.
.
MOVE 42 TO URGENT-DATA.
READ PORTF RECORD.
MOVE PORTF-72 TO IOBUF.
MOVE PORTF-FS TO TEMP1.
MOVE TEMP1 TO TEMP2[26:9:10].
IF TEMP2 IS EQUAL TO URGENT-DATA
   PERFORM PROCESS_URGENT_DATA
ELSE
   DISPLAY "NOT URGENT DATA".
STOP RUN.
```

This example shows the way to determine whether incoming data must be checked for the presence of urgent data.

Writing Urgent Data with TCPIPNATIVESERVICE

Your program can indicate the presence of urgent data by using the URGENT parameter of the WRITE statement that is available with TCPIPNATIVESERVICE. Data marked urgent by your program is not given special treatment by the TCPIPNATIVESERVICE provider. When the WRITE operation is invoked, the data is placed in the outgoing byte stream and sent following the windowing rules of the protocol. Urgent data is not placed ahead of any previously written data waiting to be sent.

Example 2

ALGOL	`WRITE (PORTF[URGENT],72,IOBUF);`
COBOL74	`WRITE PORTF-AUX-REC WITH URGENT FROM IOBUF.`

This example shows the way to indicate that urgent data is present. The syntax simply includes the URGENT specification.

Closing a Dialogue Using TCPIPNATIVESERVICE

TCPIPNATIVESERVICE provides ABORT and ORDERLY dialogue release. TCPIPNATIVESERVICE provides ABORT termination through the CLOSE ABORT statement. The CLOSE ABORT statement causes the application to stop reading and writing data and does not guarantee that all data is sent or received. The application receives the following warning message if the provider is aware that data has not been delivered:

```
FILE <port file> AT <host name> CLOSE WARNING: DATA MAY HAVE BEEN LOST
```

TCPIPNATIVESERVICE provides ORDERLY termination through the CLOSE ORDERLY statement. This CLOSE ORDERLY statement causes the application to stop writing data, but allows the application to continue to read data and guarantees the delivery of all data.

Note: *If you are converting a program from TCPNATIVESERVICE to TCPIPNATIVESERVICE and want to retain the same CLOSE operation behavior, use the ORDERLY close disposition instead of the ABORT close disposition.*

Section 27
Using TCPNATIVESERVICE

TCPNATIVESERVICE is a data-stream-oriented service offered over the TCP/IP network.

Note: *This service will be deimplemented in a future release. It is being replaced by TCPIPNATIVESERVICE.*

File Attributes Supported by TCPNATIVESERVICE

Table 15–1 contains the list of attributes supported by each service.

The following file attributes are supported by TCPNATIVESERVICE.

ACTUALMAXRECSIZE	ATTERR	ATTVALUE
ATTYPE	AVAILABLE	BLOCKEDTIMEOUT
CHANGEDSUBFILE	CHANGEEVENT	CURRENTRECORDLENGTH
DIALOGCHECKINTERVAL	FILEEQUATED	FILESTATE
FRAMESIZE	FRAMESIZECENSUS	INPUTEVENT
INTERACTIVEFILE	INTNAME	KIND
LASTSUBFILE	LTITLE	MAXFRAMESIZECENSUS
MAXSUBFILES	MYDOMAINNAME	MYHOST
MYIPADDRESS	MYNAME	OUTPUTEVENT
PASSIVEOPEN	PATHNAME	PROVIDERGROUP
REINITIALIZE	REQUESTEDMAXRECSIZE	RESULTLIST
SERVICE	STATE	SUBFILEERROR
YOURDOMAINNAME	YOURHOST	YOURIPADDRESS
YOURNAME		

Note: *Using the AWAITOPEN statement is preferable to setting the PASSIVEOPEN attribute to TRUE. Refer to "Using the AWAITOPEN Statement" in Section 18 for more information.*

If this service does not support an attribute, the only valid value of the attribute is the default value, except the attribute FILENAME, which can have the value null string.

Also, the following attributes have restrictions on the range of valid values for this service:

Attribute	Valid Range
ACTUALMAXRECSIZE	1 through 65535
FRAMESIZE	8 or 48. A value of 8 is recommended.
MYNAME	Null or "0." through "65535."
SECURITYTYPE	IGNORED
YOURNAME	Null or "0." through "65535."

Note that SECURITYTYPE can default to PRIVATE. You can change the SECURITYTYPE default setting by using the *NW NS SET MIGRATETOBASICSERVICE* system command.

Setting attributes to values invalid for this service is handled as described earlier under "Setting Proper Attribute Values" in Section 15.

Port Support for TCPNATIVESERVICE

In a TCP/IP network, a port number and an IP address are used to uniquely identify an endpoint. Port numbers are 16-bit positive integers and are partitioned into several ranges. The first range is for reserved ports. Numbers less than 1024 are typically used by servers to listen on. The first 255 reserved ports are used by well-known services. The TCP protocol uses source and destination port numbers to create a TCP dialogue. The source and the destination port numbers must be unique for a new dialogue to be created.

The following attribute values are used to define the IP address and port number on MCP systems:

- MYNAME port file attribute and MYIPADDRESS subport file attribute, MYDOMAINNAME port file attribute, or MYHOST port file attribute
- YOURNAME subport file attribute and YOURIPADDRESS subport file attribute, YOURDOMAINNAME subport file attribute, or YOURHOST subport file attribute

For other restrictions on the specification of these attributes for opening a TCP dialogue, refer to "Preparing for Dialogue Establishment Using TCPNATIVESERVICE" later in this section. You can also refer to the MYDOMAINNAME, MYHOST, MYIPADDRESS, MYNAME, YOURDOMAINNAME, YOURHOST, YOURIPADDRESS, and YOURNAME attributes in the *File Attributes Reference Manual*.

Statements Supported by TCPNATIVESERVICE

Of the set of language statements pertaining to port files, the following are supported by TCPNATIVESERVICE:

- GETATTRIBUTE
- SETATTRIBUTE
- OPEN—Control option: WAIT/DONTWAIT (NO WAIT in COBOL74)
- AWAITOPEN
  - You can use the AWAITOPEN statement.
  - Control option: WAIT/DONTWAIT (NO WAIT in COBOL74) are valid.
  - CONNECTTIMELIMIT
- READ
  - Data
  - READ length specification
  - WAIT/DONTWAIT (NO WAIT in COBOL74)
- WRITE
  - Data
  - WRITE length specification
  - WAIT/DONTWAIT (NO WAIT in COBOL74)
  - URGENT
- CLOSE
  - Close disposition:
    - ABORT is valid.
    - ORDERLY is valid, but the action taken is the same as ABORT.
  - Control option: WAIT/DONTWAIT (NO WAIT in COBOL74)

If you use a port statement not included in the preceding list, the system does not invoke the statement and an UNSUPPORTEDPRIMITIVERSLT (168) result is returned. If you use a statement parameter or a statement parameter value not included in the preceding list, the system does not invoke the statement and an UNSUPPORTEDPARAMETERRSLT (128) result is returned.

File States Supported by TCPNATIVESERVICE

The following list shows all possible port-file states.

AWAITINGOFFER	BLOCKED
CLOSED	CLOSEPENDING
DEACTIVATED	DEACTIVATIONPENDING
OFFERED	OPENED
URGENTDATAWAITING	

You can use all the file states in the previous list except AWAITINGHOST and SHUTTINGDOWN.

Figures 27–1 through 27–3 illustrate the file state transitions of TCPNATIVESERVICE. The following conventions apply for these figures:

- All user-initiated primitives, such as CLOSE ABORT, appear in uppercase letters.
- Other events, such as dialogue not established, are enclosed in square brackets.

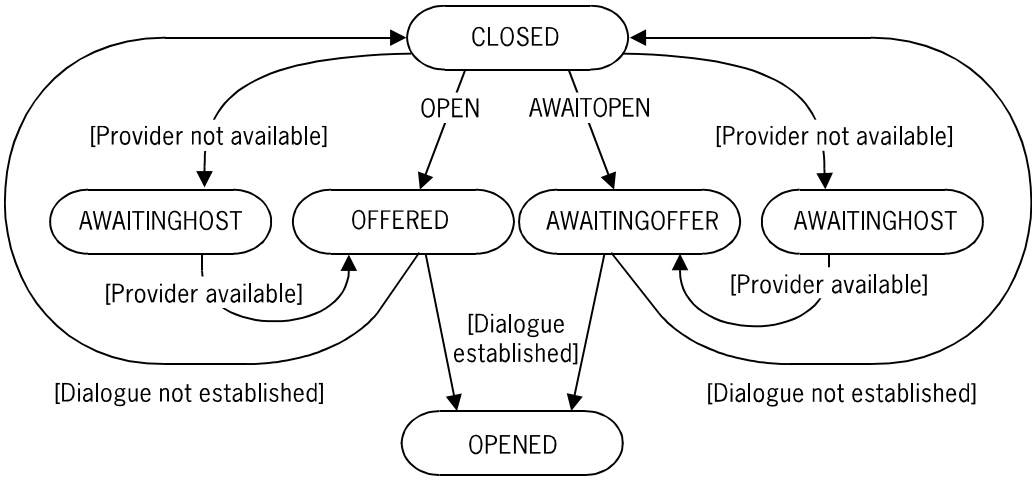


Figure 27–1. TCPNATIVESERVICE Dialogue Establishment File State Transitions

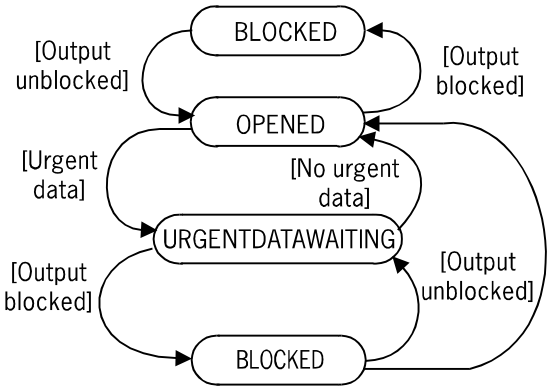


Figure 27–2. TCPNATIVESERVICE Probable File State Transitions during Data Transfer

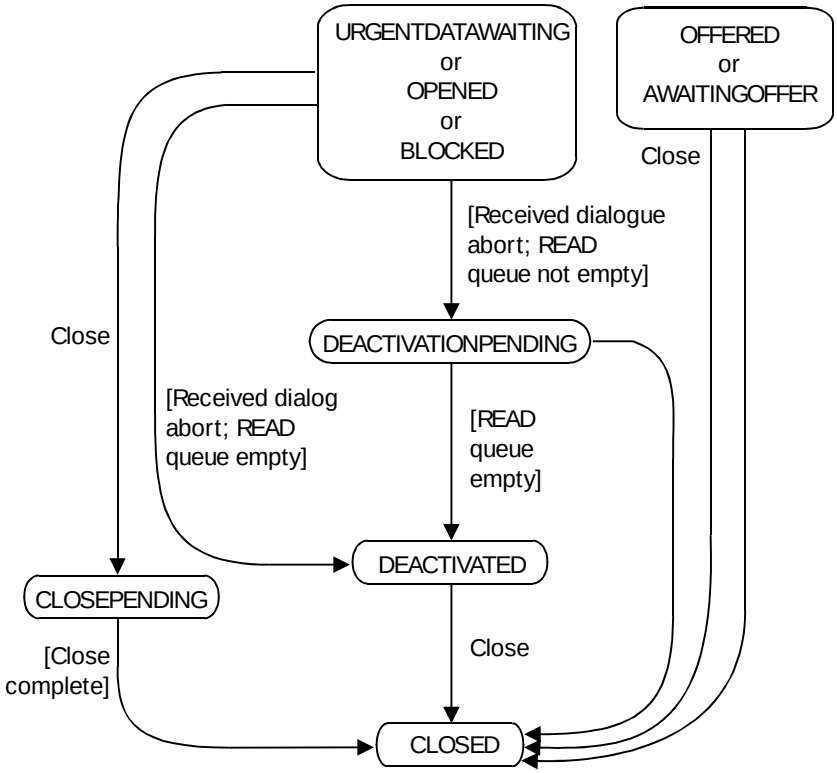


Figure 27–3. TCPNATIVESERVICE Dialogue Termination File State Transitions

If no data is being exchanged over a connection, you can specify that a "keep-alive" packet be sent to verify that the connection is still open by setting the DIALOGCHECKINTERVAL file attribute value to a nonzero value. If no response is received from the keep-alive packet, the connection goes to the blocked state and remains in that state for the time defined by the BLOCKEDTIMEOUT file attribute value.

The connection terminates after no response to the keep-alive packet is received in the time defined by the BLOCKEDTIMEOUT value. If a response to the keep-alive packet is received, the connection returns to an open state.

Keep-alive packets are sent only when the dialogue is in the open and blocked states.

Preparing for Dialogue Establishment Using TCPNATIVESERVICE

TCPNATIVESERVICE uses the following matching attributes:

Matching Attribute	Description
MYNAME	The name you are using for the dialogue, which is mapped into the TCP source port address.
MYIPADDRESS	The IP address of the subfile. This must be one of the valid IP addresses for the local host.
MYDOMAINNAME	The domain name of the host on which the program is running. This is a read-only attribute. Associated with this name is a dynamically-defined IP address.
MYHOST	The name of the host on which your program is running. This is a read-only attribute. Associated with this name is a statically-defined IP address.
YOURNAME	The name of the correspondent port, which is mapped into the TCP destination port address.
YOURIPADDRESS	The IP address of the correspondent endpoint of the subfile.
YOURDOMAINNAME	The domain name of the host on which the correspondent endpoint is located. This name is used by the MCP environment Resolver to determine the actual location of the host in the network.
YOURHOST	The name of the host on which the correspondent endpoint is located, which is mapped into the destination IP address.

During an OPEN operation, the following attribute values of the two endpoints must match:

- YOURNAME and MYNAME and YOURIPADDRESS and MYIPADDRESS
- YOURHOST, MYHOST, and YOURDOMAINNAME are resolved to IP addresses at the TCP provider level. The IP address value is accessible to the application through the YOURIPADDRESS or MYIPADDRESS attribute, or through both of these attributes.

All OPEN operations fail with a BADATTRIBUTESFOROPEN (13) SUBFILEERROR when the PASSIVEOPEN attribute is TRUE and the MYNAME value is a null string.

If you are programming an application and you are opening the file with an OPEN statement, you must at a minimum specify a value for the YOURNAME attribute, and specify a value for one of the following attributes: YOURIPADDRESS, YOURDOMAINNAME, or YOURHOST. If you specify a YOURIPADDRESS value, the YOURDOMAINNAME and YOURHOST values might still be null after the OPEN operation is complete. If you specify a YOURDOMAINNAME or a YOURHOST value, the YOURIPADDRESS value contains the appropriate information after the OPEN operation is complete.

Note: *If your application has a port file open, and the YOURHOST, YOURIPADDRESS, and YOURDOMAINNAME values are all specified, or two are specified, the TCP provider uses the values in the following order: YOURIPADDRESS, YOURDOMAINNAME, and YOURHOST.*

If you are programming an application and you are opening the file with an AWAITOPEN statement, the YOURHOST, YOURDOMAINNAME, or YOURIPADDRESS attribute value and the YOURNAME attribute value of your port file can be null strings. In this case, your port file matches any calling endpoint address when the YOURNAME value of the calling endpoint matches the MYNAME attribute value of your port file.

When a port file is opened actively, the following results are true:

Situation	Result
YOURHOST and YOURNAME values are null strings.	If the YOURDOMAINNAME and YOURIPADDRESS values are null, the OPEN operation fails and the BADATTRIBUTESFOROPEN (13) SUBFILEERROR is returned.
YOURIPADDRESS and YOURNAME values are null strings.	If the YOURDOMAINNAME and YOURHOST values are null, the OPEN operation fails and the BADATTRIBUTESFOROPEN (13) SUBFILEERROR is returned.
YOURDOMAINNAME and YOURNAME values are null strings.	If the YOURIPADDRESS and YOURHOST values are null, the OPEN operation fails and the BADATTRIBUTESFOROPEN (13) SUBFILEERROR is returned.

Situation	Result
YOURHOST value equals the MYHOST value.	The OPEN operation is successful if the YOURDOMAINNAME and YOURIPADDRESS values are null. If those values are specified, YOURHOST is not used for matching.
The MYNAME attribute value is a null string.	The OPEN operation continues.

When a port file is opened passively, the following results are true:

Situation	Result
The YOURHOST attribute value is not a null string, and the YOURNAME attribute value is a null string.	The OPEN/AWAITOPEN operation continues.
The YOURHOST attribute value is a null string, and the YOURNAME attribute value is not a null string.	The OPEN/AWAITOPEN operation continues.
The MYNAME attribute value is a null string.	The OPEN/AWAITOPEN operation fails and the BADATTRIBUTESFOROPEN (13) SUBFILEERROR is returned.
The MYNAME value is supplied for an OPEN operation and conflicts with an existing application on the local host.	The OPEN/AWAITOPEN operation fails and the CONNECTIONINUSE (59) SUBFILEERROR is returned.

Establishing a Dialogue Using TCPNATIVESERVICE

A dialogue request is issued when an OPEN operation is invoked on a port file. When the AWAITOPEN statement is invoked or the PASSIVEOPEN file attribute is set to TRUE before an OPEN operation is invoked, no dialogue request is sent; the port file waits to receive a matching dialogue request.

Using the OPEN Statement with TCPNATIVESERVICE

The OPEN statement functions for TCPNATIVESERVICE as described in Section 18, under "Using the OPEN Statement," except that the AVAILABLEONLY file attribute is ignored.

Using the AWAITOPEN Statement with TCPNATIVESERVICE

If you want to direct your program to wait for requests for dialogue establishment, you can use the AWAITOPEN statement in your program.

Refer to "Using the AWAITOPEN Statement" in Section 18 for information about the AWAITOPEN statement. Note that the AVAILABLEONLY file attribute cannot be used with TCPNATIVESERVICE.

The following ALGOL example uses the PASSIVEOPEN attribute and shows the appropriate code that opens the passive port file on one host and the active port file on another host:

Passive Port File

```
FILE TCPPORT(KIND          = PORT,
             SERVICE       = TCPNATIVESERVICE,
             MYNAME        = "2010.",              % Numeric string
             YOURNAME      = "2010.",              % Numeric string
             YOURHOST      = "MPA3H.",
             PASSIVEOPEN   = TRUE,
             FRAMESIZE     = 8,                  % Byte oriented
             SECURITYTYPE  = PUBLIC);

   .
   .
   .

OPEN(TCPPORT,WAIT);                               % Port now open
```

Active Port File

```
FILE TCPPORT(KIND          = PORT,
             SERVICE       = TCPNATIVESERVICE,
             MYNAME        = "2010.",              % Numeric string
             YOURNAME      = "2010.",              % Numeric string
             YOURHOST      = "MPA3I.",
             PASSIVEOPEN   = FALSE,
             FRAMESIZE     = 8,                  % Byte oriented
             SECURITYTYPE  = PUBLIC);

   .
   .
   .

RSLT := OPEN(TCPPORT,WAIT);                      % Port now open
```

Exchanging Data Using TCPNATIVESERVICE

The methods used to exchange data using TCPNATIVESERVICE are the same as those described earlier for data-stream-oriented services in Section 19, "Exchanging Data." There are some additional considerations, however, which are described in this subsection. These considerations involve dealing with data-stream information, using the MAXFRAMESIZECENSUS and FRAMESIZECENSUS attributes, and using the urgent message capability of TCPNATIVESERVICE. Note also that the CENSUS and MAXCENSUS attributes do not apply to TCPNATIVESERVICE.

Tables 27–1 and 27–2 contain the results of the READ or WRITE operation as related to the file state of the port file on which the operation is being performed.

The way a READ operation is handled by the port service depends upon the file state of the port file. Table 27–1 describes the way a READ operation is handled in each file state supported by TCPNATIVESERVICE.

Table 27–1. Effects of File State on the READ Operation for TCPNATIVESERVICE

File State	Action
CLOSED	Implicit OPEN
OFFERED	EOF
AWAITINGOFFER	EOF
OPENED	If the value of FRAMESIZECENSUS is greater than 0, returns input data; otherwise depends on WAIT/DONTWAIT (NO WAIT in COBOL74)
BLOCKED	If the value of FRAMESIZECENSUS is greater than 0, returns input data; otherwise depends on WAIT/DONTWAIT (NO WAIT in COBOL74)
URGENTDATAWAITING	If the value of FRAMESIZECENSUS is greater than 0, returns input data; otherwise depends on WAIT/DONTWAIT (NO WAIT in COBOL74)
CLOSEPENDING	EOF
DEACTIVATIONPENDING	Returns input data; when READ queue is empty, port file moves to DEACTIVATED
DEACTIVATED	EOF

The result of the WRITE operation depends upon the file state of the port file you are writing to. Table 27–2 explains the results of the WRITE operation on a port file in each of the file states supported by TCPNATIVESERVICE.

Table 27–2. Effects of File State on the WRITE Operation for TCPNATIVESERVICE

File State	Action
CLOSED	Implicit OPEN
OFFERED	EOF
AWAITINGOFFER	EOF
OPENED	If OUTPUTEVENT is in the happened state, then WRITE (OK); otherwise, depends on WAIT/DONTWAIT (NO WAIT in COBOL74)
BLOCKED	If OUTPUTEVENT is in the happened state, then WRITE (OK); otherwise, depends on WAIT/DONTWAIT (NO WAIT in COBOL74)
URGENTDATAWAITING	If OUTPUTEVENT is in the happened state, then WRITE (OK); otherwise, depends on WAIT/DONTWAIT (NO WAIT in COBOL74)
CLOSEPENDING	EOF
DEACTIVATIONPENDING	EOF
DEACTIVATED	EOF

Understanding Data-Stream-Oriented Data Transfer Using TCPNATIVESERVICE

As mentioned before, data-stream-oriented data transfer uses the same methods as message-oriented data transfer. With data-stream-oriented transfer, however, the data is not sent and received in message-size chunks. The amount of data read or written is specified by the data length parameter in the READ or WRITE statement. Note that the data length cannot exceed ACTUALMAXRECSIZE, and the amount of data read or written can be less than that specified in the READ or WRITE statement, because of system or network limitations.

Also, TCPNATIVESERVICE uses the MAXFRAMESIZECENSUS and the FRAMESIZECENSUS attributes instead of the MAXCENSUS and CENSUS attributes.

For TCPNATIVESERVICE, FRAMESIZE should be set to 8. Note that FRAMESIZE defaults to 48.

Unlike message-oriented port services, if a TCPNATIVESERVICE READ operation does not read the entire data buffer, the data buffer is not deallocated, and the next READ operation starts where the previous one left off. The data buffer is deallocated once all its data has been read.

If a READ operation requests more data than the current data buffer contains, the operation attempts to satisfy the READ request from subsequent data buffers of the port file until the request is satisfied. If the READ operation runs out of data before the request has been fulfilled, the operation is terminated with the data read to that point. It is the responsibility of the programmer to track the amount of data read.

Using Urgent Data with TCPNATIVESERVICE

TCPNATIVESERVICE gives you the ability to indicate that a port file has urgent data that needs to be read from the file.

The value of the FILESTATE attribute of the port file remains URGENTDATAWAITING until all of the urgent data has been read. When all of the urgent data has been read, the FILESTATE value returns to the appropriate state.

Reading Urgent Data with TCPNATIVESERVICE

The READ result informs your program that the data received in a READ operation contains urgent data. The READ result is a copy of the STATE file attribute value. If the READ result contains a zero (0) in the [01:01] field and a 42 (URGENT flag) in the [26:10] field, urgent data has been read. The URGENT flag remains on as long as there is urgent data in the user's buffer. When the last byte of urgent data has been read, the READ operation returns to the program without appending nonurgent data even though more data might have been requested with the READ operation.

Data marked as urgent is not given special treatment by TCPNATIVESERVICE; it is the responsibility of your program to handle urgent data.

Example 1

ALGOL

```
DEFINE URGENTDATAREAD(RSLT)=
   (NOT RSLT.[1:1] AND (REAL(RSLT.[26:10])=42))#;
RSLT := READ(PORTF,72,IOBUF[*]);
IF URGENTDATAREAD (RSLT) THEN
   PROCESS_URGENT_DATA
ELSE
   .
   .
   .
```

COBOL74

```
01 TEMP1                              USAGE REAL.
01 TEMP2                              USAGE REAL.
01 URGENT-DATA                        USAGE REAL.
.
.
.
MOVE 42 TO URGENT-DATA.
READ PORTF RECORD.
MOVE PORTF-72 TO IOBUF.
MOVE PORTF-FS TO TEMP1.
MOVE TEMP1 TO TEMP2[26:9:10].
IF TEMP2 IS EQUAL TO URGENT-DATA
   PERFORM PROCESS_URGENT_DATA
ELSE
   DISPLAY "NOT URGENT DATA".
STOP RUN.
```

This example shows the way to determine whether incoming data must be checked for the presence of urgent data.

Writing Urgent Data with TCPNATIVESERVICE

Your program can indicate the presence of urgent data by using the URGENT parameter of the WRITE statement that is available with TCPNATIVESERVICE. Data marked urgent by your program is not given special treatment by the TCPNATIVESERVICE provider. When the WRITE operation is invoked, the data is placed in the outgoing byte stream and sent following the windowing rules of the protocol. Urgent data is not placed ahead of any previously written data waiting to be sent.

Example 2

ALGOL

```
WRITE (PORTF[URGENT],72,IOBUF);
```

COBOL74

```
WRITE PORTF-AUX-REC WITH URGENT FROM IOBUF.
```

This example shows the way to indicate that urgent data is present. The syntax simply includes the URGENT specification.

Closing a Dialogue Using TCPNATIVESERVICE

TCPNATIVESERVICE provides ABORT and ORDERLY dialogue releases. However, when using ABORT, the action taken is the same as ORDERLY.

The default disposition of a CLOSE statement is ABORT.

Section 28
Using NETBIOSSESSIONSERVICE

NETBIOSSESSIONSERVICE is a message-oriented service that enables applications in the MCP environment to communicate with PC applications on Novell NetWare local area networks (LANs).

NETBIOSSESSIONSERVICE makes use of the NetBIOS interface definition, a communications protocol that is supported by NetWare. This section includes some information about how programming constructs in the MCP environment map onto NetBIOS fields and commands. However, this section does not explain how to write PC applications that can communicate with NETBIOSSESSIONSERVICE applications in the MCP environment.

File Attributes Supported by NETBIOSSESSIONSERVICE

The following port file attributes are supported by NETBIOSSESSIONSERVICE:

ACTUALMAXRECSIZE	ATTERR	ATTVALUE
ATTYPE	AVAILABLE	AVAILABLEONLY
BLOCKSTRUCTURE	BUFFERS	CENSUS
CHANGEDSUBFILE	CHANGEEVENT	COMPRESSING
CURRENTRECORDLENGTH	FILEEQUATED	FILENAME
FILESTATE	FRAMESIZE	INPUTEVENT
INTNAME	INTERACTIVEFILE	KIND
LASTSUBFILE	LTITLE	MAXCENSUS
MAXSUBFILES	MYNAME	NETACCESSPOINT
OUTPUTEVENT	PASSIVEOPEN	PATHNAME
PROVIDERGROUP	REINITIALIZE	REQUESTEDMAXRECSIZE
RESULTLIST	SECURITYTYPE	SERVICE
STATE	SUBFILEERROR	YOURNAME

If an attribute is not supported by this service, the only value of the attribute considered valid is the default value.

NETBIOSSESSIONSERVICE imposes restrictions on the range of valid values for the following supported attributes:

Attribute	Valid Range
COMPRESSING	Always returns FALSE.
MYNAME	Refer to "Understanding the MYNAME and YOURNAME File Attributes" later in this section.
NETACCESSPOINT	Identifier, Null or *NONE.
REQUESTEDMAXRECSIZE	65435.
SECURITYTYPE	PUBLIC.
YOURNAME	Refer to "Understanding the MYNAME and YOURNAME File Attributes" later in this section.

Note that NETBIOSSESSIONSERVICE does not use the FILENAME attribute for subfile matching. However, you can assign a value to FILENAME, and this value appears in system log entries and in the output from the *NW HLCN SHOW SUBPORT* system command.

Note that SECURITYTYPE defaults to PRIVATE if the default setting has not been changed by using the *NW NS SET MIGRATETOBASICSERVICE* system command.

Setting these attributes to values invalid for this service is handled as described under "Setting Proper Attribute Values" in Section 15. For a complete list of the attributes supported by each port service, refer to Table 15–1 in Section 15, "Using Attributes."

Statements Supported by NETBIOSSESSIONSERVICE

Of the set of language statements pertaining to port files, the following are supported by NETBIOSSESSIONSERVICE:

- GETATTRIBUTE
- SETATTRIBUTE
- OPEN
  - Control option: WAIT/DONTWAIT (NO WAIT in COBOL74)/AVAILABLE
  - CONNECTTIMELIMIT
- AWAITOPEN
  - Control option: WAIT/DONTWAIT (NO WAIT in COBOL74)/AVAILABLE
  - CONNECTTIMELIMIT
- READ
  - Message
  - READ length specification
  - WAIT/DONTWAIT (NO WAIT in COBOL74)
- WRITE
  - Message
  - WRITE length specification
  - WAIT/DONTWAIT (NO WAIT in COBOL74)
- CLOSE
  - Close disposition: ABORT
  - Control option: WAIT/DONTWAIT (NO WAIT in COBOL74)

If you use a port statement that is not included in the preceding list, the system does not invoke the statement, and an UNSUPPORTEDPRIMITIVERSLT (168) result and an UNSUPPORTEDPRIMITIVE (41) SUBFILEERROR are returned. If you use a statement parameter or a statement parameter value that is not included in the preceding list, the system does not invoke the statement, and an UNSUPPORTEDPARAMETERRSLT (128) and an UNSUPPORTEDPARAMETER (18) SUBFILEERROR are returned.

File States Supported by NETBIOSSESSIONSERVICE

The following is a list of all the possible file states a port file can have using NETBIOSSESSIONSERVICE:

CLOSED

OFFERED

AWAITINGHOST

AWAITINGOFFER

OPENED

SHUTTINGDOWN

CLOSEPENDING

DEACTIVATIONPENDING

DEACTIVATED

Figures 28–1 through 28–3 illustrate the file state transitions for NETBIOSSESSIONSERVICE. The following conventions apply for these figures:

- All user-initiated primitives, such as CLOSE ABORT, are in capital letters.
- Other events, such as [Dialogue not established], are enclosed in square brackets.

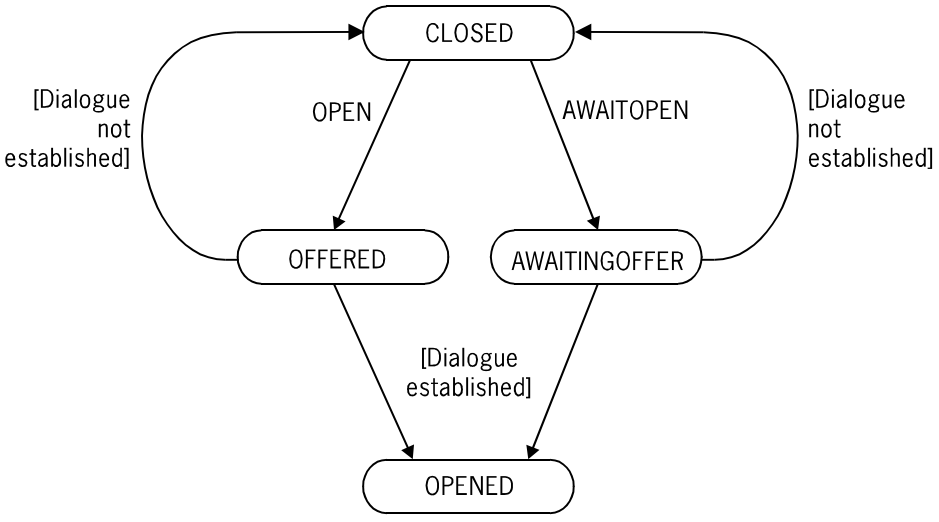


Figure 28–1. NETBIOSSESSIONSERVICE Dialogue Establishment File State Transitions

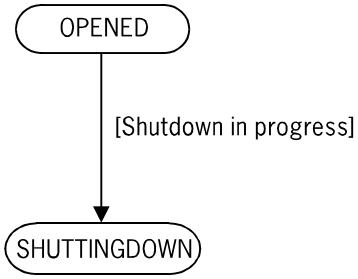


Figure 28–2. NETBIOSSESSIONSERVICE Possible File State Transitions During Data Transfer

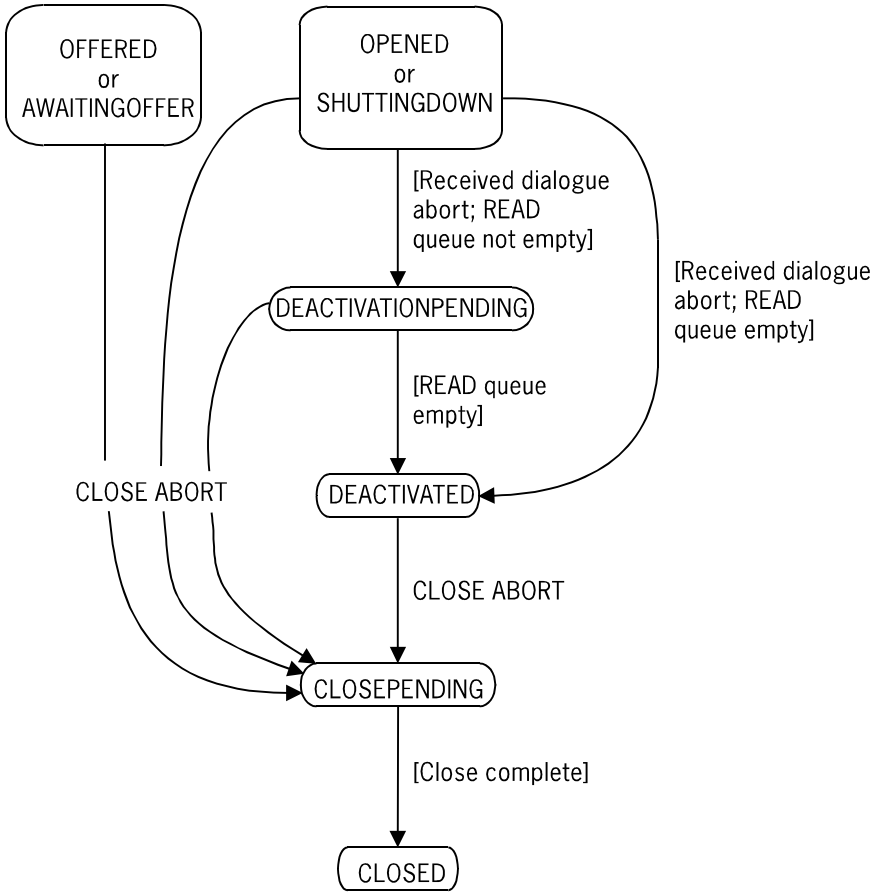


Figure 28–3. NETBIOSSESSIONSERVICE Dialogue Termination File State Transition

Preparing for Dialogue Establishment Using NETBIOSSESSIONSERVICE

During an OPEN operation, NETBIOSSESSIONSERVICE uses the following attributes for subfile matching:

Matching Attribute	Description
MYNAME	The name you are using for this dialogue. The MYNAME value must match the value of the YOURNAME attribute of the correspondent endpoint, or the value of the YOURNAME attribute of the correspondent endpoint must be null.
YOURNAME	The name of the correspondent endpoint. The YOURNAME value must match the value of the MYNAME attribute of the correspondent endpoint.
NETACCESSPOINT	The name of the network access point used to establish the dialogue. The NETACCESSPOINT subfile attribute must name a network access point that is directly attached to the system. The correspondent endpoint must be reachable by way of the specified network access point.

During an AWAITOPEN operation, NETBIOSSESSIONSERVICE uses the following attributes for subfile matching:

Matching Attribute	Description
MYNAME	The name you are using for the dialogue. The value of the MYNAME attribute must match the value of the YOURNAME attribute of the correspondent endpoint.
YOURNAME	The name of the correspondent endpoint. The value of the YOURNAME attribute must either be null or must match the value of the MYNAME attribute of the correspondent endpoint. If the YOURNAME value is null, the system sets the value to the actual MYNAME value of the matching endpoint when the file state becomes OPENED.
NETACCESSPOINT	The name of the network access point used to establish the dialogue. The NETACCESSPOINT attribute must either be null or must name a network access point that is directly attached to the system. If the NETACCESSPOINT attribute is null, the system sets the value to the actual network access point used when the file state changes to OPENED.

NETBIOSSESSIONSERVICE does not support the YOURHOST and YOURHOSTGROUP attributes, because all endpoints on a NetBIOS network are directly accessible regardless of the host they are on. NETBIOSSESSIONSERVICE also does not use the FILENAME or APPLICATIONGROUP attributes for subfile matching.

For further information about the subfile matching attributes used by NETBIOSSESSIONSERVICE, refer to "Understanding the MYNAME and YOURNAME File Attributes" and "Understanding NETBIOSNAMEINUSERSLT Errors" later in this section.

Understanding the MYNAME and YOURNAME File Attributes

The MCP environment system software maps the value of the MYNAME attribute to the NCB_NAME field of the network control block (NCB) in NetBIOS. Similarly, the YOURNAME attribute is mapped to the NCB_CALLNAME field of the NCB. NETBIOSSESSIONSERVICE supports unusual restrictions and enhancements to the MYNAME and YOURNAME attribute syntax to ensure that these values can be mapped directly to the corresponding NetBIOS field names.

In NETBIOSSESSIONSERVICE, the values assigned to the MYNAME and YOURNAME attributes are considered to be byte strings rather than character strings. These values are not required to be made up of EBCDIC or ASCII characters and can include hexadecimal 4"00" (EBCDIC null) values, hexadecimal 4"4B" (EBCDIC period) values, and other unusual values.

NETBIOSSESSIONSERVICE enforces the following restrictions on the syntax of the MYNAME and YOURNAME values:

- The value cannot exceed 16 bytes in length.

  Note: *If you assign MYNAME or YOURNAME values that are shorter than 16 bytes, the system pads the corresponding NCB_NAME and NCB_CALLNAME fields with trailing null characters. Some PC applications might pad these fields with trailing blanks instead of trailing null characters. This difference could cause two endpoints of a subfile to fail to match. You can prevent this possibility by always assigning full 16-byte values in both the PC and mainframe applications.*

- The leading byte should not be hexadecimal 4"00" (EBCDIC null). This value is reserved for use by system software.
- If the MYNAME value has a zero length or begins with a hexadecimal 4"2A" (ASCII asterisk), the system replaces the MYNAME value with a *permanent node name*. The permanent node name is a system-generated name that is guaranteed to be unique.

  The following is an example of a situation in which the permanent node name can be useful. Suppose there is a port file application that serves many clients. This server application uses a MYNAME value that is known to all the clients, and uses a null YOURNAME value for all its subfiles. To match with these subfiles, the client applications need only open an endpoint with a YOURNAME value equal to the MYNAME of the server, and a unique MYNAME value (the permanent node name).

> ***Note:*** *An application cannot interrogate its own permanent node name. If the application interrogates the MYNAME attribute, the system returns the value originally assigned by the program (rather than the permanent node name). However, the matching application can determine the permanent node name by interrogating the YOURNAME attribute after the subfile is opened.*

When a program assigns a value to a character string valued attribute, the system normally copies the characters only until a hexadecimal 4"4B" (EBCDIC period) or a hexadecimal 4"00" (EBCDIC null) occurs. However, NETBIOSSESSIONSERVICE allows these byte values to be embedded within the MYNAME and YOURNAME attribute values. To make the use of these values possible, NETBIOSSESSIONSERVICE supports a special encoded form for the MYNAME and YOURNAME attributes.

In most cases, your use of this special encoded form is optional. However, you must use the special encoded form if you assign the MYNAME or YOURNAME attribute a value that includes a hexadecimal 4"4B" (EBCDIC period) or hexadecimal 4"00" (EBCDIC null), or a value that begins with hexadecimal 4"7F" (EBCDIC quotation mark). You must first encode such a value in the following way:

- Add a hexadecimal 4"7F" (EBCDIC quotation mark) at the beginning and the end of the value.
- If any hexadecimal 4"7F" (EBCDIC quotation mark) bytes occur within the string value, you must repeat the byte so that it appears as hexadecimal 4"7F7F".

Various mechanisms exist in programming languages to construct a value containing quotation marks and nonprintable characters. For example, the following ALGOL program fragment assigns the YOURNAME attribute a byte string consisting of the ASCII characters that spell out *ASCIINAME*:

```
ARRAY      TEMPATTR [0:3];
REPLACE TEMPATTR[0] BY """, 7"ASCIINAME", """".";
REPLACE PORTF.YOURNAME BY POINTER(TEMPATTR[0]);
```

The following COBOL74 program fragment assigns YOURNAME the same value as that assigned by the preceding ALGOL example. The ASCII letters for *ASCIINAME* are represented by their hexadecimal codes, enclosed between at sign (@) characters.

```
WORKING-STORAGE SECTION.
01  TEMP-VALUE PIC X(17).
PROCEDURE DIVISION.
ONLY-HEADER.
STRING """" FOR 1,
       @41534349494E414D45@ FOR 9,
       """." FOR 2
     INTO TEMP-VALUE.
CHANGE ATTRIBUTE YOURNAME OF MYFILE TO TEMP-VALUE.
```

When a program interrogates MYNAME or YOURNAME, the system returns the value in the encoded form if and only if the value includes one or more of the hexadecimal values that require special encoding.

Note: *If the MYNAME or YOURNAME attribute was originally assigned in encoded form, but the value did not actually require special encoding, the system does* not *return the value in encoded form.*

Establishing a Dialogue Using NETBIOSSESSIONSERVICE

When an application in the MCP environment initiates an OPEN operation on a subfile, the system issues a NetBIOS *CALL* command. If an application in the MCP environment initiates an AWAITOPEN operation on a subfile, the system issues a NetBIOS *LISTEN* command.

For two endpoints to match, one endpoint must first issue a NetBIOS *LISTEN* command, and the other endpoint must then issue a NetBIOS *CALL* command. For example, an application in the MCP environment can initiate an AWAITOPEN operation and the matching PC application can then issue a NetBIOS *CALL* command. Alternatively, the PC application could issue a NetBIOS *LISTEN* command and the matching application in the MCP environment could then initiate an OPEN operation.

If both endpoints send CALL commands, the system treats them as separate requests. That is, PC applications that issue NetBIOS *CALL* commands are not matched to applications in the MCP environment that initiate OPEN operations.

Using the OPEN Statement with NETBIOSSESSIONSERVICE

The OPEN statement works as described under "Using the OPEN Statement" in Section 18, although the effects of the AVAILABLEONLY file attribute vary slightly from that description.

If AVAILABLEONLY is set to TRUE, the system makes a single attempt to establish a connection with the corresponding endpoint. However, the NetBIOS protocol includes several delays and retries even for this single attempt. If the AVAILABLEONLY attribute is TRUE and the OPEN option is DONTWAIT (NO WAIT in COBOL74), the file state is set to OFFERED and control is returned to the program. If no connection can be made, the file state changes to CLOSED.

If AVAILABLEONLY is set to FALSE, the system repeatedly attempts to establish a connection with the corresponding endpoint until either a connection is made and the file state is set to OPENED or the CONNECTTIMELIMIT expires.

Using the AWAITOPEN Statement with NETBIOSSESSIONSERVICE

The AWAITOPEN statement works as described under "Using the AWAITOPEN Statement" in Section 18, although the effect of the AVAILABLEONLY file attribute is different. If AVAILABLEONLY is set to TRUE, the AWAITOPEN statement always fails with a NOFILEFOUNDRSLT(2) OPEN result and a NOFILEFOUND (4) SUBFILEERROR. This limitation results from the fact that the HLCN software in the MCP environment does not keep a list of unmatched incoming offers.

Understanding NETBIOSNAMEINUSERSLT Errors

One type of error that is unique to NETBIOSSESSIONSERVICE can occur during an OPEN or AWAITOPEN operation. When this error occurs, an open error of 228 (NETBIOSNAMEINUSERSLT) is returned and the SUBFILEERROR attribute returns the value 54 (NETBIOSNAMEINUSE).

The usual meaning of this error is that another application on the LAN is already attempting to open an endpoint with a MYNAME file attribute value equal to the MYNAME value specified by your application. If this is the case, then you can remedy the situation by changing your application to use a different MYNAME value, and changing the matching application to use a different YOURNAME value.

If this remedy does not resolve the error, then you must consider whether the error is caused by an overlap in the network access point definitions.

A network access point is the combination of a local area network (LAN) and a particular network processor that is used to access that LAN. Network access points are defined by the network administrator with the *NW HLCN ADD NETACCESSPOINT <identifier> (NP = <np#>)* system command. In this command, the <identifier> construct specifies the name of the network access point. You can specify this name as the value of the NETACCESSPOINT subfile attribute.

The system treats each network access point as if it accessed a separate LAN. However, the network administrator can define different network access points that access the same physical LAN. The same MCP system can have multiple network access point definitions that overlap in this way. In this case, each network access point accesses the LAN through a different network processor. Such an overlap in network access point definitions can be desirable in some cases, for performance or configuration reasons.

However, suppose that an application on the MCP host attempts to open a port subfile with NETBIOSSESSIONSERVICE, using the AWAITOPEN operation and with the NETACCESSPOINT attribute set to a null string. In this case, the system notifies all network access points of the AWAITOPEN request. However, if two network access points use the same physical LAN, then that physical LAN receives two concurrent offers of an endpoint with the same MYNAME value. The NetBIOS protocol treats these as separate requests, and returns the error.

One way you can solve this problem is by assigning a specific network access point name to the NETACCESSPOINT attribute. In this case, the LAN receives only one subfile open offer.

Alternatively, the network administrator could solve the problem by dividing the LAN into separate physical LANs that are each accessed through a separate network processor and a separate network access point definition.

Exchanging Data Using NETBIOSSESSIONSERVICE

The methods used to exchange data using NETBIOSSESSIONSERVICE are the same as those described in Section 19, "Exchanging Data."

The following two tables contain the results of the READ or WRITE operation as related to the file state of the port subfile on which the operation is being performed.

The way a READ operation is handled by the port service depends on the file state of the port subfile. Table 28–1 describes the way a READ operation is handled in each file state supported by NETBIOSSESSIONSERVICE.

Table 28–1. Effects of File State on the READ Operation for NETBIOSSESSIONSERVICE

File State	Action
CLOSED	Implicit OPEN
OFFERED	EOF
AWAITINGOFFER	EOF
OPENED	If the value of CENSUS is greater than 0, returns input data; otherwise depends on WAIT/DONTWAIT (NO WAIT in COBOL74)
SHUTTINGDOWN	If the value of CENSUS is greater than 0, returns input data; otherwise depends on WAIT/DONTWAIT (NO WAIT in COBOL74)
CLOSEPENDING	EOF
DEACTIVATIONPENDING	Returns input data; when READ queue is empty, port subfile moves to DEACTIVATED
DEACTIVATED	EOF (The STATE attribute or READ result stores a value of 46 (ENDOFFILE)in field [26:10], and the end-of-file bit [9:1] and exception [0:1] bit are each set to 1.)

The way a WRITE operation is handled by the port service depends on the file state of the port subfile. Table 28–2 describes the way a WRITE operation is handled in each file state supported by NETBIOSSESSIONSERVICE.

Table 28–2. Effects of File State on the WRITE Operation for NETBIOSSESSIONSERVICE

File State	Action
CLOSED	Implicit OPEN
OFFERED	EOF
AWAITINGOFFER	EOF
OPENED	If OUTPUTEVENT is in the happened state, then WRITE (OK); otherwise depends on WAIT/DONTWAIT (NO WAIT in COBOL74)
SHUTTINGDOWN	If OUTPUTEVENT is in the happened state, then WRITE (OK); otherwise depends on WAIT/DONTWAIT (NO WAIT in COBOL74)
CLOSEPENDING	EOF
DEACTIVATIONPENDING	EOF
DEACTIVATED	EOF (The STATE attribute or WRITE result stores a value of 46 (ENDOFFILE) in field [26:10],and the end-of-file bit [9:1] and exception [0:1] bit are each set to 1.)

Closing a Dialogue Using NETBIOSSESSIONSERVICE

NETBIOSSESSIONSERVICE does not support orderly dialogue release. NETBIOSSESSIONSERVICE provides ABORT termination through the CLOSE ABORT statement, as described in Section 20, "Closing a Dialogue."

Example Applications Using NETBIOSSESSIONSERVICE

Refer to the system software release media (tape or CD-ROM) for sample port file applications using NETBIOSSESSIONSERVICE. Corresponding PC applications using the NetBIOS interface are also included on system software release diskettes.

Section 29
Understanding Virtual Files

The virtual file feature provides a mechanism through which a program in an MCP environment can access features provided outside the MCP by using native language constructs such as OPEN, CLOSE, READ, and WRITE. The MCP provides integrated generic support for linking to libraries by implementing various program requested file operations. This library is referred to as the IOHANDLER for the virtual file.

The need for a virtual file becomes most apparent when an application needs to access a resource using traditional I/O intrinsics, like READ and WRITE, but the I/O subsystem does not support access to the resource. The developer can build a library that contains the implementation for accessing the resource, but the application needs the virtual file mechanism to gain access to this library through the I/O intrinsics. The virtual file mechanism provides the MCP support that enables the application to access and use the implementation residing transparently in the developer's external library.

Instead of declaring the library and entry points in the application, the IOHANDLER library is accessed by opening a logical file with the appropriate attributes specified. Once the open routine is performed successfully, the application utilizes the library through normal READ, WRITE, and attribute modification statements.

Because the virtual file feature allows access to user-developed IOHANDLER libraries, new capabilities can be provided without modification of the MCP. Examples of these capabilities include

- Handling a new network protocol
- Accessing database management systems
- Accessing user transformation of data

This section describes how to use a virtual file in an application program, and provides details about the IOHANDLER library and examples. Section 30, "Using the REDIRSUPPORT IOHANDLER Library," describes how to use virtual files with the Unisys provided Redirector IOHANDLER library to access files on shares available to network users.

Using Virtual Files

From the programmer's point of view, virtual files appear to be just another type of file. The file attributes can be set and the files can be opened, read, written to, and closed. However, instead of communicating with an I/O device, the program communicates with a user-developed or a Unisys provided IOHANDLER library. Because the IOHANDLER implements the semantics of a virtual file, this section provides only a general overview of virtual file concepts. For details, refer to "Understanding the IOHANDLER" later in this section.

Programming for Virtual Files

Virtual files are generally used in a program as described in Section 2, "Understanding Programming for Files," except that the virtual file feature does not support DIRECT I/O. The following concepts are specific to using virtual files.

While virtual files are handled by the IOHANDLER library rather than the I/O subsystem, there are well-defined semantics for programs using virtual files. To describe these semantics, it is useful to think of the IOHANDLER as the "physical" file that corresponds to the "logical" virtual file declared in the program. The IOHANDLER uses attributes specified on the logical file to locate or create the physical file, and returns attributes for use by the MCP and program by processing operations on the logical file.

For example, when the "logical" virtual file specifies the attributes NEWFILE=FALSE and DEPENDENTSPECS=TRUE, the open routine for the IOHANDLER library returns values corresponding to the physical file. These attributes are then set on the virtual file in a manner similar to other file implementations.

Virtual File Library Attributes

Setting the KIND attribute of a file to VIRTUAL enables the IOHANDLER library to provide the functionality for the OPEN, CLOSE, READ, and WRITE intrinsics. It is necessary for the MCP to be able to locate the IOHANDLER and its entry points to service the file operation.

Table 29–1 lists the attributes that are used in the virtual file declaration to specify the location and characteristics of the IOHANDLER library. Except for IOHSTRING and IOHPREFIX, these attributes specify the form of library linkage to be used when linking to the IOHANDLER library as part of opening a virtual file. These attributes are consistent with their LIBRARY attribute counterparts and are defined to provide a comprehensive set of library functionality. For details about each attribute, refer to the *File Attributes Reference Manual*.

Table 29–1. IOHANDLER Library Attributes

Attribute	Description
IOHLIBACCESS	Specifies the form of library access, such as BYTITLE, BYFUNCTION, and BYINITIATOR. Consistent with the LIBACCESS library attribute.
IOHFUNCTIONNAME	Specifies the library function name when library access is set to BYFUNCTION. Consistent with the FUNCTIONNAME library attribute.
IOHTITLE	Specifies the library title when library access is set to BYTITLE. Consistent with the TITLE library attribute.
IOHINTERFACENAME	Corresponds to the INTERFACENAME library attribute.
IOHLIBPARAMETER	Corresponds to the LIBPARAMETER library attribute.
IOHPREFIX	Specifies the prefix string to be used when specifying the actual name of the entry points before linking to the IOHANDLER library.
IOHSTRING	Provides an Inter-Program Communication (IPC) mechanism between the application using the virtual file and the IOHANDLER library. This mechanism can be useful for propagating information in and out of the library.

Examples

The following file declarations show how the IOHANDLER library attributes can be used in a program to specify a virtual file:

```
FILE VF1 (KIND=VIRTUAL, IOHLIBACCESS = BYTITLE,
          IOHTITLE = "OBJECT/MYIOHANDLER.",...);

FILE VF2 (KIND=VIRTUAL, IOHLIBACCESS = BYFUNCTION,
          IOHFUNCTIONNAME = "MYIOHSUPPORT.",...);
```

The link to the IOHANDLER library is accomplished by transferring the library attributes declared on the virtual file to the internal library ("IOHANDLER") declared for the file. For example, the IOHFUNCTIONNAME attribute becomes the FUNCTIONNAME attribute name of the internal library. The link is the same as if the program declaration included a LIBRARY IOHANDLER and used some or all default values for the library's attributes.

The only exception in this linking scheme is the specification of LIBACCESS to BYFUNCTION instead of BYTITLE for the internal library defaults when the IOHFUNCTIONNAME attribute has been specified instead of IOHTITLE. This action simplifies setting the virtual file library attributes by making it unnecessary to set IOHLIBACCESS to BYFUNCTION when the IOHFUNCTIONNAME is specified.

Also, when the IOHPREFIX attribute is unspecified, the entry points are assumed to have a prefix of IOH_. That is, the entry points are named IOH_OPEN, IOH_CLOSE, and so forth. For example, consider the following file declarations:

```
FILE VF3 (KIND=VIRTUAL);

FILE VF4 (KIND=VIRTUAL, IOHLIBACCESS=BYFUNCTION);

FILE VF5 (KIND=VIRTUAL, IOHPREFIX = "MYDISK.");
```

For file VF3, library linkage is performed using a library access value of BYTITLE and a code file title of IOHANDLER. For file VF4, library linkage is performed using a library access value of BYFUNCTION and a function name of IOHANDLER. In either case, the entry point names are the default names described previously. For file VF5, library linkage is the same as that of file VF3 except that the library entry points are named MYDISKOPEN, MYDISKCLOSE, MYDISKREAD, and MYDISKWRITE.

Virtual File Structure Considerations

Virtual files support only a subset of file format attributes as summarized in Table 29–2. Attributes that are not specified in the table are either ignored as being irrelevant to virtual files, such as AREASIZE, or all values are supported, such as FRAMESIZE. Values specified in the virtual file for the following attributes are passed to the IOHANDLER library's open procedure. A discussion of these attributes is presented later in this section.

Table 29–2. Virtual File Format Attributes

Attribute	Value
FILEORGANIZATION	The only supported value is NOTRESTRICTED.
FILESTRUCTURE	The value of FILESTRUCTURE is ignored when set on a virtual file.
BLOCKSTRUCTURE	The valid values are FIXED and EXTERNAL. When specified on a virtual file, the value returned by the IOHANDLER library open procedure must match this value.
FILETYPE	Only values of 0 and 3 are supported, corresponding to BLOCKSTRUCTURE values of FIXED and EXTERNAL, respectively.
MINRECSIZE	A MINRECSIZE value set on a virtual file is ignored. Set the value to MAXRECSIZE for BLOCKSTRUCTURE=FIXED and to 0 for BLOCKSTRUCTURE=EXTERNAL.
MAXRECSIZE	The IOHANDLER library's open procedure returns its MAXRECSIZE value in IOHMODE characters. If DEPENDENTSPECS is TRUE for the virtual file, the MAXRECSIZE value is adjusted to match the IOHANDLER value, depending on the INTMODE value of the virtual file.
EXTMODE	The IOHANDLER returns a value of IOHMODE that corresponds to the mode that the library is operating in for this virtual file. The IOHMODE attribute of the IOHANDLER is compared with the EXTMODE attribute of the virtual file for compatibility during the open procedure, adjusting for any OVERRIDEEXTMODE specification.

The interface between the MCP Logical I/O module and the IOHANDLER library dictates some of the restrictions in Table 29–2. For details, refer to "Understanding the IOHANDLER" later in this section.

Explanation

The IOHANDLER library's READ and WRITE routines are random record-oriented interfaces. That is, the IOHANDLER library's READ and WRITE entry points always expect a record offset and a length. The MCP, on behalf of the program, converts any serial access into an appropriate random access based upon the current record pointer. Because there is no blocking of records, the FILESTRUCTURE value that corresponds to the interface is inherently STREAM. Since the I/O subsystem is not handling the physical file, the FILESTRUCTURE value specified on the logical virtual file can be ignored.

The IOHANDLER library is interfaced from the MCP. The interface is based on EXTMODE character units , and the IOHRECSIZE value is used as a parameter. The IOHRECSIZE value returned from the IOHANDLER library's open routine is expressed as a number of EXTMODE characters, and the length parameter to the READ and WRITE entry points always refers to the number of characters to be transferred. The MCP, on behalf of the program, also converts the length of the request into characters if required.

For example, if the FRAMESIZE attribute value of the logical file is 48 (words) and the EXTMODE attribute is set to EBCDIC, the length passed into the IOHANDLER is adjusted by a multiple of 6. Similarly, the result returned by the IOHANDLER after a successful READ or WRITE procedure is converted back into words when expressing the length in the result descriptor. The MCP maintains the file state of the logical virtual file consistent with its specified MAXRECSIZE value after each I/O operation.

As part of the OPEN process for a file, the MCP ensures that the IOHANDLER library's interpretation of logical record size is the same as the program's interpretation of logical record size, based on the following expression:

```
Virtual File:  MAXRECSIZE * BITS PER EXTFRAME  =

IOHANDLER:     IOHRECSIZE * BITS PER IOHMODE CHARACTER
```

A BLOCKSTRUCTURE value of either FIXED or EXTERNAL is supported and it is the IOHANDLER library's responsibility to return a proper BLOCKSTRUCTURE value as part of the open routine. If the IOHANDLER returns any value other than FIXED or EXTERNAL, or if there is a mismatch between the logical and virtual files, the result is an open error.

The differences between the BLOCKSTRUCTURE FIXED and EXTERNAL values for virtual files are

- For BLOCKSTRUCTURE = FIXED, all requests to the IOHANDLER library are rounded up to the nearest record boundary.
  - For a read request, the length passed to the IOHANDLER is rounded up to the nearest record boundary. Only the amount actually requested by the program is transferred into the user's buffer on return from the IOHANDLER library's READ routine.
  - For a write request, the request is padded with zeroes to the end of the record and the request length is adjusted to the record boundary before calling the IOHANDLER library's WRITE routine. The length is returned in the logical result descriptor for the request and stored in the logical file STATE attribute. The length is always adjusted to a record boundary.
- For BLOCKSTRUCTURE = EXTERNAL, requests are not rounded up to the next record and the READ or WRITE request made to the IOHANDLER matches the length specified by the program. The length returned in the logical result descriptor (STATE) is not adjusted to a record boundary and matches the actual requested length. However, the record pointer in the file information block (FIB) is always adjusted to the next record consistent with the MAXRECSIZE value. Note that, unlike DISK files, the MINRECSIZE value is always set to 0 when BLOCKSTRUCTURE is set to EXTERNAL.

The ANYSIZEIO attribute is supported for virtual files when the value for BLOCKSTRUCTURE is FIXED, enabling multiple records to be transferred in a single request. When BLOCKSTRUCTURE is set to EXTERNAL, setting ANYSIZEIO to TRUE results in an error in the open routine unless ADAPTABLE is also TRUE. As with other files, when ANYSIZEIO is FALSE, read or write requests that exceed the MAXRECSIZE value are truncated to fit. The partial portion of any request that is not a multiple of the MAXRECSIZE value is treated in the same manner for each BLOCKSTRUCTURE value.

Virtual File Attributes

File attributes are set and interrogated on virtual files in the same manner as for any other file. Before the file is opened, the attributes are applied and returned from the unopened file structure. After the file is opened, the file attributes supported for virtual files are handled by calling the IOHANDLER library. The IOHANDLER supports the following attributes for virtual files. For details about each attribute, refer to the *File Attributes Reference Manual*.

ACCESSDATE	EXTDELIMITER	OWNER
ACCESSTIME	FILEKIND	SECURITYGUARD
ALTERDATE	FILENAME (and related LFILENAME, TITLE, LTITLE and PATHNAME)	SECURITYMODE (and all related sub attributes)
ALTERNATEGROUPS	FILELENGTH	SECURITYTYPE
ALTERTIME	GROUP	SECURITYUSE
CCSVERSION	IOHSTRING	SYNCHRONIZE
CREATIONDATE	LASTRECORD	USERINFO
CREATIONTIME	NOTE	

Virtual File Translation

Translation can be provided by Logical I/O when the INTMODE value of a file differs from the EXTMODE value of that file. Refer to "Dealing with Translation" in Section 2, "Understanding Programming for Files." Translation is also provided for virtual files in this manner, but requires coordination with the IOHANDLER library. The IOHANDLER library is interfaced from the MCP and the IOHMODE value is used as a parameter in that interface. During the opening of a file, the IOHANDLER open routine returns the IOHMODE value corresponding to the EXTMODE value of a physical file. Because the IOHANDLER adds a third value to the mix, such as virtual file INTMODE, EXTMODE, or library IOHMODE, there is some additional complexity.

The IOHANDLER open routine returns the IOHMODE value that represents the character encoding and the FRAMESIZE value that was used to interpret the IOHRECSIZE value, which is also returned at open time. Normally, when an existing physical file is opened, the EXTMODE value of the logical file unconditionally assumes the EXTMODE value of the physical file. This is the same for virtual files, such that the EXTMODE value of the logical virtual file is set to the IOHMODE value returned from the library. Also, the EXTMODE value of the logical file is normally used when creating a new file. For virtual files, however, the library might return an IOHMODE value that differs from the EXTMODE value specified on the logical file. Due to this possibility, the EXTMODE value of the virtual file is set to the IOHMODE value that is returned from the library, even when new file creation is requested.

Logical I/O also supports the OVERRIDEEXTMODE file attribute to permit the EXTMODE value of the logical file to differ from the EXTMODE value of the physical file, perhaps to avoid or to cause translation. OVERRIDEEXTMODE is also supported for virtual files with the restriction that the character size of the IOHMODE value must match the character size of the EXTMODE value of the virtual file. This restriction is consistent with the native MCP implementation.

Once the EXTMODE value of the virtual file is determined, the value is compared for compatibility with the INTMODE value specified on the virtual file. This comparison is described in Table 2–14, "Possible EXTMODE and INTMODE Combinations." Note that the DEPENDENTINTMODE attribute is also supported for virtual files. When the attribute is set to TRUE, the INTMODE value is set to the EXTMODE value of the logical file. By using DEPENDENTINTMODE, translation is not done regardless of the EXTMODE value specified or the IOHMODE value returned from the library.

Opening a Virtual File

The virtual file open routine supports all the various open mechanisms, such as OPEN(F), F.AVAILABLE, and implicit open on read or write. For details about the results of opening a file, refer to the *File Attributes Reference Manual*.

Opening a virtual file initiates the first link with the desired IOHANDLER library. Validation of a successful open routine consists of

- MCP checks before calling the IOHANDLER library
- Checks made by the IOHANDLER
- Verification of attribute compatibility with the virtual file after values are returned by the IOHANDLER

The checks made by the MCP before calling the IOHANDLER involve verifying the values of the attributes described in "Virtual File Structure Considerations" earlier in this section. Other checks that occur for all file types also are performed during the open routine.

Once these checks are made, the MCP attempts to link to the IOHANDLER based on the virtual file library attributes. A structure containing a subset of file attributes either set explicitly or by default is passed to the IOHANDLER library's OPEN entry point. These file attributes are documented in "Understanding the IOHANDLER" later in this section. Because the IOHANDLER library implements the bulk of the virtual file, the library's use of file attributes can vary from other file kinds managed by the MCP. The requirements of each IOHANDLER library should be documented by the writer of the library.

After a successful return from the OPEN procedure, compatibility with the virtual file structure is performed. The OPEN result is returned to the application.

Virtual File I/O Requests

Virtual file I/O requests are identical to those used for accessing DISK files with a FILESTRUCTURE value of STREAM. For details about the result values returned for I/O requests, refer to the *File Attributes Reference Manual*.

ANYSIZEIO access is supported for BLOCKSTRUCTURE = FIXED. UPDATEFILE access is supported, but as with DISK files, ANYSIZEIO must be FALSE.

The SYNCHRONIZE directive is valid for the WRITE intrinsic, but it is up to the IOHANDLER library to interpret the request and act accordingly.

BUFFERSHARING is not supported for virtual files. Any BUFFERSHARING attribute is passed to the IOHANDLER open routine, but does not cause the MCP to reject the open request. The MCP allocates a single buffer for the file to process the I/O request; however, the BUFFERS value is also passed to the IOHANDLER library.

ALGOL Example

The following program demonstrates the use of KIND = VIRTUAL in an ALGOL program and utilizes the IOHANDLER library defined by the IOHFUNCTIONNAME attribute as "DISKIOHANDLER."

The program reads an existing ASCII stream file, MYMACS.TXT, which has logical records delimited by carriage-return line feed (CRLF) characters. The data is read from the file VF, translated from ASCII to EBCDIC, and written to a new EBCDIC stream file VFNEW. Each logical record as delimited by a CRLF is displayed on the remote file RMT. The IOHLIBACCESS and IOHPREFIX attributes are set to default values. The VFNEW file, although not specifying IOHLIBACCESS or IOHPREFIX, is linked by the IOHANDLER library in an identical fashion.

```
BEGIN
FILE RMT(KIND=REMOTE,
         UNITS=CHARACTERS,
         MYUSE=IO);

FILE VF (KIND=VIRTUAL
        ,FILENAME=""MYMACS.TXT""."
        ,NEWFILE=FALSE
        ,DEPENDENTSPECS
        ,EXTMODE=ASCII
        ,INTMODE=EBCDIC
        ,FILEUSE=IN
        ,UNITS=CHARACTERS
        ,ANYSIZEIO
        ,IOHPREFIX="IOH_."
        ,IOHLIBACCESS=BYFUNCTION
        ,IOHFUNCTIONNAME="DISKIOHANDLER."
       );

FILE VFNEW(KIND=VIRTUAL
```

```
        ,IOHFUNCTIONNAME="DISKIOHANDLER."
        ,NEWFILE=TRUE
        ,FILEUSE=OUT
        ,FILESTRUCTURE=STREAM
        ,FRAMESIZE=8
        ,MAXRECSIZE=1
        ,ANYSIZEIO
       );

EBCDIC ARRAY EA, EB [0:2999];
POINTER P, PA;
BOOLEAN BRD;
REAL    I, J, R, RD=BRD;
DEFINE CR=48"0D"#,
       LF=48"25"#;
TRUTHSET CRLF (CR OR LF);

OPEN (VF);
OPEN (VFNEW);
WHILE NOT BRD DO
BEGIN
    BRD := READ (VF, 3000, EA);
    I := RD.[47:20];
    IF NOT BRD THEN
    BEGIN
        WRITE (VFNEW [J], I, EA);
        J := * + I;
        PA := EA[0];
        DO BEGIN
           REPLACE P:EB[0] BY PA:PA FOR I:I UNTIL IN CRLF;
           WRITE (RMT, OFFSET(P), EB);
           SCAN PA:PA FOR I:I WHILE IN CRLF;
           END
        UNTIL I LEQ 0;
    END;
END;
CLOSE (VFNEW, LOCK);
CLOSE (VF);
END.
```

COBOL85 Example

The following COBOL85 program performs the equivalent function of the ALGOL example that precedes it.

```
 IDENTIFICATION DIVISION.
*
 ENVIRONMENT DIVISION.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
* The ASCII stream file we want to read.
  SELECT VF    ASSIGN TO VIRTUAL.
* The new EBCDIC stream file we want to write VF.
  SELECT VFNEW ASSIGN TO VIRTUAL.
* The REMOTE by which we view VF's content.
  SELECT REM   ASSIGN TO REMOTE.
*
 DATA DIVISION.
 FILE SECTION.
 FD VF
      VALUE OF
               FILENAME      IS """MYMACS.TXT"""
               NEWFILE       IS FALSE
               DEPENDENTSPECS IS TRUE
               EXTMODE       IS ASCII
               INTMODE       IS EBCDIC
               ANYSIZEIO     IS TRUE
               IOHPREFIX     IS "IOH_."
               IOHLIBACCESS  IS BYFUNCTION
               IOHFUNCTIONNAME IS "DISKIOHANDLER.".
 01 VF-REC     PIC X(3000).


 FD VFNEW
      RECORD CONTAINS 1 TO 3000 CHARACTERS
      VALUE OF
               IOHFUNCTIONNAME IS "DISKIOHANDLER."
               NEWFILE       IS TRUE
               FILESTRUCTURE IS STREAM
               FRAMESIZE     IS 8
               MAXRECSIZE    IS 1
               BLOCKSTRUCTURE IS FIXED
               ANYSIZEIO     IS TRUE.
 01 VFNEW-REC.
    03 REC-GUTS PIC X OCCURS 1 TO 3000 DEPENDING MSG-LEN.


 FD REM.
 01 REM-REC    PIC X(80).
*
 WORKING-STORAGE SECTION.
 01 REC-ARRAY    PIC X(3000).
```

```
 01 REM-ARRAY   PIC X(80).
 77 UNSTRING-NEXT  REAL.
 77 SAVED-START    REAL.
 77 UNSTRING-CNT   REAL.
 77 MSG-LEN        REAL.
*
 PROCEDURE DIVISION.
  MAIN-PARA.
    OPEN INPUT VF.
    OPEN OUTPUT VFNEW.
    OPEN I-O REM.
    PERFORM PROCESS-LOOP THRU PROCESS-LOOP-EXIT.
    CLOSE REM.
    CLOSE VFNEW SAVE.
    CLOSE VF.
    STOP RUN.


  PROCESS-LOOP.
    READ VF
             AT END GO TO PROCESS-LOOP-EXIT.
    MOVE ATTRIBUTE CURRENTRECORDLENGTH OF VF TO MSG-LEN.
*   Write a variable amount, up to 3000 characters.
    MOVE VF-REC TO VFNEW-REC
    WRITE VFNEW-REC
    PERFORM REMOTE-DISPLAY
    GO TO PROCESS-LOOP.


PROCESS-LOOP-EXIT.
  EXIT.


REMOTE-DISPLAY.
  MOVE VF-REC TO REC-ARRAY.
  MOVE 1 TO UNSTRING-NEXT.
  PERFORM WRITE-RECS-TO-REMOTE THRU
          WRITE-RECS-TO-REMOTE-EXIT UNTIL MSG-LEN <= 0.


WRITE-RECS-TO-REMOTE.
  MOVE 0 TO UNSTRING-CNT.
  MOVE UNSTRING-NEXT TO SAVED-START.
  MOVE SPACES TO REM-ARRAY.


  UNSTRING REC-ARRAY
           DELIMITED BY ALL @0D25@
           INTO REM-ARRAY
           COUNT IN UNSTRING-CNT
           WITH POINTER UNSTRING-NEXT.
```

```
*   UNSTRING-CNT is the number of characters moved to REM-ARRAY.
*   UNSTRING-NEXT is the next position in source after UNSTRING,
*   including the delimiter(s).
    WRITE REM-REC FROM REM-ARRAY.


    SUBTRACT SAVED-START FROM UNSTRING-NEXT GIVING UNSTRING-CNT.
    SUBTRACT UNSTRING-CNT FROM MSG-LEN.


  WRITE-RECS-TO-REMOTE-EXIT.
    EXIT.
```

Virtual File IOHANDLER

The functionality of a virtual file is provided by a library called an IOHANDLER that is linked to the user program's file. Logical I/O provides generic support to enable a virtual file declared in a program access to a particular IOHANDLER library. When a virtual file is opened, a link to the appropriate IOHANDLER library is established and control is passed to the IOHANDLER library's OPEN entry point. Similarly, when a READ or WRITE procedure is performed by the application, control is passed from Logical I/O to the IOHANDLER library's READ or WRITE entry points. When the virtual file is closed, the CLOSE entry point is invoked and the file is de-linked from the IOHANDLER library.

The following paragraphs describe how to create an IOHANDLER library.

Understanding the IOHANDLER

The IOHANDLER library is expected to export entry points that adhere to a well-defined protocol consistent with normal device handling. Each entry point is called from Logical I/O as part of processing a native language I/O statement on the program-declared virtual file, such as a read request or an attribute modification request.

The invocation of each IOHANDLER entry point by the virtual file results in an important side effect. Logical I/O ensures the integrity of a file that is visible to more than one process, such as a program with offspring, by securing a lock in the file description before acting on the file. Because Logical I/O provides this insurance, the IOHANDLER programmer does not need to lock the structures associated with a particular virtual file. This concept, along with common IOHANDLER entry point parameters, is intended to simplify the creation of an IOHANDLER library that requires only a single instantiation, as in a SHAREDBYALL library.

The IOHANDLER entry points and the functions they are called for are described in Table 29–3. The definitions and descriptions of each of these entry points are provided later in this section.

Table 29–3. IOHANDLER Entry Points

Entry Point	Language Function	Required
IOH_OPEN	OPEN, PRESENT, AVAILABLE, and RESIDENT functions, and implicit open	Yes
IOH_CLOSE	Explicit CLOSE and block exit close	Yes
IOH_READ	Serial or random READ request	No
IOH_WRITE	Serial or random WRITE request	No
IOH_GETATTRIBUTE	Interrogation of supported attributes	No
IOH_SETATTRIBUTE	Setting supported attributes	No
IOH_ERASEFILE	ALGOL ERASE and MCP ERASEFILE functions	No
IOH_FSYNC	POSIX FSYNC and MCP FILESYNC functions	No

A link to the IOHANDLER library occurs when an attempt is made to open a virtual file. Not all IOHANDLER entry points are required during linkage to the library on behalf of a virtual file. However, if all of the required entry points are not exported by the IOHANDLER, the linkage from the virtual file to the specified IOHANDLER library fails and an open error on the virtual file is returned to the program. Alternatively, if an optional entry point is not provided by a particular IOHANDLER library, an error result is produced only if an action on the virtual file requires the optional entry point. The one exception to this result is for the IOH_FSYNC attribute, which returns a good result.

Common IOHANDLER Entry Point Parameters

Each entry point exported by an IOHANDLER library has two common parameters, declared as the first two entries in the parameter list. The following paragraphs describe the declaration of each of the IOHANDLER entry points and their parameters.

The IOHANDLER parameters are a file and a two-dimensional array declared in Logical I/O. These parameters are referred to as F and IOH_DATA, respectively. While Logical I/O has visibility to each of these parameters, they are provided solely for the use of the library and are never referenced by Logical I/O. As such, they can be used by the library if necessary. In particular, since these parameters are passed to all IOHANDLER entry points, they can be used to hold state information about a particular virtual file without storing global state information in the library. This handling of the F and IOH_DATA parameters, coupled with the file locking as described in "Understanding the IOHANDLER" earlier in this section, is intended to make it easy for an IOHANDLER library to be written as a stateless, shared library. Such a library is included in the example IOHANDLER library later in this section.

The states of the two parameters on the initial library linkage are as follows:

- The IOH_DATA parameter is a non-present, touched multi-dimensioned array declared with 4 rows. The array rows can be resized with minimal overhead according to the needs of the IOHANDLER library.
- The file parameter F is an unassigned file descriptor with default file attributes. File equation from the program declaring the virtual file is not applied when the file is initialized.

IOH_OPEN

The required IOH_OPEN entry point is invoked when the virtual file is opened. It is expected to return an open result along with values for the by-reference parameters. This entry point must be exported to be compatible with the following procedure template:

```
REAL PROCEDURE IOH_OPEN (F, IOH_DATA, IOH_INFO, IOHRECSIZE,
     IOHMODE, IOHMAXXMTRECS, IOHACCESS,
     IOHBLOCKSTRUCTURE);
     FILE F;
     ARRAY IOH_DATA [0,0], IOH_INFO [0];
     REAL  IOHRECSIZE, IOHMODE, IOHMAXXMTRECS,
        IOHACCESS, IOHBLOCKSTRUCTURE;
```

Results

The result returned as the procedure value is defined in the *File Attributes Reference Manual*. These values are supported by the compilers, so that the mnemonic values can be used. For example, the following statements return the appropriate values of a good result and a no file condition, respectively:

```
IOH_OPEN := VALUE(OKRSLT);
IOH_OPEN := VALUE(NOFILEFOUNDRSLT);
```

Parameters

IOHRECSIZE

The value returned by the OPEN entry point is MAXRECSIZE in units of IOHMODE characters of the virtual file.

IOHMODE

The value returned by the OPEN entry point is the character encoding and size expected in the READ and WRITE entry points; that is, the length of the data and the pointer passed to those entry points are sized according to the IOHMODE value returned.

IOHMAXXMTRECS

The value returned by the OPEN entry point is the maximum number of records that can be transferred in and out of the library on a single call to its read or write routine.

IOHACCESS

The value returned by the OPEN entry point is the number of access permissions that have been granted. This parameter has the following layout:

```
SEC_RF  =  [2:1] #  % Read permitted
SEC_WF  =  [1:1] #  % Write permitted
SEC_XF  =  [0:1] #  % Execute/search permitted
                    % (Not currently used but defined
                    % for completeness.)
```

IOHBLOCKSTRUCTURE

This parameter, when returned by the OPEN entry point, contains the BLOCKSTRUCTURE value of the virtual file. The BLOCKSTRUCTURE attribute must be set to either VALUE (FIXED) or VALUE (EXTERNAL). For details about BLOCKSTRUCTURE semantics, refer to "Virtual File Structure Considerations" earlier in this section.

IOH_INFO

This array, when passed into the OPEN entry point, contains information from the logical virtual file as defined in the example that follows. Each of the defines corresponds to the file attribute value that it carries. For instance, the INFO_ANYSIZEIO define contains the ANYSIZEIO attribute setting in the virtual file.

Pointer-valued attributes are passed using an index and length define. For example, INFO_FILENAMEINX and INFO_FILENAMECHRS are the index defines in the IOH_INFO attribute and the length defines in the virtual file FILENAME attribute. Note that pointer attributes are terminated with a period, which is included in the length defines.

```
DEFINE
   INFO_ITEMP           (INFO,X) = POINTER(INFO[X]) #
  ,INFO_WORDS           (INFO)   = INFO  [0].[47:24] #
  ,INFO_FIXED_WORDS     (INFO)   = INFO  [0].[23:24] #
  ,INFO_VERSION         (INFO)   = INFO  [1].[07:08] #
     ,INFO_VERSIONV              = 1 #
  ,INFO_OPENATTRIBUTE   (INFO)   = INFO  [2].[36:01] #
  ,INFO_OPENIMPLICIT    (INFO)   = INFO  [2].[32:01] #
  ,INFO_OPENTYPE        (INFO)   = INFO  [2].[31:08] #
  ,INFO_NEWFILE         (INFO)   = INFO  [2].[23:01] #
  ,INFO_EXCLUSIVE       (INFO)   = INFO  [2].[22:01] #
  ,INFO_FILEUSE         (INFO)   = INFO  [2].[21:02] #
  ,INFO_FLEXIBLE        (INFO)   = INFO  [2].[19:01] #
  ,INFO_NORESOURCEWAIT  (INFO)   = INFO  [2].[18:01] #
  ,INFO_SENSITIVEDATA   (INFO)   = INFO  [2].[17:01] #
  ,INFO_DEPENDENTINTMODE(INFO)   = INFO  [2].[16:01] #
  ,INFO_DEPENDENTSPECS  (INFO)   = INFO  [2].[12:01] #
  ,INFO_PROTECTION      (INFO)   = INFO  [2].[11:02] #
```

```
  ,INFO_SYNCHRONIZE     (INFO)   = INFO  [2].[09:02] #
  ,INFO_FILEKIND        (INFO)   = INFO  [2].[07:08] #
  ,INFO_FILEORGANIZATION(INFO)   = INFO  [3].[47:04] #
  ,INFO_FILESTRUCTURE   (INFO)   = INFO  [3].[43:08] #
  ,INFO_BLOCKSTRUCTURE  (INFO)   = INFO  [3].[35:04] #
  ,INFO_TRANSLATE       (INFO)   = INFO  [3].[31:04] #
  ,INFO_EXTMODEBITS     (INFO)   = INFO  [3].[27:08] #
  ,INFO_OVERRIDEEXTMODE (INFO)   = INFO  [3].[19:04] #
  ,INFO_EXTMODE         (INFO)   = INFO  [3].[15:16] #
  ,INFO_INTMODE         (INFO)   = INFO  [4].[47:16] #
  ,INFO_CCSVERSION      (INFO)   = INFO  [4].[31:16] #
  ,INFO_EXTDELIMITER    (INFO)   = INFO  [4].[15:08] #
  ,INFO_FRAMESIZE       (INFO)   = INFO  [4].[07:08] #
  ,INFO_BLOCKSIZE       (INFO)   = INFO  [5].[47:16] #
  ,INFO_MINRECSIZE      (INFO)   = INFO  [5].[31:16] #
  ,INFO_MAXRECSIZE      (INFO)   = INFO  [5].[15:16] #
  ,INFO_ANYSIZEIO       (INFO)   = INFO  [6].[44:01] #
  ,INFO_ADAPTABLE       (INFO)   = INFO  [6].[40:01] #
  ,INFO_UNITS           (INFO)   = INFO  [6].[36:01] #
  ,INFO_BUFFERSHARING   (INFO)   = INFO  [6].[33:02] #
  ,INFO_BUFFERS         (INFO)   = INFO  [6].[31:08] #
  ,INFO_BUFFERSIZE      (INFO)   = INFO  [6].[23:24] #
  ,INFO_AREAS           (INFO)   = INFO  [7].[47:16] #
  ,INFO_AREALENGTH      (INFO)   = INFO  [7].[31:32] #
  ,INFO_AREASIZE        (INFO)   = INFO  [8].[47:24] #
  ,INFO_SECURITYTYPE    (INFO)   = INFO  [8].[23:04] #
  ,INFO_SECURITYUSE     (INFO)   = INFO  [8].[19:04] #
  ,INFO_SECURITYMODE    (INFO)   = INFO  [8].[15:16] #
  ,INFO_FILESNR         (INFO)   = INFO  [9] #
  ,INFO_FILEMIX         (INFO)   = INFO [10] #

% Variable Length Attributes
  ,INFO_ATT_LINKF   =  [47:24] #
  ,INFO_ATT_LENF    =  [23:24] #
  ,INFO_FILENAMEINX     (INFO)   = INFO [11].INFO_ATT_LINKF #
  ,INFO_FILENAMECHRS    (INFO)   = INFO [11].INFO_ATT_LENF #
  ,INFO_FAMILYNAMEINX   (INFO)   = INFO [12].INFO_ATT_LINKF #
  ,INFO_FAMILYNAMECHRS  (INFO)   = INFO [12].INFO_ATT_LENF #
  ,INFO_STRINGINX       (INFO)   = INFO [13].INFO_ATT_LINKF #
  ,INFO_STRINGCHRS      (INFO)   = INFO [13].INFO_ATT_LENF #
  ,INFO_FILEUCINX       (INFO)   = INFO [14].INFO_ATT_LINKF #
  ,INFO_FILEUCCHRS      (INFO)   = INFO [14].INFO_ATT_LENF #
  ,INFO_NOTEINX         (INFO)   = INFO [15].INFO_ATT_LINKF #
  ,INFO_NOTECHRS        (INFO)   = INFO [15].INFO_ATT_LENF #
  ,INFO_GUARDFILEINX    (INFO)   = INFO [16].INFO_ATT_LINKF #
  ,INFO_GUARDFILECHRS   (INFO)   = INFO [16].INFO_ATT_LENF #
  ,INFO_GROUPINX        (INFO)   = INFO [17].INFO_ATT_LINKF #
  ,INFO_GROUPCHRS       (INFO)   = INFO [17].INFO_ATT_LENF #
  ,INFO_ALTGROUPSINX    (INFO)   = INFO [18].INFO_ATT_LINKF #
  ,INFO_ALTGROUPSCHRS   (INFO)   = INFO [18].INFO_ATT_LENF #
  ,INFO_OWNERINX        (INFO)   = INFO [19].INFO_ATT_LINKF #
  ,INFO_OWNERCHRS       (INFO)   = INFO [19].INFO_ATT_LENF #
```

```
,INFO_INTNAMEINX      (INFO)   = INFO [20].INFO_ATT_LINKF #
,INFO_INTNAMECHRS     (INFO)   = INFO [20].INFO_ATT_LENF #
,INFO_HOSTNAMEINX     (INFO)   = INFO [21].INFO_ATT_LINKF #
,INFO_HOSTNAMECHRS    (INFO)   = INFO [21].INFO_ATT_LENF #
,INFO_FIRSTINFO                = 22 #
```

IOH_CLOSE

The required IOH_CLOSE entry point is invoked when the virtual file is closed, either explicitly or due to block exit action. This entry point must be exported to be compatible with the following procedure template:

```
REAL PROCEDURE IOH_CLOSE (F, IOH_DATA, ASSOCIATION,
      DISPOSITION);
      VALUE ASSOCIATION, DISPOSITION;
      FILE F;
      ARRAY IOH_DATA [0:0];
      REAL ASSOCIATION, DISPOSITION;
```

Results

The result returned as the procedure value is defined in the *File Attributes Reference Manual*. These values are supported by the compilers, so that the mnemonic values can be used. For example, the following statement returns the appropriate good result value for the close request:

```
IOH_CLOSE := VALUE(OKRSLT);
```

Parameters

When passed into the CLOSE entry point, the ASSOCIATION and DISPOSITION parameters indicate the type of close procedure and correspond to the compiler-generated values passed to Logical I/O.

ASSOCIATION

Close Operation	Value
RELEASE	1
RETAIN	2
RESERVE	3
DISABLE	4

Note: *Although all values of ASSOCIATION are listed, the IOH_CLOSE entry point currently is called only for a value of RELEASE.*

DISPOSITION

Close Operation	Value
REWIND	1
NOREWIND	2
SAVE	3
LOCK	4
PURGE	5
CRUNCH	6
HERE	7
BLOCKEXIT	8
ABORT	9
ORDERLY	10
DOWNSIZEAREA	11
DOWNSIZEAREALOCK	12

IOH_READ

The IOH_READ entry point is optional, but necessary to service a read request on the virtual file. If exported, this entry point must be compatible with the following procedure template:

```
REAL PROCEDURE IOH_READ (F, IOH_DATA, FEATUREMASK, REC, LEN,
                         PDATA);
      VALUE FEATUREMASK, REC, PDATA;
      FILE F;
      ARRAY IOH_DATA [0:0];
      REAL FEATUREMASK, REC, LEN;
      POINTER PDATA;
```

Results

The result returned as the procedure value is defined in the *File Attributes Reference Manual*. These values are supported by the compilers, so that the mnemonic values can be used. For example, the following statement returns the appropriate good result value for the read request:

```
IOH_READ := VALUE(NOERROR);
```

Parameters

FEATUREMASK

The FEATUREMASK parameter contains an indication in the following field whether the read request was serial or random:

```
SERIAL_REQUESTF  =  [1:1] #  % Indicates Serial read/write
```

REC

The REC parameter is the record offset for the read request.

LEN

The LEN parameter is the length of the request in terms of IOHMODE characters. The value returned in LEN indicates how much data was actually transferred. The LEN value returned from the entry point must be less than or equal to the requested value or a DATAERROR result is returned to the program.

If the LEN value is less than requested, the MCP builds a result descriptor based on the IOHBLOCKSTRUCTURE returned from the open routine. For details about BLOCKSTURCTURE semantics, see "Virtual File Structure Considerations" earlier in this section.

PDATA

The PDATA parameter points to the internal Logical I/O buffer that is used to hold the user data during the I/O request. The pointer size is set based on the IOHMODE value returned from the IOH_OPEN entry point. The PDATA, REC, and LEN parameters are used to move data out of the library.

IOH_WRITE

The IOH_WRITE entry point is optional, but necessary to service a write request on the virtual file. If this entry point is exported, it must be compatible with the following procedure template:

```
REAL PROCEDURE IOH_WRITE (F, IOH_DATA, FEATUREMASK, REC, LEN,
                          PDATA);
      VALUE FEATUREMASK, REC, PDATA;
      FILE F;
      ARRAY IOH_DATA [0:0];
      REAL FEATUREMASK, REC, LEN;
      POINTER PDATA;
```

Results

The result returned as the procedure value is defined in the *File Attributes Reference Manual*. These values are supported by the compilers, so that the mnemonic values can be used. For example, the following statement returns the appropriate good result value for the write request:

```
IOH_WRITE := VALUE(NOERROR);
```

Parameters

FEATUREMASK

The FEATUREMASK parameter contains an indication whether the requested read procedure was serial or random and whether SYNCHRONIZE was specified. The layout of this parameter is as follows:

```
SERIAL_REQUESTF  =  [1:1] #  % Indicates Serial read/write
WRITE_SYNCF      =  [0:1] #  % Write [SYNCHRONIZE]
```

REC

The REC parameter is the record offset for the write request.

LEN

The LEN parameter is the length of the request in terms of IOHMODE characters. The value returned in LEN indicates how much data was actually transferred. The LEN value returned from the entry point must be equal to the requested value or a DATAERROR result is returned to the program.

The LEN value might be adjusted by Logical I/O to a record boundary based on the IOHBLOCKSTRUCTURE value returned from the open routine. For details about BLOCKSTURCTURE semantics, refer to "Virtual File Structure Considerations" earlier in this section.

PDATA

The PDATA parameter points to the internal Logical I/O buffer that is used to hold the user data during the I/O request. The pointer size is set based on the IOHMODE value returned from the IOH_OPEN entry point. The PDATA, REC, and LEN parameters are used to move data in to the library to service the write request.

IOH_FSYNC

The IOH_FSYNC entry point is optional, but necessary to service a POSIX_FSYNC or FSYNC request on the virtual file. The function normally is called to flush outstanding write requests. If FSYNC is requested for a virtual file, and the entry point has not been exported from the IOHANDLER library, the program receives a good result. If this entry point is exported, it must be compatible with the following procedure template:

```
REAL PROCEDURE IOH_FSYNC (F, IOH_DATA, VARIANT);
      VALUE   VARIANT;
      FILE    F;
      ARRAY   IOH_DATA [0:0];
      REAL    VARIANT;
```

Results

The result returned as the procedure value is defined in the *File Attributes Reference Manual*. These values are supported by the compilers, so that the mnemonic values can be used. For example, the following statement returns the appropriate good result value for the FSYNC request:

```
IOH_FSYNC := VALUE(NOERROR);
```

Parameters

VARIANT

The VARIANT parameter corresponds to the POSIX_FSYNC variables when defined as follows:

```
FSYNC_SV     = 1
FDSYNC_SV    = 2
```

IOH_ERASEFILE

The IOH_ERASEFILE entry point is optional, but necessary to service an ALGOL ERASE or ERASEFILE procedure for the virtual file. If this entry point is exported, it must be compatible with the following procedure template:

```
REAL PROCEDURE IOH_ERASEFILE (F, IOH_DATA);
      FILE    F;
      ARRAY  IOH_DATA [0:0];
```

Results

The result returned as the procedure value is defined in the *File Attributes Reference Manual*. These values are supported by the compilers, so that the mnemonic values can be used. For example, the following statement returns the appropriate good result value for the erase request:

```
IOH_ERASEFILE := VALUE(NOERROR);
```

IOH_GETATTRIBUTE

The IOH_GETATTRIBUTE entry point is optional, but necessary to service file attribute interrogation for the attributes listed in "Virtual File Attributes" earlier in this section. If this entry point is not exported, an attribute error is returned to the program doing the interrogation. If the entry point is exported, it must be declared to be compatible with the following procedure template:

```
REAL PROCEDURE IOH_GETATTRIBUTE (F, IOH_DATA, ATTNUM, ATTVALUE,
                ATTPOINTER);
      VALUE ATTNUM;
      FILE F;
      ARRAY IOH_DATA [0:0];
      REAL ATTNUM, ATTVALUE;
      POINTER ATTPOINTER;
```

Results

The result returned as the procedure value is expected to be 0 for a good result. If a non-zero result is returned, the MCP returns a generic attribute error.

Parameters

ATTNUM

The ATTNUM parameter is the attribute number used to select the attribute for interrogation. Attribute numbers are defined in the *File Attributes Reference Manual.* Attribute mnemonics also can be referred to in the program, as in VALUE(FILEKIND).

ATTVAL

The ATTVAL parameter returns the non-pointer valued attributes.

ATTPOINTER

The ATTPOINTER parameter points to the array in which pointer-valued attributes are returned.

IOH_SETATTRIBUTE

The IOH_SETATTRIBUTE entry point is optional, but necessary to service file attribute modification for the attributes listed in "Virtual File Attributes" earlier in this section. If the entry point is not exported, an attribute error is returned to the program doing the modification. If the entry point is exported, it must be declared to be compatible with the following procedure template:

```
REAL PROCEDURE IOH_SETATTRIBUTE (F, IOH_DATA, ATTNUM, ATTVALUE,
                                        ATTPOINTER);
      VALUE   ATTNUM, ATTVALUE, ATTPOINTER;
      FILE    F;
      ARRAY   IOH_DATA [0:0];
      REAL    ATTNUM, ATTVALUE;
      POINTER ATTPOINTER;
```

Results

The result returned as the procedure value is expected to be 0 for a good result. If a non-zero result is returned, the MCP returns a generic attribute error.

Parameters

ATTNUM

The ATTNUM parameter is the attribute number used to select the attribute for modification. Attribute numbers are defined in the *File Attributes Reference Manual*. Attribute mnemonics also can be referred to in the program, as in VALUE(FILEKIND).

ATTVAL

The ATTVAL parameter passes the value for setting the non-pointer valued attributes.

ATTPOINTER

The ATTPOINTER parameter points to the array holding the data for setting the pointer-valued attributes.

Example IOHANDLER Library

The following ALGOL program is an example IOHANDLER library that uses much of the virtual file functionality. The IOHANDLER acts as a surrogate disk file handler, using the entry parameter file F to create a disk file from a virtual file program. The IOHANDLER library corresponds to the DISKIOHANDLER function used in the example ALGOL and COBOL85 programs earlier in this section.

```
    $SET SHARING=SHAREDBYALL
  BEGIN

     DEFINE
        INFO_ITEMP           (INFO,X) = POINTER(INFO[X(INFO)]) #
       ,INFO_WORDS           (INFO)   = INFO  [0].[47:24] #
       ,INFO_FIXED_WORDS     (INFO)   = INFO  [0].[23:24] #
       ,INFO_VERSION         (INFO)   = INFO  [1].[07:08] #
          ,INFO_VERSIONV              = 1 #
       ,INFO_OPENATTRIBUTE   (INFO)   = INFO  [2].[36:01] #
       ,INFO_OPENIMPLICIT    (INFO)   = INFO  [2].[32:01] #
       ,INFO_OPENTYPE        (INFO)   = INFO  [2].[31:08] #
          ,OPENWAITV                  =  1 #
          ,AVAILABLEV                 =  2 #
          ,OFFERV                     =  3 #
          ,RESIDENTV                  =  4 #
          ,PRESENTV                   =  5 #
          ,PBTREELSWITCHOPENV         =  6 #
          ,STKASSIGNEDOPENREVERSEV    =  7 #
          ,F77BACKSPACEV              =  8 #
          ,UNCONDITIONALV             =  9 # % EVEN IF EXCLUSIVE
          ,SERVERREOPENV              = 10 #
          ,CONDITIONALV               = 11 # % POSIX OPEN CONDITIONAL
          ,MUSTBENEWV                 = 12 # % POSIX OPEN MUSTBENEW
       ,INFO_NEWFILE         (INFO)   = INFO  [2].[23:01] #
       ,INFO_EXCLUSIVE       (INFO)   = INFO  [2].[22:01] #
       ,INFO_FILEUSE         (INFO)   = INFO  [2].[21:02] #
       ,INFO_FLEXIBLE        (INFO)   = INFO  [2].[19:01] #
       ,INFO_NORESOURCEWAIT  (INFO)   = INFO  [2].[18:01] #
       ,INFO_SENSITIVEDATA   (INFO)   = INFO  [2].[17:01] #
       ,INFO_DEPENDENTINTMODE(INFO)   = INFO  [2].[16:01] #
       ,INFO_DEPENDENTSPECS  (INFO)   = INFO  [2].[12:01] #
       ,INFO_PROTECTION      (INFO)   = INFO  [2].[11:02] #
       ,INFO_SYNCHRONIZE     (INFO)   = INFO  [2].[09:02] #
       ,INFO_FILEKIND        (INFO)   = INFO  [2].[07:08] #
       ,INFO_FILEORGANIZATION(INFO)   = INFO  [3].[47:04] #
       ,INFO_FILESTRUCTURE   (INFO)   = INFO  [3].[43:08] #
       ,INFO_BLOCKSTRUCTURE  (INFO)   = INFO  [3].[35:04] #
       ,INFO_TRANSLATE       (INFO)   = INFO  [3].[31:04] #
       ,INFO_EXTMODEBITS     (INFO)   = INFO  [3].[27:08] #
       ,INFO_OVERRIDEEXTMODE (INFO)   = INFO  [3].[19:04] #
       ,INFO_EXTMODE         (INFO)   = INFO  [3].[15:16] #
```

```
,INFO_INTMODE         (INFO)   = INFO  [4].[47:16] #
,INFO_CCSVERSION      (INFO)   = INFO  [4].[31:16] #
,INFO_EXTDELIMITER    (INFO)   = INFO  [4].[15:08] #
,INFO_FRAMESIZE       (INFO)   = INFO  [4].[07:08] #
,INFO_BLOCKSIZE       (INFO)   = INFO  [5].[47:16] #
,INFO_MINRECSIZE      (INFO)   = INFO  [5].[31:16] #
,INFO_MAXRECSIZE      (INFO)   = INFO  [5].[15:16] #
,INFO_ANYSIZEIO       (INFO)   = INFO  [6].[44:01] #
,INFO_ADAPTABLE       (INFO)   = INFO  [6].[40:01] #
,INFO_UNITS           (INFO)   = INFO  [6].[36:01] #
,INFO_BUFFERSHARING   (INFO)   = INFO  [6].[33:02] #
,INFO_BUFFERS         (INFO)   = INFO  [6].[31:08] #
,INFO_BUFFERSIZE      (INFO)   = INFO  [6].[23:24] #
,INFO_AREAS           (INFO)   = INFO  [7].[47:16] #
,INFO_AREALENGTH      (INFO)   = INFO  [7].[31:32] #
,INFO_AREASIZE        (INFO)   = INFO  [8].[47:24] #
,INFO_SECURITYTYPE    (INFO)   = INFO  [8].[23:04] #
,INFO_SECURITYUSE     (INFO)   = INFO  [8].[19:04] #
,INFO_SECURITYMODE    (INFO)   = INFO  [8].[15:16] #
,INFO_FILESNR         (INFO)   = INFO  [9] #
,INFO_FILEMIX         (INFO)   = INFO [10] #

% Variable Length Attributes
,INFO_ATT_LINKF   =  [47:24] #
,INFO_ATT_LENF    =  [23:24] #
,INFO_FILENAMEINX     (INFO)   = INFO [11].INFO_ATT_LINKF #
,INFO_FILENAMECHRS    (INFO)   = INFO [11].INFO_ATT_LENF #
,INFO_FAMILYNAMEINX   (INFO)   = INFO [12].INFO_ATT_LINKF #
,INFO_FAMILYNAMECHRS  (INFO)   = INFO [12].INFO_ATT_LENF #
,INFO_STRINGINX       (INFO)   = INFO [13].INFO_ATT_LINKF #
,INFO_STRINGCHRS      (INFO)   = INFO [13].INFO_ATT_LENF #
,INFO_FILEUCINX       (INFO)   = INFO [14].INFO_ATT_LINKF #
,INFO_FILEUCCHRS      (INFO)   = INFO [14].INFO_ATT_LENF #
,INFO_NOTEINX         (INFO)   = INFO [15].INFO_ATT_LINKF #
,INFO_NOTECHRS        (INFO)   = INFO [15].INFO_ATT_LENF #
,INFO_GUARDFILEINX    (INFO)   = INFO [16].INFO_ATT_LINKF #
,INFO_GUARDFILECHRS   (INFO)   = INFO [16].INFO_ATT_LENF #
,INFO_GROUPINX        (INFO)   = INFO [17].INFO_ATT_LINKF #
,INFO_GROUPCHRS       (INFO)   = INFO [17].INFO_ATT_LENF #
,INFO_ALTGROUPSINX    (INFO)   = INFO [18].INFO_ATT_LINKF #
,INFO_ALTGROUPSCHRS   (INFO)   = INFO [18].INFO_ATT_LENF #
,INFO_OWNERINX        (INFO)   = INFO [19].INFO_ATT_LINKF #
,INFO_OWNERCHRS       (INFO)   = INFO [19].INFO_ATT_LENF #
,INFO_INTNAMEINX      (INFO)   = INFO [20].INFO_ATT_LINKF #
,INFO_INTNAMECHRS     (INFO)   = INFO [20].INFO_ATT_LENF #
,INFO_HOSTNAMEINX     (INFO)   = INFO [21].INFO_ATT_LINKF #
,INFO_HOSTNAMECHRS    (INFO)   = INFO [21].INFO_ATT_LENF #
,INFO_FIRSTINFO                = INFO [22] #
```

```
     ,INFO_STRINGP(INFO)    = INFO_ITEMP(INFO,INFO_STRINGINX)   #
     ,INFO_FILENAMEP(INFO)  = INFO_ITEMP(INFO,INFO_FILENAMEINX) #
     ,INFO_FAMILYNAMEP(INFO)=INFO_ITEMP(INFO,INFO_FAMILYNAMEINX)#
     ,INFO_FILEUCP(INFO)    = INFO_ITEMP(INFO,INFO_FILEUCINX)   #
     ,INFO_NOTEP(INFO)      = INFO_ITEMP(INFO,INFO_NOTEINX)     #
     ,INFO_GROUPP (INFO)    = INFO_ITEMP(INFO,INFO_GROUPINX)    #
     ,INFO_ALTGROUPSP(INFO) = INFO_ITEMP(INFO,INFO_ALTGROUPSINX)#
     ,INFO_GUARDFILEP(INFO) = INFO_ITEMP(INFO,INFO_GUARDFILEINX)#
     ,INFO_INTNAMEP (INFO)  = INFO_ITEMP(INFO,INFO_INTNAMEINX)  #
     ,INFO_HOSTNAMEP (INFO) = INFO_ITEMP(INFO,INFO_HOSTNAMEINX) #
     ;
DEFINE FILE_RECSIZE   = IOHSTATE [0] #,
       FILE_CHARSIZE  = IOHSTATE [1] #,
       FILE_FRAMESIZE = IOHSTATE [2] #,
       FILE_UNITS     = IOHSTATE [3] #,
       IOH_STATE_WORDS= 4 #;

DEFINE BITSPERBYTE =  8 #,
       BITSPERWORD = 48 #,
       BITSPERUNIT(M) =
         (IF M > 6 THEN
            IF M=VALUE(UCS2NT) THEN 16
              ELSE 8
            ELSE
             (0 & 48 [ 0:6]              % CASE (M) OF (48
                & 48 [ 6:6]              %             ,48
                &  4 [12:6]              %             ,4
                &  6 [18:6]              %             ,6
                &  8 [24:6]              %             ,8
                &  8 [30:6]              %             ,8
                &  0 [36:6]).[(M)*6:6] )#;%            ,0)

DEFINE COVQ  (V, D) = (((V) + (D) - 1) DIV (D)) #,
       COVER (V, D) = ((D) * (COVQ ((V), (D)))) #;

REAL PROCEDURE IOH_OPEN (F,IOHDATA,IOHINFO,RECSIZE,MODE,
                         MAXXMT,ACCESS,BLOCKSTRUCT);
  FILE F;
  ARRAY IOHDATA [0,0], IOHINFO [0];
  REAL RECSIZE,MODE,MAXXMT,ACCESS,BLOCKSTRUCT;
  BEGIN
     REAL    OPENRSLT,
             FILE_USE;
     BOOLEAN CREATEFILE,
             DEPENDENT_SPECS;
     ARRAY REFERENCE
             IOHSTATE [0];
```

```
ON ANYFAULT, PROGRAMDUMP(ALL,TODISK);

     IOHSTATE := IOHDATA [0,*];
     IF SIZE(IOHSTATE) < IOH_STATE_WORDS THEN
       RESIZE(IOHSTATE,IOH_STATE_WORDS);
     CREATEFILE      := BOOLEAN(INFO_NEWFILE(IOHINFO));
     DEPENDENT_SPECS := BOOLEAN(INFO_DEPENDENTSPECS(IOHINFO));
     IF INFO_FILENAMECHRS(IOHINFO) > 0 THEN
       REPLACE F.LFILENAME BY INFO_FILENAMEP(IOHINFO);
     IF INFO_FAMILYNAMECHRS(IOHINFO) > 0 THEN
       REPLACE F.FAMILYNAME BY INFO_FAMILYNAMEP(IOHINFO);
     IF INFO_HOSTNAMECHRS(IOHINFO) > 0 THEN
       REPLACE F.HOSTNAME BY INFO_HOSTNAMEP(IOHINFO);

     IF INFO_BUFFERS(IOHINFO) > 0 THEN
       F.BUFFERS := INFO_BUFFERS(IOHINFO);
     IF INFO_BUFFERSIZE(IOHINFO) > 0 THEN
       F.BUFFERSIZE := INFO_BUFFERSIZE(IOHINFO);
     IF INFO_BUFFERSHARING(IOHINFO) NEQ VALUE(NONE) THEN
       F.BUFFERSHARING := INFO_BUFFERSHARING(IOHINFO);
     F (KIND             = DISK
       ,NEWFILE          = CREATEFILE
       ,FILEUSE          = INFO_FILEUSE(IOHINFO)
       ,EXCLUSIVE        = BOOLEAN(INFO_EXCLUSIVE(IOHINFO))
       ,ANYSIZEIO        = BOOLEAN(INFO_ANYSIZEIO(IOHINFO))
       ,SYNCHRONIZE      = INFO_SYNCHRONIZE(IOHINFO)
       ,ADAPTABLE        = BOOLEAN(INFO_ADAPTABLE(IOHINFO))
       ,DEPENDENTSPECS   = DEPENDENT_SPECS
       ,DEPENDENTINTMODE = TRUE  % Translation already done.
       );
     IF CREATEFILE OR NOT DEPENDENT_SPECS THEN
       BEGIN
         FILE_CHARSIZE  := INFO_EXTMODEBITS(IOHINFO);
         FILE_UNITS     := IF INFO_FRAMESIZE(IOHINFO) = 0 THEN
                             INFO_UNITS(IOHINFO)
                           ELSE
                           IF INFO_FRAMESIZE(IOHINFO) = 48 THEN
                             VALUE(WORDS)
                           ELSE
                             VALUE(CHARACTERS);
         FILE_FRAMESIZE := IF FILE_UNITS = VALUE(WORDS) THEN 48
                           ELSE FILE_CHARSIZE;
         IF FILE_UNITS = VALUE(WORDS) THEN  % convert to words
           RECSIZE := RECSIZE DIV
                      (BITSPERWORD DIV FILE_CHARSIZE);
```

```
F (MAXRECSIZE       = RECSIZE
          ,FRAMESIZE        = FILE_FRAMESIZE
          ,INTMODE          = MODE
          ,EXTMODE          = MODE
          ,BLOCKSTRUCTURE   = BLOCKSTRUCT
          ,FILEKIND         = INFO_FILEKIND(IOHINFO)
          ,FILESTRUCTURE    = INFO_FILESTRUCTURE(IOHINFO)
          ,FILEORGANIZATION = INFO_FILEORGANIZATION(IOHINFO)
          ,SECURITYMODE     = INFO_SECURITYMODE(IOHINFO)
          ,PROTECTION       = INFO_PROTECTION(IOHINFO)
          );

        IF INFO_FILESTRUCTURE(IOHINFO) NEQ VALUE(STREAM) AND
           INFO_BLOCKSIZE(IOHINFO) > 0 THEN
          F.BLOCKSIZE := INFO_BLOCKSIZE(IOHINFO);
        IF INFO_AREAS(IOHINFO) > 0 THEN
          F.AREAS := INFO_AREAS(IOHINFO);
        IF INFO_AREASIZE(IOHINFO) > 0 THEN
          F.AREASIZE := INFO_AREASIZE(IOHINFO);
        IF INFO_AREALENGTH(IOHINFO) > 0 THEN
          F.AREALENGTH := INFO_AREALENGTH(IOHINFO);
        IF INFO_NOTECHRS(IOHINFO) > 0 THEN
          REPLACE F.NOTE BY INFO_NOTEP(IOHINFO);
        IF INFO_STRINGCHRS(IOHINFO) > 0 THEN
          REPLACE F.IOHSTRING BY INFO_STRINGP(IOHINFO);
        IF INFO_GROUPCHRS(IOHINFO) > 0 THEN
          REPLACE F.GROUP BY INFO_GROUPP(IOHINFO);
        IF INFO_ALTGROUPSCHRS(IOHINFO) > 0 THEN
          REPLACE F.ALTERNATEGROUPS BY INFO_ALTGROUPSP(IOHINFO);
        IF INFO_SECURITYTYPE(IOHINFO) = VALUE(GUARDED) AND
           INFO_GUARDFILECHRS(IOHINFO) > 0 THEN
          REPLACE F.SECURITYGUARD BY INFO_GUARDFILEP(IOHINFO);
      END;

    OPENRSLT := IF INFO_OPENTYPE(IOHINFO) = RESIDENTV THEN
                  IF F.RESIDENT THEN VALUE(OKRSLT)
                                ELSE VALUE(NOFILEFOUNDRSLT)
                ELSE
                  F.AVAILABLE;

    IF OPENRSLT = VALUE(OKRSLT) THEN
      IF INFO_OPENTYPE(IOHINFO) NEQ RESIDENTV OR
         (NOT CREATEFILE AND INFO_DEPENDENTSPECS(IOHINFO)=1)THEN
        BEGIN
          FILE_RECSIZE   := F.MAXRECSIZE;
          FILE_FRAMESIZE := F.FRAMESIZE;
          FILE_UNITS     := F.UNITS;
          MODE           := F.INTMODE;
          FILE_CHARSIZE  := BITSPERUNIT(MODE);
```

```
            % Return the RECSIZE in MODE Character Size.
            RECSIZE := FILE_RECSIZE*FILE_FRAMESIZE DIV
                       FILE_CHARSIZE;
            MAXXMT  := IF BOOLEAN(INFO_ANYSIZEIO(IOHINFO)) THEN
                         IF INFO_BUFFERSIZE(IOHINFO) > 0 THEN
                           INFO_BUFFERSIZE(IOHINFO)
                         ELSE
                           65535
                       ELSE
                         RECSIZE;
            MAXXMT := MAX (1, MAXXMT DIV RECSIZE);% in records
            FILE_USE := F.FILEUSE;
            ACCESS   := 0 & REAL(FILE_USE NEQ VALUE(OUT)) [2:1]
                          & REAL(FILE_USE NEQ VALUE(IN))  [1:1];
            BLOCKSTRUCT := F.BLOCKSTRUCTURE;
          END;
      IOH_OPEN := OPENRSLT;
    END IOH_OPEN;

  REAL PROCEDURE IOH_CLOSE (F, IOHDATA, ASSOCIATION, DISPOSITION);
    VALUE ASSOCIATION, DISPOSITION;
    FILE F;
    ARRAY IOHDATA [0,0];
    REAL  ASSOCIATION, DISPOSITION;

    BEGIN
    DEFINE
      % ASSOCIATION
        RELEASEV = 1 #
       ,RETAINV  = 2 #
       ,RESERVEV = 3 #
       ,DISABLEV = 4 #

      % DISPOSITION
       ,REWINDV     = 1 #
       ,NOREWINDV   = 2 #
       ,SAVEV       = 3 #
       ,LOCKV       = 4 #
       ,PURGEV      = 5 #
       ,CRUNCHV     = 6 #
       ,HEREV       = 7 #  % (ASSOCIATION MUST BE RETAINV)
       ,BLOCKEXITV  = 8 #
       ,ABORTV      = 9 #
       ,ORDERLYV   = 10 #
       ,DOWNSIZEAREAV =11 #
       ,DOWNSIZEAREALOCKV = 12 #
       ;
```

```
ON ANYFAULT, PROGRAMDUMP(ALL,TODISK);

  IF NOT F.OPEN THEN
    IOH_CLOSE := VALUE(OKRSLT)
  ELSE
    CASE DISPOSITION.[3:4] OF
      BEGIN
        REWINDV:
            IOH_CLOSE := CLOSE (F,REWIND);
        LOCKV  :
        SAVEV  :
            IOH_CLOSE := CLOSE (F,LOCK);
        CRUNCHV:
            IOH_CLOSE := CLOSE (F,CRUNCH);
        PURGEV:
            IOH_CLOSE := CLOSE (F,PURGE);

        DOWNSIZEAREAV:
            IOH_CLOSE := CLOSE (F,DOWNSIZEAREA);
        DOWNSIZEAREALOCKV:
            IOH_CLOSE := CLOSE (F,DOWNSIZEAREALOCK);
        ELSE:
            IOH_CLOSE := CLOSE (F);
      END;
  END IOH_CLOSE;

REAL PROCEDURE IOH_READ (F, IOHDATA, FEATUREMASK, REC, LEN, PDATA);
  VALUE FEATUREMASK, REC, PDATA;
  FILE F;
  ARRAY IOHDATA [0,0];
  REAL FEATUREMASK, REC, LEN;
  POINTER PDATA;
  BEGIN
  REAL    RD;
  BOOLEAN BRD = RD;
  ARRAY REFERENCE IOHSTATE [0];
  ON ANYFAULT, PROGRAMDUMP(ALL,TODISK);
  IOHSTATE := IOHDATA [0,*];
  BRD := READ (F [REC],
               LEN*FILE_CHARSIZE DIV FILE_FRAMESIZE, PDATA);
  LEN := RD.[47:20]*FILE_FRAMESIZE DIV FILE_CHARSIZE;
  IOH_READ := IF BRD THEN RD.[26:10] ELSE VALUE(NOERROR);
  END IOH_READ;

REAL PROCEDURE IOH_WRITE (F, IOHDATA, FEATUREMASK, REC, LEN, PDATA);
  VALUE FEATUREMASK, REC, PDATA;
  FILE F;
  ARRAY IOHDATA [0,0];
  REAL FEATUREMASK, REC, LEN;
  POINTER PDATA;
  BEGIN
  REAL    RD;
```

```
  BOOLEAN BRD = RD;
  ARRAY REFERENCE IOHSTATE [0];
  ON ANYFAULT, PROGRAMDUMP(ALL,TODISK);
  IOHSTATE := IOHDATA [0,*];
  BRD := IF BOOLEAN(FEATUREMASK) THEN
           WRITE (F [REC],
                  LEN*FILE_CHARSIZE DIV FILE_FRAMESIZE, PDATA)
         ELSE
           WRITE (F [REC],
                  LEN*FILE_CHARSIZE DIV FILE_FRAMESIZE, PDATA);
  LEN := RD.[47:20]*FILE_FRAMESIZE DIV FILE_CHARSIZE;
  IOH_WRITE := IF BRD THEN RD.[26:10] ELSE VALUE(NOERROR);
  END IOH_WRITE;

LIBRARY MCPSUPPORT (LIBACCESS=BYFUNCTION);

BOOLEAN PROCEDURE FILESYNC(F,FORCEHEADERWRITE);
   VALUE   FORCEHEADERWRITE;
   FILE    F;
   BOOLEAN FORCEHEADERWRITE;
   LIBRARY MCPSUPPORT;

REAL PROCEDURE IOH_FSYNC (F, IOHDATA, VARIANT);
    VALUE VARIANT;
    FILE  F;
    ARRAY IOHDATA [0,0];
    REAL  VARIANT;
  BEGIN
    BOOLEAN FSYNC;
    FSYNC := FILESYNC (F, BOOLEAN(VARIANT));
    IOH_FSYNC := REAL(FSYNC).[26:10];
  END IOH_FSYNC;

REAL PROCEDURE IOH_ERASEFILE (F, IOHDATA);
    FILE  F;
    ARRAY IOHDATA [0,0];
  BEGIN
    IOH_ERASEFILE := REAL(ERASE (F));
  END IOH_ERASEFILE;

REAL PROCEDURE IOH_GETATTRIBUTE (F, IOHDATA, ATTNUM, ATTVAL, ATTPTR);
    VALUE ATTNUM;
    FILE  F;
    ARRAY IOHDATA [0,0];
    REAL  ATTNUM, ATTVAL;
    POINTER ATTPTR;
```

```
BEGIN
  BOOLEAN BATTVAL = ATTVAL;
    ON ANYFAULT, PROGRAMDUMP(ALL,TODISK);
    CASE ATTNUM OF
    BEGIN
    VALUE(ACCESSDATE):
       ATTVAL := F.ACCESSDATE;
    VALUE(ACCESSTIME):
       ATTVAL := F.ACCESSTIME;
    VALUE(ALTERDATE):
       ATTVAL := F.ALTERDATE;
    VALUE(ALTERTIME):
       ATTVAL := F.ALTERTIME;
    VALUE(CREATIONDATE):
       ATTVAL := F.CREATIONDATE;
    VALUE(CREATIONTIME):
       ATTVAL := F.CREATIONTIME;
    VALUE(SYNCHRONIZE):
       ATTVAL := F.SYNCHRONIZE;
    VALUE(SECURITYTYPE):
       ATTVAL := F.SECURITYTYPE;
    VALUE(SECURITYUSE):
       ATTVAL := F.SECURITYUSE;
    VALUE(SECURITYMODE):
       ATTVAL := F.SECURITYMODE;
    VALUE(OWNERRWX):
       ATTVAL := F.OWNERRWX;
    VALUE(GROUPRWX):
       ATTVAL := F.GROUPRWX;
    VALUE(OTHERRWX):
       ATTVAL := F.OTHERRWX;
    VALUE(OWNERR):
       BATTVAL := F.OWNERR;
    VALUE(OWNERW):
       BATTVAL := F.OWNERW;
    VALUE(OWNERX):
       BATTVAL := F.OWNERX;
    VALUE(GROUPR):
       BATTVAL := F.GROUPR;
    VALUE(GROUPW):
       BATTVAL := F.GROUPW;
    VALUE(GROUPX):
       BATTVAL := F.GROUPX;
    VALUE(OTHERR):
       BATTVAL := F.OTHERR;
    VALUE(OTHERW):
       BATTVAL := F.OTHERW;
    VALUE(OTHERX):
       BATTVAL := F.OTHERX;
    VALUE(USEGUARDFILE):
       BATTVAL := F.USEGUARDFILE;
```

```
    VALUE(GUARDOWNER):
       BATTVAL := F.GUARDOWNER;
    VALUE(SETUSERCODE):
       BATTVAL := F.SETUSERCODE;
    VALUE(SETGROUPCODE):
       BATTVAL := F.SETGROUPCODE;
    VALUE(GROUP):
       REPLACE ATTPTR:ATTPTR BY F.GROUP;
    VALUE(OWNER):
       REPLACE ATTPTR:ATTPTR BY F.OWNER;
    VALUE(ALTERNATEGROUPS):
       REPLACE ATTPTR:ATTPTR BY F.ALTERNATEGROUPS;
    VALUE(IOHSTRING):
       REPLACE ATTPTR:ATTPTR BY F.IOHSTRING;
    VALUE(NOTE):
       REPLACE ATTPTR:ATTPTR BY F.NOTE;
    VALUE(FILELENGTH):
       ATTVAL := F.FILELENGTH;
    VALUE(LASTRECORD):
       ATTVAL := F.LASTRECORD;
    VALUE(CCSVERSION):
       ATTVAL := F.CCSVERSION;
    VALUE(EXTDELIMITER):
       ATTVAL := F.EXTDELIMITER;
    VALUE(FILEKIND):
       ATTVAL := F.FILEKIND;
    VALUE(USERINFO):
       ATTVAL := F.USERINFO;
    ELSE:
       IOH_GETATTRIBUTE := 1;
    END;
END IOH_GETATTRIBUTE;

REAL PROCEDURE IOH_SETATTRIBUTE (F, IOHDATA, ATTNUM, ATTVAL, ATTPTR);
  VALUE ATTNUM, ATTVAL, ATTPTR;
  FILE  F;
  ARRAY IOHDATA [0,0];
  REAL  ATTNUM, ATTVAL;
  POINTER ATTPTR;
BEGIN
  POINTER P;
  BOOLEAN BATTVAL = ATTVAL;
    ON ANYFAULT, PROGRAMDUMP(ALL,TODISK);
    CASE ATTNUM OF
    BEGIN
    VALUE(SYNCHRONIZE):
       F.SYNCHRONIZE := ATTVAL;
    VALUE(SECURITYTYPE):
       F.SECURITYTYPE := ATTVAL;
    VALUE(SECURITYUSE):
       F.SECURITYUSE := ATTVAL;
```

```
VALUE(SECURITYMODE):
   F.SECURITYMODE := ATTVAL;
VALUE(OWNERRWX):
   F.OWNERRWX := ATTVAL;
VALUE(GROUPRWX):
   F.GROUPRWX := ATTVAL;
VALUE(OTHERRWX):
   F.OTHERRWX := ATTVAL;
VALUE(OWNERR):
   F.OWNERR := BATTVAL;
VALUE(OWNERW):
   F.OWNERW := BATTVAL;
VALUE(OWNERX):
   F.OWNERX := BATTVAL;
VALUE(GROUPR):
   F.GROUPR := BATTVAL;
VALUE(GROUPW):
   F.GROUPW := BATTVAL;
VALUE(GROUPX):
   F.GROUPX := BATTVAL;
VALUE(OTHERR):
   F.OTHERR := BATTVAL;
VALUE(OTHERW):
   F.OTHERW := BATTVAL;
VALUE(OTHERX):
   F.OTHERX := BATTVAL;
VALUE(GROUP):
   REPLACE F.GROUP BY ATTPTR;
VALUE(OWNER):
   REPLACE F.OWNER BY ATTPTR;
VALUE(ALTERNATEGROUPS):
   REPLACE F.ALTERNATEGROUPS BY ATTPTR;
VALUE(TITLE):
   REPLACE F.TITLE BY ATTPTR;
VALUE(LTITLE):
   REPLACE F.LTITLE BY ATTPTR;
VALUE(FILENAME):
   REPLACE F.FILENAME BY ATTPTR;
VALUE(LFILENAME):
   REPLACE F.LFILENAME BY ATTPTR;
VALUE(IOHSTRING):
   REPLACE F.IOHSTRING BY ATTPTR;
VALUE(NOTE):
   REPLACE F.NOTE BY ATTPTR;
VALUE(CCSVERSION):
   F.CCSVERSION := ATTVAL;
VALUE(EXTDELIMITER):
   F.EXTDELIMITER := ATTVAL;
VALUE(FILEKIND):
   F.FILEKIND := ATTVAL;
```

```
      VALUE(USERINFO):
         F.USERINFO := ATTVAL;
      ELSE:
         ;
      END;
  END IOH_SETATTRIBUTE;

EXPORT
     IOH_OPEN
    ,IOH_CLOSE
    ,IOH_READ
    ,IOH_WRITE
    ,IOH_FSYNC
    ,IOH_ERASEFILE
    ,IOH_GETATTRIBUTE
    ,IOH_SETATTRIBUTE
    ;

FREEZE (PERMANENT);

END.
```

Section 30
Using the REDIRSUPPORT IOHANDLER Library

In conjunction with the new virtual file implementation, the REDIRSUPPORT IOHANDLER library provides access to network resources by utilizing Server Message Block (SMB) protocol redirection. REDIRSUPPORT IOHANDLER supports access to Printer, Disk, and CD shares in a Microsoft Networking (TCP/IP) environment. The REDIRSUPPORT IOHANDLER feature can be used only on ClearPath NX and LX systems. For details about the MCP implementation of TCP/IP, refer to the *TCP/IP Distributed System Services Operations Guide*.

This section describes how to use the REDIRSUPPORT IOHANDLER library for normal file access as well as for directory manipulation.

For information on the print redirection capabilities of the REDIRSUPPORT feature, refer to *Unisys e-@ction ClearPath Enterprise Servers Installing a Printer for MCP Print System Use*.

For details on any of the attributes referenced in this section, refer to the *File Attributes Reference Manual*.

Accessing REDIRSUPPORT IOHANDLER

Programmatic access to files on network shares is provided through the virtual file mechanism, using the REDIRSUPPORT IOHANDLER library. There are two methods to identify a file that is requesting service by REDIRSUPPORT IOHANDLER, as follows:

- Set the REDIRECTION file attribute to TRUE. The following attributes are implicitly set:

  ```
  KIND = VIRTUAL

  IOHLIBACCESS = BYFUNCTION

  IOHFUNCTIONNAME = REDIRSUPPORT
  ```

- Set the attributes KIND, IOHLIBACCESS, and IOHFUNCTIONNAME explicitly as part of the file declaration or file equation.

Redirector File Structure

The REDIRSUPPORT IOHANDLER library supports access to a class of files described as byte-stream files. The REDIRSUPPORT file structure corresponds to a disk file with the following physical attributes:

- FRAMESIZE = 8
- MAXRECSIZE = 1
- BLOCKSTRUCTURE = FIXED
- FILESTRUCTURE = STREAM
- FILEORGANIZATION = NOTRESTRICTED
- EXTMODE = ASCII

In order to open a file on a network share using REDIRSUPPORT IOHANDLER, the declared file must be compatible with the preceding attributes. For existing files, setting the DEPENDENTSPECS attribute to TRUE guarantees this compatibility. When creating a file (NEWFILE = TRUE) or when DEPENDENTSPECS is not TRUE, you must explicitly set the file attributes to these values for the open procedure to succeed.

In addition to the physical attributes of the file, byte-stream files are normally accessed with ANYSIZEIO set to TRUE to permit I/O requests of more than one character at a time.

For additional information about programming for REDIRSUPPORT IOHANDLER access to network files, refer to "REDIRSUPPORT IOHANDLER Considerations for Use" later in this section.

For general information on accessing byte-stream files, refer to "Using Byte Files in a Program" in Section 2, "Understanding Programming for Files."

Locating a Network File

Before a file can be opened and accessed, its location must be specified through an appropriate attribute or combination of attributes. The following paragraphs describe the various mechanisms that can be used to identify a network file to the REDIRSUPPORT IOHANDLER library.

Several pieces of information must be conveyed to REDIRSUPPORT IOHANDLER to locate and access a network file. In many cases, this information can be conveyed using a naming convention defined expressly for REDIRSUPPORT IOHANDLER. However, it is not always possible to locate and successfully open a network file from the MCP environment simply by naming the file.

IOHSTRING Parameters

To provide additional information for controlling access to a network file, REDIRSUPPORT IOHANDLER supports use of the new virtual file string attribute IOHSTRING. The IOHSTRING attribute is used for passing information between a virtual file program and the IOHANDLER library associated with the virtual file. The REDIRSUPPORT IOHANDLER library defines keywords to be contained in the IOHSTRING attribute for passing information about the network file to be opened. Table 30–1 lists the keywords described in this section.

Table 30–1. REDIRSUPPORT IOHANDLER Keywords

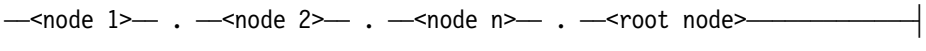
Keyword	Description	Example
DOMAINNAME	Domain name services (DNS) name	TAD.FROG.POND.COM
IPADDRESS	Internet protocol (IP) address	201.63.228.29
SERVERNAME	Computer server (NETBIOS) name	EAST-SVR
SHARENAME	Server-defined share name	PUBLIC
CREDENTIALS	Account name and password on server	ABC/PW
USERDOMAIN	User domain used to verify CREDENTIALS	CORP_SEC
TIMEOUT	Length of time to wait for network resources	Time in seconds
SMBTRACE	Diagnostic tool used to trace SMB traffic	TRUE/FALSE

The syntax and usage of each of the keywords are described in the following paragraphs. The shortest abbreviation for each keyword is always shown in capital letters.

DOMAINNAME

The DOMAINNAME keyword specifies the server domain name used by RESOLVER to locate the destination system. The general syntax of a domain name is shown below. For a complete description of domain names, refer to the *TCP/IP Distributed Systems Services Operations Guide*.

Syntax



A node must be structured as follows:

- The name must begin with a numeral or letter.
- The name must end with a numeral or letter.
- Internal characters can be a letter, numeral, hyphen (-), or an underscore (_).
- The entire node cannot contain more than 63 characters.
- The entire domain name string cannot contain more than 255 characters, including the separating periods.

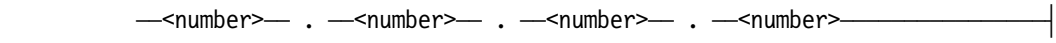
Example

```
DOMAINNAME = TADPOLE.FROG.POND.COM
```

IPADDRESS

The IPADDRESS keyword specifies the IP address of the destination server when normal name resolution fails.

Syntax



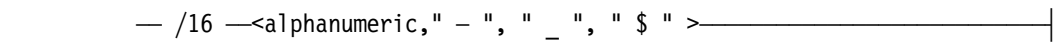
Example

```
IPADDRESS = 128.63.224.12
```

SERVERNAME

The SERVERNAME keyword specifies the name of the destination server. This keyword must be used with SHARENAME.

Syntax



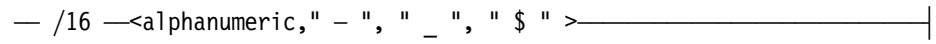
Example

```
SERVERNAME = REGION1SVR
```

SHARENAME

The SHARENAME keyword specifies the name of the share on the remote system. This keyword must be used with SERVERNAME.

Syntax



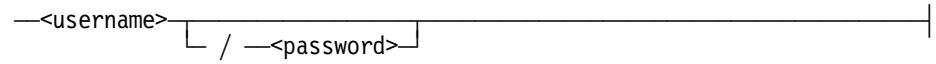
Example

```
SHARENAME = MYSHARE
SHARENAME = C$
```

CREDENTIALS

The CREDENTIALS keyword explicitly specifies a username and password.

Syntax



Examples

```
CREDENTIALS = USER1/FROG
```

A user name of USER1 and a password of FROG are used.

```
CREDENTIALS = USER1
```

A user name of USER1 and a password of GUEST are used.

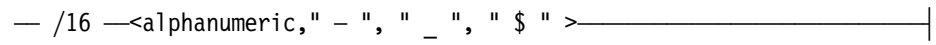
```
CREDENTIALS = /FROG
```

A null user name and a password of FROG are used.

USERDOMAIN

The USERDOMAIN attribute is used to specify the name of the Windows NT domain to be used when authenticating the login request. This information is advisory only; it cannot be guaranteed that the server will honor it.

Syntax



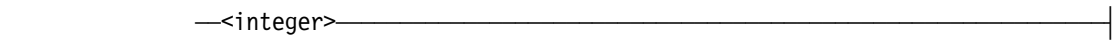
Example

```
USERDOMAIN = A_SERVER
```

TIMEOUT

The TIMEOUT keyword specifies the timeout value in seconds used by the SMB IOHANDLER library when timing out SMB responses.

Syntax



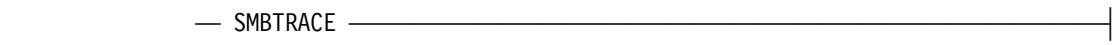
Example

```
TIMEOUT = 30
```

SMBTRACE

The SMBTRACE keyword causes all SMB requests and responses to be written to an appropriate trace file for diagnostic analysis.

Syntax



Uniform Naming Convention

In a Microsoft networking environment, the location of a file is specified with the Uniform Naming Convention (UNC) as defined in the X/OPEN Standard for PC Interworking.

An UNC name has the following general syntax:

```
\\<host>\<share>\<path>\<filename>
```

In the following example, the UNC name refers to the file doc1.doc in the temp directory within the PUBLIC share on the computer named TR-PRESTIJC:

```
\\TR-PRESTIJC\PUBLIC\temp\doc1.doc
```

In the PC environment, files are usually located by mapping a network share as a device and specifying the location of the file as being within a path on the mapped device. The mapping process requires the name of the computer and the name of the share, as in the \\<host>\<share> portion of the UNC name.

In the MCP environment, however, there is no equivalent device mapping mechanism. The format of the \<path>\<file name> portion of the UNC name does not conform to the MCP naming convention of <directory>/<file name>. Therefore, the REDIRSUPPORT IOHANDLER supports a specific naming convention for locating a network file by its name. This syntax is as follows:

```
*UNC/<host>/<share>/<path>/<filename>
```

When the FILENAME attribute starts with the node *UNC, the REDIRSUPPORT IOHANDLER assumes it has been given a typical UNC name following the *UNC node. The REDIRSUPPORT IOHANDLER converts the *UNC variant of a standard MCP file into a typical UNC name as part of opening the file. The conversion strips the quotes around the file name nodes and changes the slashes to back slashes. The REDIRSUPPORT IOHANDLER requires that a host name, share name, and at least one file name node appear in the name following the *UNC node or an open error is generated.

This naming convention is applied to the previous example in the following manner:

```
*UNC/TR-PRESTIJC/PUBLIC/"temp"/"doc1.doc"
```

The <host> portion of the UNC name can be in any of several forms. These forms correspond to the SERVERNAME, DOMAINNAME, and IPADDRESS keywords described in "IOHSTRING Parameters," earlier in this section.

The last two nodes are quoted as required by MCP file naming rules for lowercase and special characters. *UNC is a naming convention that is interpreted by the REDIRSUPPORT IOHANDLER only for redirected files; it has no effect or meaning outside this usage.

Relative File Names

As a second way of locating a file on a remote system, you can use the IOHSTRING attribute in conjunction with a "relative" file name. A "relative" file name is a name that does not specify the host name and share name, and so does not begin with *UNC. In this case, you specify the server and share name as part of the IOHSTRING attribute, using the SERVERNAME and SHARENAME parameters. The file name used in the previous examples corresponds to the following file attribute specification as follows:

```
FILE VF (...
    ,IOHSTRING = "SERVER=TR-PRESTIJC SHARE=PUBLIC."
    ,FILENAME  = ""temp"""/"""doc1.doc""""."
    ...
    );
```

As with the *UNC variant of a standard MCP file name, the MCP converts the file name into a PC-style relative file name by stripping the quotes around the file name nodes and changing the slashes to back slashes.

If you specify a full UNC name and also specify a server and share in the IOHSTRING attribute, the server and share specified within the UNC name take precedence.

NXSERVICES CONFIG Files

An additional mechanism for locating a network file combines a naming convention with the IOHSTRING parameter keywords stored in a configuration file. The naming convention is

```
*NXCONFIG/<config name>/<path>/<file>
```

The *NXCONFIG node indicates to the REDIRSUPPORT IOHANDLER that IOHSTRING keyword parameters are maintained in a file with the name

```
NXSERVICES/CONFIG/<config name>
```

Normal usercode and family substitution is applied when searching for configuration files. Normal MCP security governs a user's access to an NXSERVICES/CONFIG file.

The REDIRSUPPORT IOHANDLER permits the use of configuration files with printer shares as described for TEMPLATE printers in *Installing a Printer for Print System Use*.

Credentials

Credentials are needed to specify the username and password to be used on the remote host. Typically, these credentials do not coincide with their MCP counterparts and are not handled in the same way. For example, in the PC environment, credentials can be provided as part of the Connect Network Drive operation or as a member of a trusted domain. Or, credentials can be omitted, resulting in guest access to a share that has been defined for guest access. For resources like shared printers, omitting credentials and permitting guest access to all users is adequate.

It is possible to use the IOHSTRING CREDENTIALS keyword to supply the user name and password for the network resource. However, these credentials are stored and passed in clear text and can be discovered by unauthorized persons. As a result, the REDIRSUPPORT IOHANDLER supports an encrypted mechanism for specifying the credentials to be used when accessing the remote system.

MAKECREDENTIALS Utility

The MAKECREDENTIALS utility provides a method of encrypting credentials, eliminating the need to specify explicit credentials as part of the IOHSTRING attribute, and creates a credentials file for each system for which the user has remote access.

MAKECREDENTIALS is a simple command mode utility that expects the following login credentials for a network host:

- <host>
- <username>
- <password>
- <user domain> (optional)

These four pieces of information correspond to the IOHSTRING keywords as follows:

- SERVER = <host>
- CREDENTIALS = <username>/ <password>
- USERDOMAIN = <user domain>

Examples

The following example shows how the MAKECREDENTIALS utility is run from the CANDE terminal emulator for a host of TR-PRESTIJC, a username of JCP, and a password of FROG:

```
U *SYSTEM/NXSERVICES/MAKECREDENTIALS TR-PRESTIJC JCP FROG
```

As with the Uniform Naming Convention, the <host> portion can be specified as a domain name, an IP address, or a NETBIOS name. Modifying the previous example to use an IP address results in

```
U *SYSTEM/NXSERVICES/MAKECREDENTIALS 192.63.228.29 JCP FROG
```

The file created by MAKECREDENTIALS is named using the <host> portion of the command information.

NXSERVICES CREDENTIALS Files

The MAKECREDENTIALS utility creates credentials files with the usercode and on the primary family of the MAKECREDENTIALS task. The credentials file has the following naming convention:

```
NXSERVICES/CREDENTIALS/<host>
```

The <host> portion of the name is derived from the <host> parameter to the MAKECREDENTIALS utility. When the host is an IP address or domain name, the embedded periods are replaced by the underscore (_) character. For the previous examples described in "MAKECREDENTIALS Utility," the credentials files are created as follows:

```
CREDENTIALS/TR-PRESTIJC

CREDENTIALS/192_63_229_28
```

When the MAKECREDENTIALS task is run without a usercode, the file is created as a global file named *CREDENTIALS/<host>. Whether the credentials file is created with or without a usercode is important, as credentials files are useful only if accessed during a file open routine under the appropriate usercode.

The credentials file contains the following information:

- MCP usercode of the MAKECREDENTIALS task when the file was created
- Username on the remote system
- Password on the remote system
- User log-on domain name (optional)

When you want to open a file on a remote server, you can determine which credentials are used in the following manner:

- If CREDENTIALS is explicitly specified as part of the IOHSTRING attribute, then it is used.
- If CREDENTIALS is not explicitly specified, the usercode and family associated with the process that declared the logical file is used. This convention preserves the declarer model when locating the appropriate credentials file. If the file is found, the information is extracted.

If the MCP usercode stored as part of the credentials file does not match the usercode of the credentials file itself, the file is not used and an open error is generated. The MCP usercode is stored and compared with the MCP usercode of the physical credentials file, preventing one user from using a credentials file created by another user.

A global credentials file, such as that created by an unusercoded run of MAKECREDENTIALS, is supported and contains a special usercode to match against the * usercode of the global credentials file. This special usercode prevents a credentials file created by a specific user from being installed and used as a global credentials file or a global credentials file from being used as the credentials file for a specific user.

Global credentials permit

- Unusercoded tasks to access a remote system using the credentials file mechanism
- A default credentials file to be used for a set of users

The normal MCP file security rules govern whether the credentials file can be opened. In the first case, the unusercoded task is the owner of the file and access is granted based upon the owner's privileges. In the second case, the security state of the file governs whether a particular user can open the file, as the file could be PUBLIC or GUARDED. In either case, a usercoded task first attempts to locate a credentials file under the appropriate usercode. If the file is not found, the search looks for a global file, using the normal MCP file search semantics.

The REDIRSUPPORT IOHANDLER library permits the use of credentials files with printer shares as described in *Installing a Printer for Print System Use*.

If an optional user domain value is specified in the credentials file, this value is used when establishing the user login. The user domain name corresponds to the Domain field on the Windows NT Logon screen.

If the user domain is specified in both the credentials file and in the IOHSTRING attribute, the value specified in the credentials file takes precedence. If the credentials file is being used and the file contains a user domain, the value from the credentials file overrides the value specified in the IOHSTRING attribute.

REDIRSUPPORT Considerations for Use

The following paragraphs describe how to access files from an MCP program using the REDIRSUPPORT IOHANDLER library, including information about establishing the connection through a file open operation, resolving problems, and buffer size considerations that can affect performance.

Networking Considerations

The REDIRSUPPORT IOHANDLER uses the Server Message Block (SMB) protocol to redirect file handling requests to SMB servers in a Microsoft networking environment. The SMB servers are located using various addressing and domain name services as described in the *TCP/IP Distributed Systems Services Operations Guide* and the *TCP/IP Implementation and Operations Guide*. These manuals describe the underpinnings of the networking environment; however, this information is not required to use the REDIRSUPPORT IOHANDLER library.

To establish a connection to an SMB server, the REDIRSUPPORT IOHANDLER ultimately opens a PORT file with the TCP/IP and requests a NetBIOS session. To locate the server when opening the port, TCP/IP requires an IP address. The required NetBIOS name and IP address are determined using information provided to the REDIRSUPPPORT IOHANDLER through file attribute specification. Because you can specify the required information in several ways, the REDIRSUPPORT IOHANDLER follows conventions when evaluating the information provided at file open time.

First, the NetBIOS name is determined from the server or domain name as described in the examples below. Second, the IP address is either explicitly set in the file attributes, or is discovered using name services from WINS with the NetBIOS name or from DNS (provided by RESOLVER) using the domain name. For WINS name resolution to be used, the server name must be a simple identifier, and neither a domain name nor an IP address may be specified.

Examples

Consider a file FYL in the share PUB residing on the server CCK. The IP address of the server is 195.56.10.24 and the full domain name is CCK.CHADDS.FORD.COM. The addressing information for the SMB server can be provided to the REDIRSUPPORT IOHANDLER as either a domain name or an IP address. The address can be derived explicitly when set with the DOMAINNAME or IPADDRESS keywords in the IOHSTRING file attribute or by specifying the full domain name in the <host> portion of a *UNC name as follows:

```
FILE VF (FILENAME = "*UNC/CCK/PUB/FYL.",

IOHSTRING = ""IP=195.56.10.24""".",
```

or

```
IOHSTRING = ""DOMAINNAME=CCK.CHADDS.FORD.COM""".",

FILE VF(FILENAME = "*UNC/""""CCK.CHADDS.FORD.COM""""/PUB/FYL."
```

The address can also be derived utilizing the WINS and/or DNS name services when an IP address or domain name is not explicitly set as in the previous example. In this case, the name from the SERVERNAME keyword in the IOHSTRING file attribute or from the <host> portion of the *UNC name is treated as the NetBIOS name of the server. This name is first passed to WINS to find the IP address. If a value is returned, it is applied to the port file. If the name cannot be resolved, the port domain name is set as a partial domain name based on the server name.

In each of the following examples, the NetBIOS name is set to CCK. The name CCK is then passed to WINS for resolution. If an IP address is returned, it is set on the port file. If the name cannot be resolved, the port file domain name is set to CCK.

```
FILE VF(IOHSTRING= "SERVER=CCK SHARE=PUB");

FILE VF(FILENAME = "*UNC/CCK/PUB/FYL.");
```

Finally, if the NetBIOS name cannot be determined from the SERVERNAME keyword of the IOHSTRING attribute, or the <host> node of the *UNC name, and a full domain name has been specified, the NetBIOS name is implicitly set with the first node of the domain name.

```
FILE VF(FILENAME = "*UNC/""""CCK.CJADDS.FORD.COM""""/PUB/FYL.");
```

The NETBIOS name is explicitly set with either the SERVERNAME keyword of the IOHSTRING attribute or the <host> node of the *UNC name, or is implicitly set with the first node of an explicit domain name. In each of the previous examples, the NETBIOS name is set to CCK.

The default buffer size specified on the REDIRSUPPORT IOHANDLER port file is 16K bytes. When the default 16K setting is not optimal, you can use the BUFFERSIZE file attribute for the virtual file to specify a larger buffer size. The SMB server, however, might negotiate the new buffer size to a smaller value. The program must interrogate the attribute after the file is opened to see the actual size that is in effect.

Declaring a Network File

The file structure for network files accessed with the REDIRSUPPORT IOHANDLER library corresponds to an ASCII byte stream file accessed with the ANYSIZEIO attribute set to TRUE. The following paragraphs describe how to create and access network files with redirected file declarations.

Creating a File

The following example shows a redirected file declaration for creating a network file:

```
FILE VFOUT (REDIRECTION,

            FILENAME = "*UNC/TR-PRESTIJC/PUBLIC/"""JIM.TXT""","

            NEWFILE = TRUE,

            FILEUSE = OUT,

            FRAMESIZE = 8,

            MAXRECSIZE = 1,

            ANYSIZEIO = TRUE,

            INTMODE = EBCDIC,

            EXTMODE = ASCII);
```

The attributes indicate that the file JIM.TXT is to be created in the PUBLIC share on the computer named TR-PRESTIJC (FILENAME) using a virtual file with the SL function REDIRSUPPORT (REDIRECTION). The program using VFOUT can only write data to the file (FILEUSE) and the data written is translated from EBCDIC to ASCII (INTMODE, EXTMODE). The units for an I/O request are 8-bit characters (FRAMESIZE) and any number of characters can be transferred on a single WRITE (ANYSIZEIO) request.

As in the example, programs using byte-stream files generally set the ANYSIZEIO attribute to TRUE, enabling I/O requests of multiple records (characters) to be performed. If ANYSIZEIO is not set to TRUE, I/O requests are for a single character at a time. Any request for more than one character is truncated to one character by the MCP. Due to these restrictions, it is recommended that ANYSIZEIO be set to TRUE when using the REDIRSUPPORT library.

By convention, the FILESTRUCTURE attribute of redirected files is set to STREAM and is not needed in the declaration. The BLOCKSTRUCTURE attribute is not specified and defaults to the required FIXED value. Similarly, the FILEORGANIZATION attribute is not specified and defaults to the required NOTRESTRICTED value.

Accessing an Existing File

As with most MCP files, the created file can be accessed for input with a corresponding file declaration such as

```
FILE VFIN  (REDIRECTION,

            FILENAME = "*UNC/TR-PRESTIJC/PUBLIC/"""JIM.TXT""","

            NEWFILE = FALSE,

            FILEUSE = IN,

            DEPENDENTSPECS = TRUE,

            ANYSIZEIO = TRUE,

            INTMODE = EBCDIC);
```

The file that was created from the VFOUT example is declared for input. The DEPENDENTSPECS attribute indicates that the FRAMESIZE, MAXRECSIZE, BLOCKSTRUCTURE, FILESTRUCTURE, FILEORGANIZATION, and EXTMODE attributes are to be assigned by the MCP from values returned by the REDIRSUPPORT IOHANDLER after a successful open routine. The values of these attributes correspond to those in the VFOUT example.

If used, the BLOCKSTRUCTURE and FILEORGANIZATION values must be set as in the preceding example. These two values are the default values, however, and need not be explicitly set. The FILESTRUCTURE attribute can be any valid value as it is ignored by the REDIRSUPPORT IOHANDLER.

The default EXTMODE value for files supported by the REDIRSUPPORT IOHANDLER library is ASCII, representing the EXTMODE value of the physical file. As for files with other values of the KIND attribute, the EXTMODE value of the logical file is set to the EXTMODE value of the physical file (ASCII) when the file is opened. The OVERRIDEEXTMODE attribute can be used to modify the EXTMODE value of the virtual file, as long as the new EXTMODE value represents an 8-bit or a 16-bit character.

When the INTMODE value of the virtual file does not match the EXTMODE value, translation is attempted as described in “Dealing with Translation” in Section 2, “Understanding Programming for Files.” To avoid translation, you can set the file attribute DEPENDENTINTMODE. Alternatively, you can set the INTMODE attribute explicitly to ASCII without overriding the ASCII mode returned from the REDIRSUPPORT IOHANDLER.

File Attribute Considerations

The following paragraphs describe how the REDIRSUPPORT IOHANDLER handles supported attributes while a file is opened and assigned. For general information on any of these attributes, refer to the *File Attributes Reference Manual*.

ACCESSDATE, ACCESSTIME, ALTERDATE, ALTERTIME, CREATIONDATE, and CREATIONTIME

These read-only timestamp attributes are returned from the SMB server and correspond to the file properties shown in various PC graphics displays.

ALTERNATEGROUPS, GROUP, OWNER, and SECURITYGUARD

These attributes are ignored by the REDIRSUPPORT IOHANDLER when assigned and return null strings (".") upon interrogation.

CCSVERSION, EXTDELIMITER, IOHSTRING, and NOTE

These attributes are managed for the REDIRSUPPORT IOHANDLER in the same manner as FILEKIND. That is, the virtual file values are saved, modified, and returned from the virtual file state maintained by the REDIRSUPPORT IOHANDLER. Note that the CCSVERSION attribute cannot be modified when assigned.

These attributes are not permanent network attributes. If they are modified on the open virtual file, the values are not stored in the FIB and are lost at file close.

FILEKIND

The FILEKIND value of a virtual file is saved during the file open routine in the virtual file state maintained for the REDIRSUPPORT IOHANDLER and is returned when interrogated. FILEKIND is not a permanent network attribute. If the FILEKIND value is modified on the open virtual file, the new value is not set in the FIB and is lost when the file is closed. In support of directory operation, attempting to change the FILEKIND value to or from PERMDIR for an open redirected file is not allowed.

FILENAME, LFILENAME, LTITLE, PATHNAME, and TITLE

Interrogation of the various file name attributes is handled by Logical I/O when returning the value from the FIB. Assignment of the various FILENAME attributes is supported through calls to the REDIRSUPPORT IOHANDLER library, with a successful result causing the name to be stored in the FIB for future interrogation.

Assignment of the file name to an open file, as in renaming the file, is somewhat problematic for the REDIRSUPPORT IOHANDLER. The network protocol requires that the file be closed before renaming. That is, the file must be closed, renamed, and reopened under the new name. This closing and reopening of a file is inconsistent with the concept of exclusive file use, in that once the file is closed for renaming, another process might be able to open the file, causing the subsequent renaming or EXCLUSIVE reopen operation to fail. Since the opening of the file with the EXCLUSIVE attribute precedes the requested name change, an attempt to rename the exclusively opened file fails with an attribute error.

Any attempt to change the name of a file to one in which either the share or server differ from the original file name is not allowed. A file cannot be moved from one network share to another by changing its file name attribute.

While each of the preceding file naming attributes can be used to name a virtual file, the use of an ON <family> part in the title is restricted. In fact, as with any other file KIND, the specification of a family for a file causes the KIND attribute to be set to DISK.

FILELENGTH, LASTRECORD

The attributes LASTRECORD and FILELENGTH are read-only on an opened assigned file, but the values must be returned from the REDIRSUPPORT IOHANDLER to get an accurate value for the current size of the file. When these attributes are interrogated, the REDIRSUPPORT IOHANDLER retrieves the values from the SMB server.

SECURITYMODE, SECURITYTYPE

Because MCP security differs from the network security model, these attributes do not directly map from the MCP to the SMB server and are managed as follows:

- For interrogation, SECURITYTYPE always returns VALUE (PRIVATE) and SECURITYMODE always returns 3"700", which is a security mode mask that corresponds to a SECURITYTYPE value of PRIVATE. Programs do not experience spurious attribute errors when interrogating security. The value used when an application propagates the security value from one file to a new file results in the creation of a private file.
- Assignment of the SECURITYTYPE and SECURITYMODE attributes is not supported and fails without an error being generated.

SECURITYUSE

SECURITYUSE is supported for both interrogation and assignment as follows:

- When interrogated, the REDIRSUPPORT IOHANDLER returns a value that is consistent with the DOS read-only file attribute. If the target file is marked as a read-only file, then a value of IN is returned; otherwise, the value of IO is returned.
- On assignment, a value of IN causes the target file's read-only file attribute to be set. Similarly, a value of IO causes the target file's read-only file attribute to be reset. Any other value results in an attribute error.

SYNCHRONIZE

The SYNCRONIZE attribute is used to cause the REDIRSUPPORT IOHANDLER to flush all WRITE requests to the server.

USERINFO

The USERINFO attribute is used to both interrogate and modify the basic DOS-related attributes. USERINFO has the following format:

```
Readonly flag           = [0:1]

Hidden file flag        = [1:1]

System file flag        = [2:1]

Volume Id Flag          = [3:1]

Directory Flag          = [4:1]

File Changed flag       = [5:1]
```

Note: *The Volume Id and Directory flags cannot be modified. Attempts to change these flags fail without an attribute error.*

Example Program

The following ALGOL program uses the REDIRSUPPORT IOHANDLER library to create a file on a workstation in a Microsoft networking environment and read that file, comparing the data read to the original data written to the file. The program demonstrates the use of the IOHSTRING attribute to specify a domain name when that name differs from the server name contained in the *UNC file name. The program also gives examples of new and existing file handling, attribute interrogation and propagation, and file removal.

```
BEGIN
FILE
    VFOUT (REDIRECTION,
           FILENAME = "*UNC/PRESTIJC/PUBLIC/"""JIM.TXT""".",
           IOHSTRING = ""DOMAIN=JCP.PA.USA.COM""".",
           NEWFILE = TRUE,
           FILEUSE = OUT,
           FRAMESIZE = 8,
           MAXRECSIZE = 1,
           ANYSIZEIO = TRUE,
           INTMODE = EBCDIC,
           EXTMODE = ASCII),

    VFIN  (REDIRECTION,
           NEWFILE = FALSE,
           FILEUSE = IN,
           DEPENDENTSPECS = TRUE,
           ANYSIZEIO = TRUE,
           INTMODE = EBCDIC);

EBCDIC ARRAY
    EA, EB [0:299];
POINTER
    P;
REAL
    LEN1, LEN2;

    OPEN (VFOUT);
    REPLACE P:EA [0] BY "Now is the time for all good men to ",
                        "come to the aid of their country.";
    LEN1 := OFFSET(P);
    WRITE (VFOUT, LEN1, EA);
    REPLACE EB [0] BY VFOUT.FILENAME;
    REPLACE VFIN.FILENAME BY EB [0];
    REPLACE EB [0] BY VFOUT.IOHSTRING;
    REPLACE VFIN.IOHSTRING BY EB [0];
    CLOSE (VFOUT);
    OPEN (VFIN);
    LEN2 := VFIN.FILELENGTH;
    READ (VFIN, LEN2, EB[0]);
    IF LEN1 = LEN2 AND EA [0] = EB [0] FOR LEN2 THEN
        DISPLAY (EA)
```

```
    ELSE
        PROGRAMDUMP (ALL, TODISK);
    CLOSE (VFIN, PURGE);
END.
```

Directory Operations

Basic directory operations are supported by the REDIRSUPPORT IOHANDLER, providing the capability to create, delete, and rename directories in a network share. In addition, the REDIRSUPPORT IOHANDLER provides the capability to read the contents of a directory, that is, to obtain the names of the files and directories along with the attributes immediately subordinate to those directories.

REDIRSUPPORT IOHANDLER Directory Semantics

A program manipulates directories with the REDIRSUPPORT IOHANDLER by assigning the FILEKIND value to the PERMDIR option for the redirected (virtual) file. A directory can then be opened in the same way as any other file with the following semantics:

- If NEWFILE = FALSE and the directory exists, a good open result is returned to the program.
- If NEWFILE = TRUE, the directory is created. If the directory already exists, an open error is returned.

Once a directory has been successfully opened the following actions are possible:

- Changing the file name through the LFILENAME/FILENAME attribute of the virtual file causes the directory to be renamed.
- Closing the virtual file with a PURGE operation, such as CLOSE(PURGE), causes the directory to be deleted.
- Reading the file returns the directory in a format described later in this section.

Attempting to change the FILEKIND value to or from PERMDIR for an open redirected file is not allowed. In addition, an error results if a WRITE request is attempted on a redirected file with the PERMDIR option set.

Reading a Directory

After a directory has been opened, the READ intrinsic can be used to read the contents of the directory. Reading the directory causes the names of the directory entries along with their attributes to be returned in a structured format suitable for programmatic processing. Because the information returned contains a mixture of binary and character data, you should open the file with DEPENDENTINTMODE to disable any unwanted translation.

The reading of directory data using the REDIRSUPPORT IOHANDLER follows the byte stream mechanism used for other network files. The amount of data requested by the virtual file program is returned to the program as a stream of bytes. Access to the directory information must be in a serial fashion. The format of the data returned is such that each entry starts at a word boundary and contains fixed attribute information followed by the variable length name of the directory stub. This format is shown in Table 30–2, where NEXTIX is the word offset in the DIRINFO array of a directory entry.

Table 30–2. Returned Format of Directory Entries

Entry Word/Field	Description
DIRINFO [NEXTIX].[47:16]	Size of entry in words
DIRINFO [NEXTIX].[31:32]	Reserved
DIRINFO [NEXTIX+1].[47:16]	DOS file attributes
DIRINFO [NEXTIX+1].[31:32]	File length in bytes
DIRINFO [NEXTIX+2]	Creation date (YYYDDD)
DIRINFO [NEXTIX+3]	Creation time (time of day in tics)
DIRINFO [NEXTIX+4]	Access date
DIRINFO [NEXTIX+5]	Access time
DIRINFO [NEXTIX+6]	Modify date
DIRINFO [NEXTIX+7]	Modify time
DIRINFO [NEXTIX+8].[47:8]	Length of directory node
POINTER(DIRINFO [NEXTIX+8])+1	Null terminated EBCDIC name

Directory Programming Example

The following program illustrates how the directory information returned is processed when a REDIRSUPPORT IOHANDLER directory file is read. The program reads the directory named in the FILENAME for F and displays the contents on the remote file.

```
BEGIN
FILE RMT (KIND=REMOTE, UNITS=CHARACTERS, MYUSE=IO),

     F   (REDIRECTION,
          FILEKIND=PERMDIR,
          NEWFILE=FALSE,
          DEPENDENTSPECS,
          DEPENDENTINTMODE,
          FILEUSE=IN,
          ANYSIZEIO,
          FILENAME="*UNC/CCK2/SHARED.");

REAL    REC,LEN,RD,NAMES,TOTALSYZE,LINES;
BOOLEAN BRD = RD;
ARRAY   A [0:9999];
EBCDIC ARRAY EA [0] = A, EDISP[0:299];
POINTER P;
TRUTHSET LETTERS ("ABCDEFGHIJKLMNOPQRSTUVWXYZ"),
         NUMBERS ("0123456789"),
         LETTERSORNUMBERS (LETTERS OR NUMBERS),
         VALIDCHRS (LETTERSORNUMBERS OR "_-");

PROCEDURE INSERTDATE (P, DATE);
VALUE P, DATE;
POINTER P;
REAL DATE;
BEGIN
LABEL AWAY;
REAL M,D,Y,T;
     Y := DATE DIV 1000;
     D := DATE MOD 1000;
     FOR T := 0, 31,
               IF (Y MOD 4)=0 AND ((Y MOD 400)=0 OR (Y MOD 100) NEQ 0)
               THEN 29 ELSE 28,
               31, 30, 31, 30, 31, 31, 30, 31, 30 DO
          IF D GTR T THEN
          BEGIN
               D := D - T;
               M := M + 1;
          END ELSE GO AWAY;
AWAY:
     REPLACE P:P BY M FOR 2 NUMERIC, "/",
                    D FOR 2 DIGITS,  "/",
                    Y FOR 2 DIGITS;
END INSERTDATE;
```

```
PROCEDURE INSERTTIME (P, TYME);
VALUE P, TYME;
POINTER P;
REAL TYME;
BEGIN
IF ABS(TYME := TYME*2.4@-6) > 4"007FFFFFFFFF" THEN
  REPLACE P:P BY "Integer Overflow"
ELSE
  BEGIN
  TYME := INTEGERT(TYME); % SECONDS
  REPLACE P:P BY TYME DIV 3600        FOR 2 DIGITS, ":",
                 TYME MOD 3600 DIV 60 FOR 2 DIGITS, ":",
                 TYME          MOD 60 FOR 2 DIGITS;
  END;
END INSERTTIME;

PROCEDURE DISPDIR(EA,L);
  REAL  L;
  EBCDIC ARRAY EA[0];
BEGIN
  ARRAY REFERENCE A[0];
  REAL CHRS, IDCHRS, ATTR, CDATE, CTIME, ADATE, ATIME, MDATE, MTIME,
       SYZE, IX, QUOTED;
  BOOLEAN DONE;
  POINTER P;
  A := EA;
  WHILE NOT DONE DO
    BEGIN
      CHRS   := A[IX].[47:16];
      IF NOT DONE := CHRS = 0 OR IX*6 + CHRS > L THEN
      BEGIN
        ATTR   := A[IX+1].[47:16];
        SYZE   := A[IX+1].[31:32];
        CDATE  := A[IX+2];
        CTIME  := A[IX+3];
        ADATE  := A[IX+4];
        ATIME  := A[IX+5];
        MDATE  := A[IX+6];
        MTIME  := A[IX+7];
        IDCHRS := A[IX+8].[47:8];
        P := POINTER(A[IX+8]) + 1;
        IF NOT P IN VALIDCHRS FOR 1 THEN
          QUOTED := 1
        ELSE
          BEGIN
            SCAN P FOR QUOTED:IDCHRS WHILE IN VALIDCHRS;
            QUOTED := MIN(QUOTED,1);
          END;
        REPLACE EDISP[0] BY " " FOR 132;
        REPLACE EDISP[0] BY """ FOR QUOTED,
                            P FOR IDCHRS,
                            """ FOR QUOTED;
```

```
        REPLACE P:EDISP[IDCHRS+2*QUOTED] BY " : ";
        IF ATTR.[4:1] = 1 THEN
          REPLACE P:P BY "Directory"
        ELSE
          BEGIN
            REPLACE P:P BY "DATA";
            INSERTDATE (EDISP[30],CDATE);
            REPLACE EDISP [39] BY "@";
            INSERTTIME (EDISP[41],CTIME);
            INSERTDATE (EDISP[52],ADATE);
            REPLACE P:EDISP[64] BY SYZE FOR * NUMERIC, " bytes";
          END;
        WRITE(RMT,OFFSET(P),EDISP);
        IX := * + (CHRS DIV 6);
        NAMES  := * + 1;
        TOTALSYZE := * + SYZE;
        IF NAMES MOD LINES = 0 THEN READ(RMT);
        DONE := IX*6 GEQ L;
      END;
    END;
  L := IX*6;
END DISPDIR;
OPEN(RMT);
LINES := RMT(1).SCREENSIZE - 1;
IF F.AVAILABLE = VALUE(OKRSLT) THEN
BEGIN
    WHILE NOT BRD := READ(F[REC],2000,EA) DO
    BEGIN
        LEN := RD.[47:20];
        DISPDIR(EA,LEN);
        REC := * + LEN;
    END;
    REPLACE P:EDISP[0] BY "Total Names = ",
                          NAMES for * DIGITS;
    WRITE(RMT,OFFSET(P),EDISP);
    REPLACE P:EDISP[0] BY "Total Bytes = ",
                          TOTALSYZE FOR * DIGITS;
    WRITE(RMT,OFFSET(P),EDISP);
END;
END.
```

Section 31
Using the STREAMIOH IOHANDLER Library

The STREAMIOH IOHANDLER library allows ALGOL, NEWP, COBOL, and other primarily record-oriented programs to create and access delimited character-stream files as record files. The access is provided by file-equating the record file to a virtual file and using the IOHSTRING attribute to pass parameters to the STREAMIOH IOHANDLER library.

Declaring the Record File to Use STREAMIOH IOHANDLER

The delimited lines in a character-stream file are converted to fixed-length records and fixed-length records are converted to the delimited lines by the STREAMIOH IOHANDLER library. Because the STREAMIOH IOHANDLER library is a virtual file library, the program logical file must have KIND=VIRTUAL specified, and an IOHFUNCTIONNAME attribute or IOHTITLE attribute indicating the linkage mechanism to the library.

Examples

```
FILE RECF  (KIND = VIRTUAL
            IOHFUNCTIONNAME = "STREAMIOH.",...);

FILE RECF  (KIND = VIRTUAL
            IOHTITLE = "*SYSTEM/STREAMIOH.",...);
```

In naming the physical delimited file, one of the various forms of the TITLE attribute might be used. However, when specifying a DISK file title including an ON part, the FAMILYNAME attribute must be specified separately. Otherwise, the KIND=VIRTUAL file equation is overridden and forced to KIND=DISK.

For example, the file equation for the existing file SOMEFILE on the FAMILY "HI" would be as follows:

```
FILE RECF  (KIND = VIRTUAL
            IOHFUNCTIONNAME = "STREAMIOH."
            NEWFILE = FALSE
            DEPENDTSPECS = TRUE
            FILENAME = "SOMEFILE."
            FAMILYNAME = "HI."
```

For more information on virtual files, refer to Section 29, "Understanding Virtual Files."

Terminology Definitions

The following terms are used throughout this section:

- Character-stream file

  A file consisting of a sequence of characters. Character-stream disk and CD-ROM files can be addressed randomly at the character level. A byte-stream file is a character-stream file in which the character size is 8 bits.

- Conversion

  The process of presenting a character-stream physical file to a program as a record file.

- Delimited file

  A character-stream file that is divided into variable-length records or lines by delimiter characters.

- Delimiter or delimiter character

  A character or characters that mark the end of records in delimited files. The most common delimiters are

UNIX	new line (NL)
DOS/Windows	carriage return followed by line feed (CRLF)
Macintosh	carriage return (CR)

  The characters are generally encoded in ASCII. The UNIX new line character is the same as the ASCII line-feed character.

- Record file

  A file consisting of a sequence of fixed- or variable-length records. Most existing MCP files are record files.

- Record-oriented program

  A program that expects to deal only with record files. Traditionally, MCP-based programs have been record oriented.

FILEKIND and File Extension Handling

In the MCP environment, default record formats have been defined based on the FILEKIND value of the file. These record formats define the default record size for the FILEKIND as well as TEXT, SEQUENCE, and MARKID offsets and lengths in the record.

The STREAMIOH IOHANDLER library uses a FILEKIND specified for the record file to perform a conversion from the physical delimited file. The FILEKIND might be explicitly equated on the record file, included in the parameters for the virtual file, or implied from a PC-type file extension. The delimited file conversion is performed based on the defaults for the FILEKIND determined for the file. That is, once a FILEKIND value is determined, the TEXT, SEQUENCE, and MARKID fields are predetermined. Extending the example from the previous topic, the file equation is modified as follows:

```
FILE F     (KIND = VIRTUAL
            IOHFUNCTIONNAME = "STREAMIOH."
            NEWFILE = FALSE
            DEPENDENTSPECS = TRUE
            FILENAME = "SOMEFILE.TXT."
            FAMILYNAME = "HI.");
```

The FILEKIND is omitted since the .TXT extension implies that the expected FILEKIND is TEXTDATA.

The file extensions recognized by STREAMIOH IOHANDLER library are the same as those defined for the Editor and Programmer's Workbench products. However, for Programmer's Workbench files, the extension has an _M added to the extension. For example, while Editor recognizes .ALG as an ALGOLSYMBOL file extension, Programmer's Workbench recognizes .ALG_M.

Table 31–1 lists base extensions and their associated FILEKIND values.

Table 31–1. File Extensions Recognized by STREAMIOH IOHANDLER Library

Base Extension	FILEKIND Value
ALG	ALGOLSYMBOL
BAS	BASICSYMBOL
BND	BINDERSYMBOL
C	CCSYMBOL
H	CCSYMBOL
CCC	CCSYMBOL
CDT	CDATA
COB	COBOLSYMBOL
C74	COBOL74SYMBOL

Table 31–1. File Extensions Recognized by STREAMIOH IOHANDLER Library

Base Extension	FILEKIND Value
C85	COBOL84SYMBOL
CPP	CPPSYMBOL
CSD	CSEQDATA
DAS	DASDLSYMBOL
DAT	DATA
DCA	DCALGOLSYMBOL
DMA	DMALGOLSYMBOL
FOR	FORTRANSYMBOL
F77	FORTRAN77SYMBOL
JAV	JAVASYMBOL
JAVA	JAVASYMBOL
NWP	NEWPSYMBOL
PAS	PASCALSYMBOL
P83	PASCAL83SYMBOL
RPG	RPGSYMBOL
ATX	TEXTDATA
TXT	TEXTDATA
SEQ	SEQDATA
WFL	JOBSYMBOL

Programmer's Workbench maintains files in the MCP record file format within the delimited character-stream file. That is, the delimited character-stream file contains fixed-length records and includes the SEQUENCE and MARKID fields as appropriate for the FILEKIND. This format corresponds to parameters of TRIM = NONE and FOLDING = ID, as described further under "STREAMIOH Parameters" later in this section.

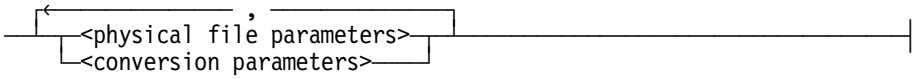
STREAMIOH Parameters

The STREAMIOH parameters are specified for the IOHSTRING parameter of the virtual file. The IOHSTRING pointer-valued parameter is passed to the virtual file IOHANDLER when the parameter is set, or it is retrieved from the IOHANDLER virtual file when the parameter is interrogated. As a pointer-valued parameter, the parameter is assigned using string or pointer operands.

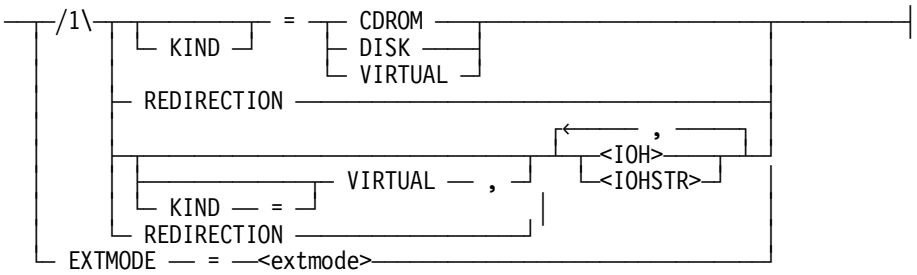
The allowable IOHSTRING parameters fall into two basic categories. The first category provides the STREAMIOH with information regarding the location and characteristics of the physical file, while the second category controls the conversion of delimited lines into logical records and logical records into delimited lines. The remainder of this topic describes the IOHSTRING parameters and their semantics.

Syntax

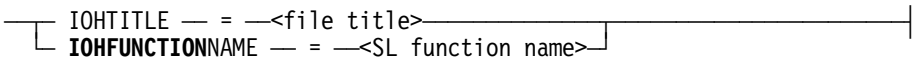
<IOHSTRING parameters>

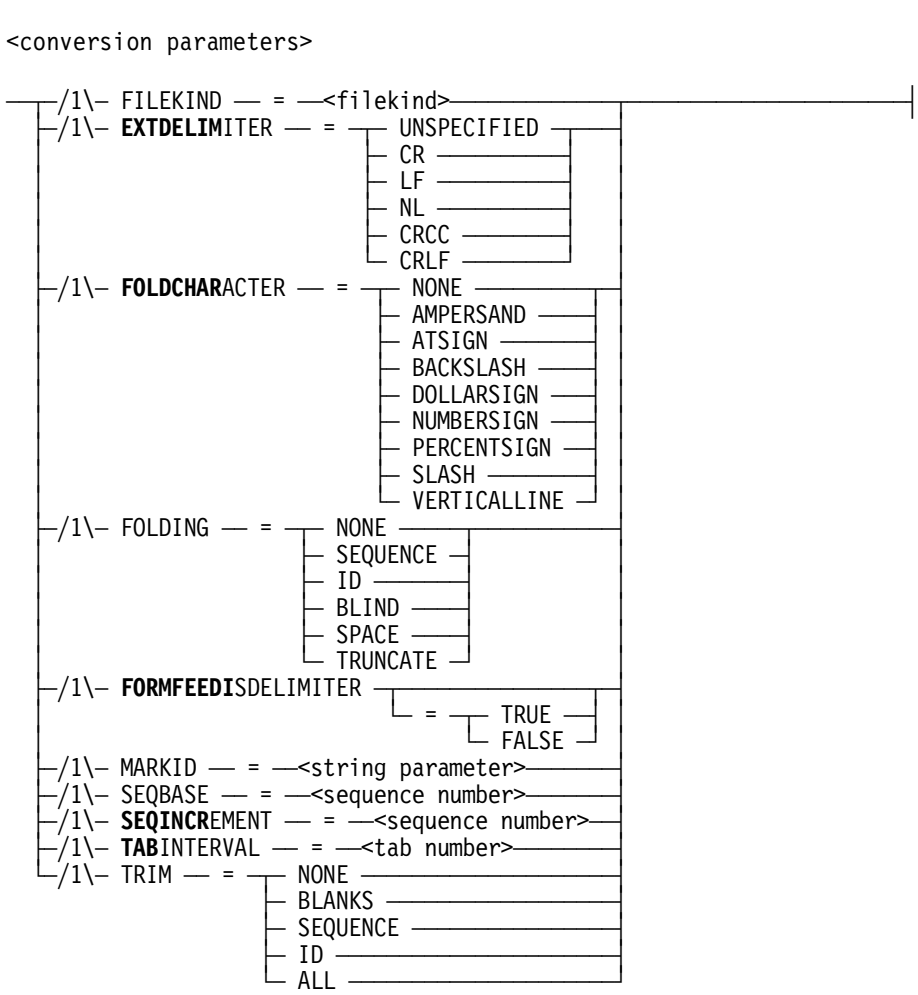


<physical file parameters>



<IOH>





Explanation

- A sequence number is a non-negative integer between 0 and 99999999.
- The string parameter (n) is a string of *n* or fewer characters surrounded by quotation marks. This convention makes the input string simpler to enter and simpler for the STREAMIOH IOHANDLER library to handle.
- The tab number is a non-negative value that does not exceed the length of the record ultimately set for the file.

Parameter Semantics

When the logical record file is declared as a virtual file with an IOHANDLER of STREAMIOH and is opened, the STREAMIOH is invoked and requested to open the physical character-stream file. The STREAMIOH has to locate the file and determine the following characteristics of the underlying delimited file:

- The KIND attribute of the delimited file; for example, DISK or CDROM
- The IOH attributes for the delimited file if the KIND is VIRTUAL; for example, IOHFUNCTIONNAME
- The FILEKIND attribute of the delimited file

Once the physical file is located and the FILEKIND is determined, the remaining parameters are used to control the conversion process.

Physical File Parameters

KIND

The physical file is located using the FILENAME and FAMILYNAME attributes of the logical file. However, because the KIND attribute of the logical file is set to VIRTUAL, it cannot be used to indicate the KIND attribute of the delimited file. Therefore, the KIND information for the delimited file might optionally be provided in the IOHSTRING parameter.

By default, the STREAMIOH assumes that the KIND attribute of the delimited file is DISK, and uses the FAMILYNAME attribute specified for the logical record file to locate the delimited file. When the file does not reside locally on DISK, the IOHSTRING parameters associated with KIND must be used, as in the following ALGOL examples.

For CD-ROM files:

```
FILE RECF  (KIND = VIRTUAL
            IOHFUNCTIONNAME = "STREAMIOH."
            FILENAME = "CDFILE."
            FAMILYNAME = "SOMECD."
            IOHSTRING = "KIND = CDROM.",...);
```

If the file resides on a network share accessed using the MCP REDIRECTOR, the delimited file must be specified to use the REDIRECTION attribute (that is, REDIRECTION implies KIND=VIRTUAL, IOHFUNCTIONNAME= "REDIRSUPPORT"). Therefore, the IOHSTRING parameter of the record file should indicate REDIRECTION. In addition, because REDIRSUPPORT might require IOHSTRING parameters, these might also be passed by way of the parameters to the record file. To help clarify this arrangement, the record file parameters are passed at run time by setting the IOHSTRING attribute in an attribute assignment statement (REPLACE). Quotation marks are used to delimit the internal IOHSTRING passed to REDIRSUPPORT.

```
FILE RECF  (KIND = VIRTUAL
            IOHFUNCTIONNAME = "STREAMIOH."
            FILENAME = "PCFILE.");

REPLACE RECF.IOHSTRING BY

    REDIRECTION,
    IOHSTRING = "SERVER = APC, SHARE = ASHARE".;
```

Similarly, if the physical file is to be accessed using another IOHANDLER, the KIND=VIRTUAL and IOHFUNCTION or IOHTITLE parameters are used in place of REDIRECTION:

```
 REPLACE RECF.IOHSTRING BY

    VIRTUAL,
    IOHFUNCTION = "SOMEIOHANDLER.";
```

EXTMODE

The STREAMIOH operates using the EBCDIC character set. The logical virtual file cannot override this setting. The EXTMODE parameter is provided to allow the program to override the physical file EXTMODE. See the *File Attributes Reference Manual* for the list of valid EXTMODE values.

Conversion Parameters

FILEKIND

A FILEKIND parameter might be specified on the record file IOHSTRING to override a default value implied by a file extension, or in lieu of specifying the FILEKIND on the logical record file. The FILEKIND is key to the conversion process in that it is used to determine the record size and format of the record file.

When the FILEKIND parameter is specified, it overrides both the file-equated value and the file extension. When the parameter is not set, a FILEKIND explicitly set for the file overrides the file extension.

The MCP FILEKIND values are documented in the *File Attributes Reference Manual*.

The default value of FILEKIND is SEQDATA.

EXTDELIMITER

The EXTDELIMITER mnemonic parameter identifies the delimiter character, if any, that separate records in the delimited file. The allowed values are

Value	Hexadecimal Representation
UNSPECIFIED	N/A
NL	4825"
CR	48"0D"
LF	48"25"
CRLF	48"0D25"
CRCC	48"0D0C"

When the value is UNSPECIFIED, the FILEKIND is used to determine the record length, and fixed-length records are read from the character-stream file.

The default value of EXTDELIMITER is CRLF.

FOLDCHARACTER

The FOLDCHARACTER mnemonic parameter has the following values and corresponding character representations (Table 31–2).

The default value for FOLDCHARACTER is BACKSLASH.

Table 31–2. FOLDCHARACTER Values and Character Representations

Value	Character
NONE	No fold character
AMPERSAND	&
ATSIGN	@
BACKSLASH	\
DOLLARSIGN	$
NUMBERSIGN	#
PERCENTSIGN	%
SLASH	/
VERTICALLINE	|

The semantics of the FOLDCHARACTER parameter are described with the semantics for the FOLDING parameter under "FOLDING" earlier in this section.

FOLDING

The FOLDING mnemonic parameter controls the handling of delimited file records that exceed the size of the record text area. The size of the text area is a function of the FILEKIND of the record file.

The parameter also controls how unfolding of trailing fold characters is handled during record-to-delimited conversion; for example, when writing a record to the delimited file.

The default value of FOLDING is TRUNCATE.

When you review the FOLDING semantics, remember that

- The TRIM parameter controls the trimming of blank characters from a delimited record before the FOLDING is applied.
- The FOLDING IO values of ID and SEQUENCE indicate that the delimited record contains sequence and mark ID fields in addition to text data—implying that folding is not expected to take place.
- Some values of TRIM and FOLDING are inconsistent and might cause an open error at run time.

Folding during Record-to-Delimited Conversion

During record-to-delimited conversions (read I/O operations), folding might occur depending on the FOLDING parameter:

- NONE

  An I/O error (DATAERROR) is reported if, after any blank trimming, the delimited record exceeds the size of the record file record text area. If the program ignores the I/O error on a read operation, a subsequent sequential I/O operation begins at the character following the next delimiter in the physical file.

- SEQUENCE

  A FOLDING value of SEQUENCE implies that the delimited lines are a fixed-record size and contain sequence number fields. An I/O error (DATAERROR) is reported if the delimited record is not exactly the size of the text and sequence fields of the record file record. If the program ignores the I/O error on a read operation, a subsequent sequential I/O operation begins at the character following the next delimiter in the physical file.

- ID

  A FOLDING value of ID implies that the delimited lines are a fixed-record size and contain sequence number and mark ID fields. An I/O error (DATAERROR) is reported if the delimited record is not exactly the size of the record file record. If the program ignores the I/O error on a read operation, a subsequent sequential I/O operation begins at the character following the next delimiter in the physical file.

- BLIND

  If, after any blank trimming, the delimited record exceeds the size of the record file record text area, the character specified by FOLDCHARACTER, if any, is inserted as the last character in the record text area. A subsequent sequential read operation begins at the character following the last character transferred in the delimited file. Subsequent records might also be folded if the remainder of the folded record exceeds the size of the record text area.

- SPACE

  The handling of a FOLDING value of SPACE is identical to that of BLIND except that instead of inserting the character specified by FOLDCHARACTER as the last character in the text area, the fold character is inserted in place of the last blank character in the text area, and the remainder of the text area is blank.

  If no last blank character exists, or if the data consists solely of blank characters, the handling of SPACE is identical to that of BLIND.

- TRUNCATE

  If, after any blank trimming, the delimited record exceeds the size of the record file record text area, extra data in the delimited record is discarded. After truncation on a read operation, a subsequent sequential I/O operation begins at the character following the next delimiter in the physical file.

Unfolding during Record-to-Delimited Conversion

During record-to-delimited conversion (write I/O operations), unfolding might occur depending on the value of the FOLDING and FOLDCHARACTER parameters. When unfolding occurs, neither the fold character nor a delimiter is inserted into the delimited record, thus in effect reversing a folding operation that was done during delimited-to-record conversion. The actions for each value are described as follows:

- NONE

  Unfolding never occurs.

- SEQUENCE

  Unfolding never occurs.

- ID

  Unfolding never occurs.

- TRUNCATE

  Unfolding never occurs.

- BLIND

  If FOLDCHARACTER is not NONE, and the last character position in the record file record text area contains the fold character, unfolding occurs.

- SPACE

  If FOLDCHARACTER is not NONE, and the last nonblank character in the record file record text area contains the fold character, unfolding occurs.

FORMFEEDISDELIMITER

The FORMFEEDISDELIMITER Boolean parameter, when set, indicates that FORMFEED character (48"OC") is also to be considered as a delimiter. If the EXTDELIMITER value is CRCC, a CR-FF pair in the delimited file is treated as a single delimiter. Otherwise, the CR-FF pair is considered to be two delimiters, and a second blank record is returned for the FF character.

MARKID

The MARKID string parameter is only applicable when the FILEKIND of the record file has a defined ID field. When MARKID is set, the value of the parameter is inserted into the ID field, truncated or blank-filled on the right as necessary. If MARKID is not set, the ID field is blank-filled.

By default, MARKID is set to the null string, and any ID field is blank-filled.

SEQBASE

The SEQBASE integer parameter is only applicable when the FILEKIND of the record file has a defined sequence number field and the SEQINCREMENT parameter is set to a non-zero value. In this case, the record number inserted into the record sequence number of the current record retrieved is determined based on the logical record number in the delimited file. That is, for record N of the file, the sequence number inserted is SEQBASE + (N–1)*SEQINCREMENT. When the value determined exceeds the capacity of the sequence number field, the value is set to 0 (zero).

The default value of SEQBASE is 100.

SEQINCREMENT

The SEQINCREMENT integer parameter is only applicable when the FILEKIND of the record file has a defined sequence number field. If the SEQINCREMENT value exceeds the capacity of the sequence number field, the SEQINCREMENT value is set to 1 when the file is opened. SEQINCREMENT is used as SEQBASE is used. Refer to SEQBASE earlier in this section.

The default value for SEQINCREMENT is 100.

TABINTERVAL

When the TABINTERVAL integer parameter is set to non-zero value, tab characters appearing in the data are replaced by sufficient space characters to advance the destination pointer to a character offset (counting from 1) space—that is, one beyond the next multiple of the tab interval.

For example, given a delimited record starting with the string "ABCDEF<tab>G<tab>H", tab expansion would result in the following:

```
          11111111112
12345678901234567890
ABCDEF    G    H          if TABINTERVAL = 5

ABCDEF  G       H         if TABINTERVAL = 8
```

It is possible for tab expansion to cause the destination pointer to be advanced beyond the end of the destination record. If this happens, folding as specified by the FOLDING parameter occurs.

Tab expansion is a nonreversible operation. Once tab characters have been converted to spaces, no automatic way exists to convert them to tab characters.

The default value of TABINTERVAL is 0 (zero). When the TABINTERVAL parameter is set to the number 0, tab characters appearing in the data are treated like ordinary data characters.

TRIM

TRIM is a mnemonic parameter used to control record trimming, primarily during record-to-delimited conversion (write requests).

Table 31–3 lists the mnemonic values of TRIM and the TRIM semantics. The default value of TRIM is ALL.

Table 31–3. TRIM Mnemonic Values and Semantics

Value	Result
NONE	No trimming
BLANKS	Trim blanks
SEQUENCE	Trim sequence field
ID	Trim mark ID field
ALL	Trim blanks, sequence and mark ID fields

Trimming during Delimited-to-Record Conversion

During delimited-to-record conversion (read I/O operations), if TRIM has a value of NONE, the record is not examined for trimming. For TRIM values other than NONE, if a delimited line exceeds the size of the data portion of the record file record and the excess characters consist solely of blank characters, the excess blanks are discarded instead of folding the line.

Trimming during Record-to-Delimited Conversion

During record-to-delimited conversion (write operations), the value of TRIM indicates the fields that are to be discarded. In addition to causing the specified information to be discarded during conversion, the values BLANKS, SEQUENCE, and ID also cause subsequent, adjacent record fields to be discarded during conversion. For example, for a FILEKIND ALGOLSYMBOL file, specifying SEQUENCE would cause both the sequence number and mark ID fields to be discarded.

File Attribute Considerations

The function of STREAMIOH is to convert character-stream physical files into logical record files, and logical record files into character-stream physical files. Because of the two views of a single file, and the operation of the STREAMIOH on behalf of logical I/O, the handling of the following attributes needs explanation.

FRAMESIZE, UNITS, and Related Attributes

FRAMESIZE and UNITS, along with all attributes that are measured in frame size units, are handled as defined on the logical file.

DEPENDENTSPECS and Related Attributes

If DEPENDENTSPECS is set to TRUE, conversion is based on the FILEKIND of the character-stream file or the IOHSTRING parameter, with the parameter taking precedence. Actual record size and layout is set to match those defined in the MCP FILERECFORMAT procedure for the FILEKIND. If a default record size is not defined for the FILEKIND, a value of 80 characters or 15 words is assigned depending on the UNITS attribute of the file being opened.

If DEPENDENTSPECS is set to FALSE or a file is being created, the FILEKIND of the physical file is set from that of the logical file or from the IOHSTRING parameter, with the parameter taking precedence. The MAXRECSIZE is set from the logical file MAXRECSIZE. If the program MAXRECSIZE value is inconsistent with the FILEKIND record length, records are right-truncated or extended to the right to match the MAXRECSIZE value.

As a virtual file, the FILESTRUCTURE is always ignored. If interrogated, it is reported as STREAM.

BLOCKSTRUCTURE is set to FIXED when DEPENDENTSPECS is TRUE, and allowed to be FIXED or EXTERNAL when DEPENDENTSPECS is FALSE or when creating a character-stream file.

NEXTRECORD and RECORD Attributes

NEXTRECORD and RECORD are reported from the logical record file and do not provide information relative to the character offset in the physical file.

LASTRECORD and FILELENGTH Attributes

LASTRECORD and FILELENGTH are reported based on the logical record size for the FILEKIND of the file. Using these attributes might result in sequentially reading the file to determine these values.

CURRENTRECORDLENGTH Attribute

CURRENTRECORDLENGTH and the length returned in I/O results reflect the size of the converted record, not the physical file delimited record.

I/O Operation Semantics

You should understand the following I/O semantics unique to the STREAMIOH conversion. Additional semantics are described in Section 29.

General Considerations

The following statements clarify various read/write semantics with respect to STREAMIOH:

- Unless unfolding occurs, each user write request causes exactly one delimiter-terminated record to be written to the physical file.
- If a user write request specifies a size value that is less than the size of the logical file record while taking into account any requested trimming, only as much data as was requested is written to the physical file.
- If a user write request specifies a size value that is greater than the size of the logical file record, the request is truncated.
- Unless folding occurs, each user read request causes exactly one record (zero or more characters followed by a delimiter) to be read from the physical file. The delimiter is always discarded.
- If a user read request specifies a size value that is less than the logical file record size, the program receives only as much data as requested. A subsequent sequential read or write operation begins following the next delimiter character in the physical file.
- If a user read request specifies a size value that is greater than the logical file record size, the request is truncated.
- If, during the transfer of data for a user read request, end-of-file is reached before the text portion of the user record has been filled, conversion continues as though the physical file contained sufficient space characters to fill the text area and an error-free result is returned. Assuming the file is not expanded in the interim, a subsequent sequential read operation returns an end-of-file result.

Random Access

Random access is provided by using a record file version of the stream file. Serial access is assumed until a nonserial operation is encountered. At this time, the stream file is read in total to create the record file used for random access. When the logical file is closed, the physical file is rewritten (if necessary) to apply the updates, and the temporary record file is discarded.

Note: *Requesting the LASTRECORD or FILELENGTH file attributes is considered a random access. If such a request is the first random access, it causes the physical file to be read completely and a temporary record file to be created.*

Translation

The STREAMIOH is implemented as an EBCDIC program. Translation of data to EBCDIC as required for the STREAMIOH is performed by logical I/O. Translation does not occur in the STREAMIOH. Specifically, if the EXTMODE of the physical file is not EBCDIC, logical I/O translates the data when read or written from the physical file by STREAMIOH. Similarly, if the INTMODE of the logical file is not EBCDIC, translation occurs between the STREAMIOH and the user program.

If an EXTMODE other than EBCDIC is specified on the virtual file, and OVERRIDEEXTMODE is set to ALWAYS, an open error occurs. To override the physical file EXTMODE, use the EXTMODE parameter on the IOHSTRING. For new file creation, however, the EXTMODE of the virtual file is honored if it differs from EBCDIC and the EXTMODE parameter is not specified on the IOHSTRING parameter.

Example Program

The ALGOL utility program (CONVERTSTREAM) listed in the following example creates a record file from a character-stream file using the STREAMIOH library. The input file, FIN, is a logical record file and a physical delimited character-stream file. The conversion is provided by declaring FIN with KIND = VIRTUAL and IOHFUNCTIONNAME = STREAMIOH. The output file, FOUT, is created using attributes that are the defaults for the FILEKIND that is being created. The parameters to the utility are the STREAMIOH parameters that are applied to the FIN.IOHSTRING attribute. The main loop of the program is simply "WHILE NOTE READ(FIN..) DO WRITE(FOUT..);", as the conversion is done in the IOHANDLER.

An example of a CANDE use of the utility is listed before the program. The FIN file extension of .TXT is used to set the FILEKIND of the output file.

```
RUN CONVERTSTREAM("SEQBASE = 1000 SEQINCR = 100");%
FILE FIN = "TEXTFILE.TXT";%
FILE FOUT = TEXTFILE
#RUNNING 23634
SEQBASE = 1000 SEQINCR = 100
Converting: "TEXTFILE.TXT"
FILEKIND = TEXTDATA, MAXRECSIZE = 90 characters
Creating:   TEXTFILE
```

```
FILEKIND = TEXTDATA, MAXRECSIZE = 15 words
File Locked:TEXTFILE
#ET = 0.8 PT = 0.2 IO = 0.6

PROCEDURE CONVERTSTREAM (INPUT);
ARRAY INPUT[*];
BEGIN
FILE
   FIN (KIND = VIRTUAL,
        IOHFUNCTIONNAME = "STREAMIOH.",
        NEWFILE = FALSE,
        DEPENDENTSPECS,
        FILEUSE = IN),

   FOUT(KIND = DISK,
        NEWFILE = TRUE,
        FILEUSE = OUT),

   RMT (KIND = REMOTE,
        FRAMESIZE = 8,
        MAXRECSIZE = 1920,
        FILEUSE = OUT);

REAL
   N,
   R,
   FIN_FILEKIND,
   FIN_RECSIZE,
   FIN_FRAMESIZE,
   FOUT_FILEKIND,
   FOUT_RECSIZE,
   FOUT_FRAMESIZE,
   FOUT_BLOCKSIZE;

POINTER
   P,
   PINPUT;

EBCDIC ARRAY
   EA [0:255];

ARRAY
   A [0:0];

DEFINE
   COMMA = , #,
   ABORT (S) = BEGIN
                   REPLACE P:EA BY S;
                   WRITE(RMT,OFFSET(P),EA);
                   MYSELF.STATUS := -1;
               END #;
```

```
PINPUT := POINTER(INPUT);
IF PINPUT NEQ 48"00" THEN
BEGIN
    REPLACE P:EA[0] BY """;
    REPLACE P:P BY PINPUT FOR REMAININGCHARS(P)-2
                                        UNTIL = 48"00";
    REPLACE P:P BY "".";
    REPLACE FIN.IOHSTRING(R) BY EA;
    WRITE(RMT,OFFSET(P)-3, EA[1]);
    IF BOOLEAN(R) THEN
        ABORT("IOHSTRING attribute error" COMMA
               PINPUT FOR OFFSET(P)-3);
END;
R := FIN.AVAILABLE;
IF R NEQ VALUE(OKRSLT) THEN
    ABORT("Input file not available (" COMMA
           R FOR * DIGITS COMMA ")");
FIN_FILEKIND := FIN.FILEKIND;
FIN_RECSIZE := FIN.MAXRECSIZE;
FIN_FRAMESIZE := FIN.FRAMESIZE;
FOUT_FRAMESIZE :=
 IF NOT FOUT(FRAMESIZE).FILEEQUATED THEN
     48
 ELSE
     FOUT.FRAMESIZE;
FOUT_RECSIZE :=
 IF NOT FOUT(MAXRECSIZE).FILEEQUATED THEN
     FIN_RECSIZE*FIN_FRAMESIZE DIV FOUT_FRAMESIZE
 ELSE
     FOUT.MAXRECSIZE;
FOUT_FILEKIND :=
 IF NOT FOUT(FILEKIND).FILEEQUATED THEN
     FIN_FILEKIND
 ELSE
     FOUT.FILEKIND;
FOUT_BLOCKSIZE :=
 IF NOT FOUT(BLOCKSIZE).FILEEQUATED THEN
     FOUT_RECSIZE*30
 ELSE
     FOUT.BLOCKSIZE;
FOUT (FRAMESIZE  = FOUT_FRAMESIZE,
      MAXRECSIZE = FOUT_RECSIZE,
      BLOCKSIZE  = FOUT_BLOCKSIZE,
      FILEKIND   = FOUT_FILEKIND);
REPLACE P:EA BY "Converting: ", FIN.TITLE;
WRITE (RMT, OFFSET(P)-1, EA);
REPLACE P:EA BY "FILEKIND = ";
ATTRIBINFORMER (EA, 8, P, N, R, FIN_FILEKIND,
                2 & 1[11:4] & 0[43:2] & 1[47:4]);
REPLACE P:P+N BY  ", MAXRECSIZE = ", FIN_RECSIZE FOR * DIGITS;
 IF FIN_FRAMESIZE = 48 THEN REPLACE P:P BY " words"
                       ELSE REPLACE P:P BY " characters";
```

```
WRITE (RMT, OFFSET(P), EA);
REPLACE P:EA BY "Creating:   ", FOUT.TITLE;
WRITE (RMT, OFFSET(P)-1, EA);
REPLACE P:EA BY "FILEKIND = ",
N := 0;
R := 0;
ATTRIBINFORMER (EA, 8, P, N, R, FOUT_FILEKIND,
                2 & 1[11:4] & 0[43:2] & 1[47:4]);
REPLACE P:P+N BY  ", MAXRECSIZE = ", FOUT_RECSIZE FOR * DIGITS;
IF FOUT_FRAMESIZE = 48 THEN REPLACE P:P BY " words"
                       ELSE REPLACE P:P BY " characters";
WRITE (RMT, OFFSET(P), EA);
RESIZE (A,MAX(FIN_RECSIZE*FIN_FRAMESIZE DIV 48,
              FOUT_RECSIZE*FOUT_FRAMESIZE DIV 48));
WHILE NOT READ(FIN,FIN_RECSIZE,A) DO
    WRITE(FOUT,FIN_RECSIZE,A);
LOCK(FOUT);
REPLACE P:EA BY "File Locked:", FOUT.TITLE;
WRITE (RMT, OFFSET(P)-1, EA);

END CONVERTSTREAM.
```

Appendix A
Device Types and Associated File Attributes

Use Table A–1 to identify the general file attributes that can be used with each device type.

Table A–1. Device Types and Associated File Attributes

Attribute	Disk	CD	Printer	Remote	Tape	Other	Port
ACCESSDATE	X	X					
ACCESSDATEUT	X						
ACCESSTIME	X	X					
ACCESSTIMEUT	X						
ACCESSTZ	X						
ACTUALMAXRECSIZE							X
ADAPTABLE	X						
AFTER			X				
ALIGNFILE			X				
ALIGNMENT			X				
ALLOWSPECIALFILE	X						
ALTERDATE	X	X					
ALTERDATEUT	X						
ALTERNATEGROUPS	X						
ALTERTIME	X	X					
ALTERTIMEUT	X						
ALTERTZ	X						
ANYSIZEIO	X	X					
APL	X	X					
APPEND	X						
APPLICATIONCONTEXT							X

Table A–1. Device Types and Associated File Attributes

Attribute	Disk	CD	Printer	Remote	Tape	Other	Port
APPLICATIONGROUP							X
AREAADDRESS	X	X					
AREAALLOCATED	X	X					
AREALENGTH	X	X					
AREAS	X	X					
AREASECTORS	X	X					
AREA SINUSE	X	X					
AREASIZE	X	X					
ASSIGNTIME				X			
ASSOCIATEDFILENAME					X		
ATTERR	X	X	X	X	X	X	X
ATTMODIFYDATE	X						
ATTMODIFYDATEUT	X						
ATTMODIFYTIME	X						
ATTMODIFYTIMEUT	X						
ATTMODIFYTZ	X						
ATTVALUE	X	X	X	X	X	X	X
ATTYPE	X	X	X	X	X	X	X
AUTOUNLOAD					X		
AVAILABLE	X	X	X	X	X	X	X
AVAILABLEONLY							X
BACKUPDATE	X						
BACKUPDATEUT	X						
BACKUPKIND			X				
BACKUPTIME	X						
BACKUPTIMEUT	X						
BACKUPTZ	X						
BANNER			X				
BLANK	X	X		X	X	X	X
BLOCK	X		X		X	X	
BLOCKEDTIMEOUT							X

Table A–1. Device Types and Associated File Attributes

Attribute	Disk	CD	Printer	Remote	Tape	Other	Port
BLOCKSIZE	X		X	X	X	X	
BLOCKSTRUCTURE	X	X	X	X	X	X	X
BUFFERS	X	X	X	X	X	X	X
BUFFERSHARING	X						
BUFFERSIZE	X	X	X		X	X	
CARRIAGECONTROL			X				
CCSVERSION	X						
CENSUS				X			X
CHANGEDSUBFILE							X
CHANGEEVENT							X
CHECKPOINT			X				
CLEARAREAS	X						
COMPRESSING					X		X
COMPRESSION							X
COMPRESSIONCONTROL					X		X
COMPRESSIONREQUESTED					X		X
COPYDESTDATE	X						
COPYDESTDATEUT	X						
COPYDESTTIME	X						
COPYDESTTIMEUT	X						
COPYDESTTZ	X						
COPYSOURCEDATE	X						
COPYSOURCEDATEUT	X						
COPYSOURCETIME	X						
COPYSOURCETIMEUT	X						
COPYSOURCETZ	X						
CREATEPASSWORD	X						
CREATIONDATE	X	X			X		
CREATIONDATEUT	X						
CREATIONTIME	X	X					
CREATIONTIMEUT	X						
CREATIONTZ	X						

Table A–1. Device Types and Associated File Attributes

Attribute	Disk	CD	Printer	Remote	Tape	Other	Port
CRUNCHED	X	X					
CURRENTBLOCKLENGTH	X		X	X	X	X	
CURRENTRECORDLENGTH	X	X	X	X	X	X	X
CYCLE	X				X		
DENSITY					X		
DEPENDENTINTMODE	X	X	X	X	X	X	
DEPENDENTSPECS	X	X	X	X	X	X	
DESTINATION			X				
DIALOGCHECKINTERVAL							X
DIALOGPRIORITY							X
DIOFILESTRUCTURE	X	X					
DIRECTION					X		
DISPOSITION				X			
DOCUMENTTYPE	X	X					
DONOTSEARCHNETWORK							X
DONTCLEARAREASBYDE FAULT	X						
DUMMYFILE	X	X	X		X	X	
ENABLEINPUT				X			
EOF					X		
ESTIMATEDRECORDS	X		X				
EXCLUSIVE	X						
EXECUTEDATE	X						
EXECUTEDATEUT	X						
EXECUTETIME	X						
EXECUTETIMEUT	X						
EXECUTETZ	X						
EXTDELIMITER	X	X	X	X	X	X	X
EXTMODE	X	X	X	X	X	X	X
FAMILYINDEX	X	X					
FAMILYNAME	X	X	X				
FAMILYOWNER					X		

Table A–1. Device Types and Associated File Attributes

Attribute	Disk	CD	Printer	Remote	Tape	Other	Port
FILECLASS	X	X	X	X	X	X	X
FILEEQUATED	X	X	X	X	X	X	X
FILEKIND	X	X			X		
FILELENGTH	X	X					
FILENAME	X	X	X	X	X	X	X
FILEORGANIZATION	X	X					
FILESECTION					X		
FILESTATE	X	X	X	X	X	X	X
FILESTRUCTURE	X	X	X				
FILETYPE	X	X	X	X	X	X	X
FILEUSE	X	X	X	X	X	X	
FLEXIBLE	X						
FORMID			X				
FRAMESIZE	X	X	X	X	X	X	X
FRAMESIZECENSUS							X
GENERATION	X				X		
GROUP	X				X		
HOSTNAME	X	X	X	X	X	X	X
HSFILECOPY	X						
INPUTEVENT				X			X
INPUTTABLE	X	X	X	X	X	X	
INTERACTIVEFILE	X	X	X	X	X	X	X
INTMODE	X	X	X	X	X	X	X
INTNAME	X	X	X	X	X	X	X
IOCLOCKS	X	X	X	X	X	X	X
IOHFUNCTIONNAME						X	
IOHINTERFACENAME						X	
IOHLIBACESS						X	
IOHLIBPARAMETER						X	
IOHPREFIX						X	
IOHSTRING						X	
IOHTITLE						X	

Table A–1. Device Types and Associated File Attributes

Attribute	Disk	CD	Printer	Remote	Tape	Other	Port
IOINERROR	X	X			X		
KERBEROSACCESS	X						
KIND	X	X	X	X	X	X	X
LABEL	X	X	X	X	X	X	
LABELKIND					X		
LASTACCESSIBLEAREA	X						
LASTRECORD	X	X					
LASTSUBFILE							X
LFILENAME	X	X	X	X	X	X	X
LIBMAINTAPPEND					X		
LIBMAINTDIR					X		
LICENSEKEY	X						
LINENUM			X				
LOCATECAPABLE					X		
LOCKEDFILE	X		X		X		
LTITLE	X	X	X	X	X	X	X
MAXCENSUS							X
MAXFRAMESIZECENSUS							X
MAXRECORDNUMBER	X						
MAXRECSIZE	X	X	X	X	X	X	X
MAXSUBFILES							X
MINRECSIZE	X	X	X	X	X	X	
MYDOMAINNAME							X
MYHOST							X
MYHOSTGROUP							X
MYIPADDRESS							X
MYNAME							X
MYUSE	X	X	X	X	X	X	
NETACCESSPOINT							X
NEWFILE	X	X	X	X	X	X	
NEXTRECORD	X	X	X		X	X	
NONBLOCK	X						

Table A–1. Device Types and Associated File Attributes

Attribute	Disk	CD	Printer	Remote	Tape	Other	Port
NORECOVERY	X						
NORESOURCEWAIT	X						
NOTE	X		X				
ODDBLOCKSIZE					X		
OFFSITE					X		
OFNOTIFICATION	X	X		X	X	X	
OPEN	X	X	X	X	X	X	X
OPTIONAL	X	X		X	X	X	
OUTPUTEVENT							X
OUTPUTTABLE	X	X	X	X	X	X	
OWNER	X						
OVERRIDEEXTMODE	X	X			X		
PAGE			X				
PAGECOMP			X				
PAGESIZE			X				
PARITY					X		
PASSIVEOPEN							X
PATHNAME	X	X	X	X	X	X	X
PENDINGPROTECTEDFILE		X					
PERMITTEDACTIONS	X	X					
POPULATION	X			X			
PORTSEGMENTIO							X
PRESENT	X	X	X	X	X	X	
PRINTCHARGE			X				
PRINTCOPIES			X				
PRINTDISPOSITION	X		X				
PRINTERBACKUPDATA	X						
PRINTERKIND			X				
PRINTPARTIAL			X				
PRINTREQUEST	X		X				
PRODUCT	X						

Table A–1. Device Types and Associated File Attributes

Attribute	Disk	CD	Printer	Remote	Tape	Other	Port
PROPAGATESECURITYTO DIRS	X						
PROPAGATESECURITYTO FILES	X						
PROTECTION	X	X			X		
PROVIDERGROUP							X
READDATE	X						
READDATEUT	X						
READREVERSECAPABLE					X		
READTIME	X						
READTIMEUT	X						
READTZ	X						
RECEPTIONS				X			
RECORD	X		X		X	X	
RECORDINERROR	X	X			X		
RECORDLEVELLOCK	X						
REDIRECTION						X	
REINITIALIZE	X	X	X	X	X	X	X
RELEASEID	X						
REQUESTEDMAXRECSIZE							X
RESIDENT	X	X	X	X	X	X	
RESTRICTED	X	X					
RESULTLIST	X	X	X	X	X	X	X
ROWADDRESS	X	X					
ROWSINUSE	X	X					
SAVEBACKUPFILE			X				
SAVEFACTOR	X				X		
SAVEPRINTFILE	X		X				
SCRATCHPOOL					X		
SCREEN				X			
SCREENSIZE				X			
SEARCHRULE	X						

Table A–1. Device Types and Associated File Attributes

Attribute	Disk	CD	Printer	Remote	Tape	Other	Port
SECTORSIZE	X	X					
SECURITYADMIN	X						
SECURITYGUARD	X	X	X†		X		
SECURITYMODE	X				X		
SECURITYTYPE	X	X	X†		X		X
SECURITYUSE	X	X	X†		X		
SENSITIVEDATA	X	X					
SERIALNO	X				X		
SERVICE	X						X
SINGLEUNIT	X						
SIZEMODE	X	X	X	X	X	X	
SIZEOFFSET	X	X	X	X	X	X	
SIZEVISIBLE	X	X	X	X	X	X	
SIZE2	X	X	X	X	X	X	
STATE	X	X	X	X	X	X	X
STATIONCOUNT				X			
STATIONLIST				X			
STATIONNAME				X			
STATIONSALLOWED				X			
STATIONSDENIED				X			
SUBFILEERROR							X
SYNCHRONIZE	X	X			X		
TANKING				X			
TAPEREELRECORD					X		
TIMELIMIT				X			
TIMESTAMP	X	X					
TITLE	X	X	X	X	X	X	X
TOTALSECTORS	X	X					
TRAINID			X				
TRANSFORM			X				
TRANSLATE	X	X	X	X	X	X	X
TRANSLATING	X	X	X	X	X	X	X

Table A–1. Device Types and Associated File Attributes

Attribute	Disk	CD	Printer	Remote	Tape	Other	Port
TRANSMISSIONNO				X			
TRANSMISSIONS				X			
TRIMBLANKS			X	X			
UNIQUETOKEN	X						
UNITNO	X	X	X		X	X	
UNITS	X	X	X	X	X	X	X
UPDATEFILE	X						
USECATALOG	X				X		
USEDATE	X	X					
USERBACKUPNAME			X				
USERCODE	X						
USERINFO	X	X					
USETIME	X	X					
VERSION	X				X		
WARNINGS	X						
WIDTH				X			
YOURDOMAINNAME							X
YOURHOST	X	X	X	X	X	X	X
YOURHOSTGROUP							X
YOURIPADDRESS							X
YOURNAME							X
YOURNSAPA							X
YOURPRESENTATIONSEL							X
YOURSESSIONSEL							X
YOURTRANSPORTSEL							X
YOURUSERCODE							X

† Applies to backup files only.

Appendix B
Format of Pack Labels

Table B–1 describes the format and contents of pack labels.

Table B–1. Format of Pack Labels

Position	Length (Bytes)	Contents
000–003	004	VOL1 identifier
004—009	006	Pack serial number
010	001	Reserved
011–027	017	Pack identification (FAMILYNAME)
028—029	002	System-Interchange code Native Mode = 67
030	001	Reserved
031—036	006	Reserved
037—050	014	Owner's identification
051—056	006	Mirrored disk date/time stamp
057	001	Relative index of pack within set
058	001	Mask of online mirrored copies
059	001	Recovery option D = DMS Blank = DISCARD
060	1	Auditing marker V = Set closed C = Creation or audit application
061	001	Mirrored Disk Pooling Facility (MDPF) spare disk pool marker 0 = not a spare 1 = free spare 2 = in-use spare

Table B–1. Format of Pack Labels

Position	Length (Bytes)	Contents
062—078	027	Reserved
079	001	Reserved
080—083	004	VOL2 identifier
084—088	005	Initialization date
089—094	006	Initializing system. Native = 67MC <mark digit> <level no>
095–102	008	Directory link. Native mode links to pack master header directory block.
103–110	008	Master available table link
111–118	008	Available table link Native mode unused
119	001	Integrity flag
120–125	006	Reserved
126–131	006	Reserved
132–179	048	Reserved
180–359	180	Reserved for security information

Appendix C
Disk File Headers

Disk file headers contain the attributes that define disk files, including such information as file title, actual location of each area of the file, maximum and minimum record sizes, block size, title of the security guard file, if any, and so on. This information is used primarily by the master control program (MCP) directory-management and logical-I/O routines. The system stores the disk file headers for permanent disk files in disk directory files located on the base unit or units of each disk family. Disk file headers are also found on the Library Maintenance system tapes.

Disk file headers vary in size. The first portion of the header is the header fixed data area. It contains attributes that are fixed in size and pertain to most files. It also contains items used by the MCP to maintain the headers of open files. The size of the fixed data area is given by HDRFIXEDSIZEF.

The fixed data area is followed by zero or more area address words that contain the address information for the data areas assigned to the file.

The area address words are followed by zero or more optional attribute words. The optional attribute words define attributes that pertain to only a limited number of files. Depending on the type of attribute, an optional attribute word either contains the value of the attribute or points to an area in the header data area where the value of the attribute is stored.

The optional attribute words are followed by an optional header data area. If the header data area is present, it contains the file title and the values of those optional attributes whose values cannot be stored in the optional attribute words.

The last word in the header is a checksum word.

Note: *Changes to the layout of the disk file header can be made without notice, and occasionally, changes can be inadvertently omitted from the documentation. Consequently, the operation of user programs that use disk file headers should be verified whenever the MCP release levels are changed.*

There are various methods to obtain disk file header information that is independent of changes to the structure of the disk file header. One method is to use GETSTATUS. Refer to the GETSTATUS/SETSTATUS Reference Manual *for information about Request Type 3 (Directory Calls). Another method, if a disk file header is already available, is to use the GETHEADERATTRIBUTE procedure documented later in this section. If neither of the preceding methods are used, defines should be used to describe the location of all words and fields to be accessed. The correctness of these defines should be verified at every software release by examining the declaration of the disk file header in the MCP symbol code.*

User Interface Procedures

Two interface procedures are available as MCP support library procedures: CONVERTHEADER and GETHEADERATTRIBUTE. Programs using these procedures can extract header information independent of the header layout.

CONVERTHEADER

A disk file header can be converted to the Version 7 format for transparency of disk file header formats to utility programs. The conversion is done by calling CONVERTHEADER and passing the header as a parameter. The conversion is done in place so that the header will be modified when the call completes.

This interface procedure is available in both ALGOL and Pascal.

The declarations within the user (ALGOL) program should be as follows:

```
LIBRARY MCP ( FUNCTIONNAME = "MCPSUPPORT.",
              LIBACCESS    = BYFUNCTION
            );

INTEGER PROCEDURE CONVERTHEADER (HDR);
ARRAY HDR [*];
   LIBRARY MCP;;
```

HDR contains the input disk file header to be converted.

The procedure converts HDR to the current format and returns the result of the operation. The following are the values returned:

Value	Meaning
> 0	Successful conversion. HDR contains the converted header. Its new size in words is returned.
–1	A fault occurred during conversion. A program dump can be obtained by running the calling program with the task FAULT option set.
–2	A bad header was passed as input.
–3	The header passed has an incorrect version; it is not 3, 4, 5, 6, or 7.

Notes:

- *HDR is converted to the current header version, if it is not in that format, and crunched. That is, no unused space will remain in the header. The length of the header might increase or decrease. If the length of the converted header increases, it might exceed the length of the array passed in. In this case, CONVERTHEADER resizes the header array so it is large enough to hold the converted header.*

  Be aware that Pascal programmers must be sure to pass a large enough array, because RESIZE does not work properly for Pascal arrays. In Pascal, the length of the HDR parameter should be at least 30 words longer than the size of the header contained in it.
- *The ORIGHEADERVERSIONF field in word 35 of HDR is set to the version number of the header passed to CONVERTHEADER.*
- *If a fault occurs during conversion, the contents of HDR are undefined.*
- *The procedure returns a "bad header" error if a header obtained using the DCALGOL* DISKHEADER READ *operation is passed to it as a parameter, since these headers have a special format.*

The GETHEADERATTRIBUTE procedure is used to interrogate disk file header attributes.

This interface procedure is available in ALGOL and Pascal.

The declarations within the user (ALGOL) program should be as follows:

```
LIBRARY MCP ( FUNCTIONNAME = "MCPSUPPORT.",
              LIBACCESS    = BYFUNCTION
            );

INTEGER PROCEDURE GETHEADERATTRIBUTE(HDR,WHICH,AREA,VAL,A);
VALUE   WHICH, AREA;
INTEGER WHICH, AREA;
REAL    VAL;
ARRAY   HDR[*], A[*];
LIBRARY MCP;
```

HDR contains the input header from which the requested attribute is to be extracted.

WHICH specifies the attribute being interrogated (Table C-1 shows the attributes and their corresponding mnemonics). In general, only attributes can be interrogated, not fields that are used for the maintenance of the header. However, certain fields that are useful in the analysis of the header are also accessible. An attribute is specified in ALGOL by *VALUE (<attribute name mnemonic>)* and in Pascal by *DFHVALUE (<attribute name mnemonic>)*.

AREA contains the area number for area-address relative attributes. It is not used for nonarea-relative attributes. The AREA number is zero relative.

VAL receives the value of the attribute if a word, integer, or a Boolean-valued attribute is being interrogated. For Boolean-valued attributes, 0 (zero) is returned if the attribute is false; otherwise, 1 is returned. Zero is returned in VAL if the attribute being interrogated is not set.

For list-valued attributes like TITLE, RELEASEID, and so forth, the length of the attribute value in the units in which it is stored is returned in VAL. Zero is returned in VAL if the list-valued attribute is not set.

"A" is an array in which list-type attributes are returned.

The following values are returned for the GETHEADERATTRIBUTE procedure:

Value	Meaning
0	Operation successful. For word, integer, or Boolean-valued attributes, VAL receives the value of the attribute. For list-type attributes, VAL receives the length of the attribute in the units in which it is stored, or 0 (zero) if the attribute is not set. Array A contains the list-type attribute.
–1	A fault has occurred. A program dump can be obtained by running the calling program with the task FAULT option set.
–2	A bad header was passed as input.
–3	The header passed has an incorrect version; it is not 3, 4, 5, 6, or 7.
–4	The attribute specified by the user is unknown to the system.
–5	The area number specified by the user is invalid.
–6	The attribute specified by the user has been deimplemented (is no longer supported by the operating system).

Notes:

- *Except when GETHEADERATTRIBUTE is called for the DFHHDRBLOCKLENGTH and DFHHDRLENGTH attributes, GETHEADERATTRIBUTE converts the HDR to the current header version, if it is not in that format, and crunches the HDR. That is, no unused space will remain in the header. If the length of the converted header increases, it might exceed the length of the array passed in. In this case, CONVERTHEADER resizes the header array so it is large enough to hold the converted header.*

  You can call GETHEADERATTRIBUTE with the DFHHDRLENGTH attribute in order to determine the size of the header in words. Calls for DFHHDRLENGTH do not cause the header to be converted, so you only need to pass the first three words of the header when you call GETHEADERATTRIBUTE for the DFHHDRLENGTH attribute. You need to know the size of the header so that you can be sure that you put all the words of the header into the array before you make other calls on GETHEADERATTRIBUTE or CONVERTHEADER.

Be aware that Pascal programmers must be sure to pass a large enough array, because RESIZE does not work properly for Pascal arrays. In Pascal, the length of the HDR parameter should be at least 30 words longer than the size of the header contained in it.

- *The COREINDEXF field in word 35 of HDR is set to 0 (zero).*
- *If a fault occurs during conversion the contents of HDR are undefined.*
- *The procedure returns a "bad header" error if a header obtained using the DCALGOL* DISKHEADER READ *operation is passed to it as a parameter, since these headers have a special format. The DCALGOL* DISKHEADER ATTRIBUTES *must be used to obtain the attributes from such headers.*

Examples

In ALGOL the following call returns 0 (zero) in X if the operation is successful, and returns the field contained in bits [15:16] of word 3 of HDR in VAL.

```
X :=GETHEADERATTRIBUTE(HDR, VALUE(DFHMAXRECSIZE),
                       0, VAL, A);
```

Similarly, the following call returns 0 (zero) in X if the operation is successful, extracts the file name from HDR and stores it in A in standard form, and returns the length, in bytes, of the standard-form name in VAL.

```
X := GETHEADERATTRIBUTE(HDR, VALUE(DFHTITLE),
                        0, VAL, A);
```

Similarly, the following call returns 0 (zero) in X if the operation is successful, and returns in VAL the field contained in bits [32:33] of the area address word corresponding to area 10.

```
X :=GETHEADERATTRIBUTE(HDR, VALUE(DFHSECTORADDRESS),
                       10, VAL, A);
```

In Pascal the following call returns 0 (zero) in X if the operation is successful, and returns the field contained in bits [15:16] of word 3 of HDR in VAL.

```
X := GETHEADERATTRIBUTE(HDR, DFHVALUE(DFHMAXRECSIZE),
                        0, VAL, A);
```

Table C–1 lists the attributes that can be interrogated and the type of information returned in each case. Note that the GETHEADERATTRIBUTE procedure returns the values of attributes as they are stored in the header; no conversion or translation is done. For example, the file name is stored as a standard form name within the header, but never includes the disk family name. GETHEADERATTRIBUTE returns this name in standard form without conversion, whereas the TITLE file attribute returns the name in display form. Similarly, the SECURITYUSE file attribute has values different from the ones reported for DFHSECURITYUSE. In all cases, the interface procedure returns data in disk file header format.

Table C–1. Disk File Header Attributes

Attribute Mnemonic	Field Name or Item in Header	Row Number Required	Type
DFHACCESSTIMESTAMP	Access Time Stamp	No	Word
DFHACCESSTZ	ACCESSTZF	No	Integer
DFHACTIVEROW	ACTIVEROWF	Yes	Boolean
DFHALIGNFILE	ALIGNFILE attribute	No	Std Form
DFHALIGNMENT	ALIGNMENT attribute	No	Boolean
DFHALLOCATEDROW	ALLOCATEDROWF	Yes	Boolean
DFHALTERNATEGROUPS	Alternate Groups list	No	Alternate Groups Standard Form▸
DFHALTERTIMESTAMP	Alter Time Stamp	No	Word
DFHALTERTZ	ALTERTZF	No	Integer
DFHAPLFILE	APLMAF	No	Boolean
DFHATTMODIFYTIMESTAMP	Attribute Modify Timestamp	No	Word
DFHATTMODIFYTZ	ATTMODIFYTZF	No	Integer
DFHBACKUPTIMESTAMP	Backup Timestamp	No	Word
DFHBACKUPTZ	BACKUPTZF	No	Integer
DFHBADINFO	BADINFO information	No	Byte list
DFHBANNER	BANNER attribute	No	Boolean
DFHBLOCKLENGTH	HDRBLOCKLENGTHF	No	Integer
DFHBLOCKSIZE	BLOCKSIZEF	No	Integer
DFHCATALOGED	HDRCATF	No	Boolean
DFHCCSVERSION	CCSVERSION	No	Integer
DFHCCSVERSIONSET	Returns TRUE if header CCSVERSION is set.	No	Boolean
DFHCHECKEOF	CHECKEOFF	No	Boolean
DFHCLEARAREAS	CLEARAREASF	No	Boolean
DFHCODEFILEHANDLING	CODEFILE handling	No	Word
DFHCODEFILEINFO	CODEFILE information	No	Word list
DFHCODEFILESECURITY	CODEFILE security	No	2-word list

Table C–1. Disk File Header Attributes

Attribute Mnemonic	Field Name or Item in Header	Row Number Required	Type
DFHCOPYDESTTIMESTAMP	Copy Destination Timestamp	No	Word
DFHCOPYDESTTZ	COPYDESTTZF	No	Integer
DFHCOPYNUMBER	COPYNUMBERF	No	Integer
DFHCOPYSOURCETIMESTAMP	Copy Source Timestamp	No	Word
DFHCOPYSOURCETZ	COPYSOURCETZF	No	Integer
DFHCREATIONTIMESTAMP	Creation Time Stamp	No	Word
DFHCREATIONTZ	CREATIONTZF	No	Integer
DFHCRUNCHED	CRUNCHEDF	No	Boolean
DFHCYCLE	CYCLE	No	Integer
DFHDELETEDROW	DKDELETEDF	Yes	Boolean
DFHDMLOCKBITS	DMLOCKBITSF	Yes	Integer
DFHDMREADLOCK	DMREADLOCKF	Yes	Boolean
DFHDMWRITELOCK	DMWRITELOCKF	Yes	Boolean
DFHDOCUMENTTYPE	DOCUMENTTYPE	No	Word
DFHDOCUMENTTYPEPARAMS	DOCUMENTTYPE parameters	No	Byte list
DFHDUPLICATED	DUPLICATEDBIT	No	Boolean
DFHEOFLASTBITS	EOFU	No	Integer
DFHEOFSCRUBBED	EOFSCRUBBEDF	No	Boolean
DFHEOFSECTOR	EOFV	No	Integer
DFHEXECUTETIMESTAMP	Execute Timestamp	No	Word
DFHEXECUTETZ	EXECUTETZF	No	Integer
DFHEXTMODE	EXTMODE†	No	Integer
DFHFILEFAMILYINDEX	FILEFAMILYINDEXF	No	Integer
DFHFILEKIND	FILEKINDF	No	Integer
DFHFILELENGTH	FILELENGTH attribute	No	Word
DFHFILEORGANIZATION	FILEORGANIZATIONF	No	Integer
DFHFILESTRUCTURE	FILESTRUCTUREF	No	Integer
DFHFILESTRUCTURESOURCE	FILESTRUCTURE_SOURCF	No	Integer
DFHFLATHDRLENGTH	FLATHDRLENGTHF	No	Integer

Table C–1. Disk File Header Attributes

Attribute Mnemonic	Field Name or Item in Header	Row Number Required	Type
DFHFORMID	FORMID attribute	No	Std Form
DFHGENEALOGY	GENEALOGYF	No	Integer
DFHGRANULATEDPRIVILEGES	Granulated privilege information	No	2-word list
DFHGROUP	Group Usercode	No	Byte list▲
DFHHDRLENGTH	HDRLENGTHF	No	Integer
DFHHEADERVERSION	VERSIONF	No	Integer
DFHIADORWLOROW	DKIADORWLOF	Yes	Boolean
DFHIDENTITY	IDENTITY	No	Byte list‡
DFHINDEXWASSET	INDEXWASSETF	Yes	Boolean
DFHKEYEDIOINFO	KEYEDIOINFO	No	Byte list
DFHLABEL	LABEL attribute	No	Integer
DFHLICENSEKEY	LICENSEKEY	No	Byte list□
DFHLOCATION	HDRLOCATIONF	No	Integer
DFHLOCKEDFILE	LOCKEDFILEF	No	Boolean
DFHMAXRECSIZE	MAXRECSIZEF	No	Integer
DFHMINRECSIZE	MINRECSIZEF	No	Integer
DFHMULTIUSEWORD	Multiuse Word	No	Word
DFHNFTCSUM	NFT check sum	No	Word
DFHNFTFILEKIND	NFT file kind	No	Integer
DFHNFTREC	NFT record number	No	Integer
DFHNOTE	NOTE	No	Byte list‡
DFHNUMBEROFROWS	NUMROWSF	No	Integer
DFHORIGHEADERVERSION	ORIGHEADERVERSIONF	No	Integer
DFHOWNER	Owner Usercode	No	Byte list▸
DFHPAGECOMP	PAGECOMP attribute	No	Std Form

Table C–1. Disk File Header Attributes

Attribute Mnemonic	Field Name or Item in Header	Row Number Required	Type
DFHPERMANENCY	PERMANENCYF	No	Boolean
DFHPERMITTEDACTIONS	PERMITTEDACTIONS	No	Word
DHFPHYSICALMODESIZEF	PHYSICALMODESIZEF	No	Integer
DFHPRINTERBACKUPDATA	PRINTERBACKUPDATA attribute	No	Word list
DFHPRINTERKIND	PRINTERKIND attribute	No	Integer
DFHPRIVILEGEDFILE	PRIVUSERF	No	Boolean
DFHPRODUCT	PRODUCT	No	Byte list▲
DFHPROPAGATESECURITYTODIRS	Propagate security to directories	No	Integer
DFHPROPAGATESECURITYTOFILES	Propagate security to files	No	Integer
DFHPROTECTED	PROTECTIONF	No	Boolean
DFHPUBLICRWX	PUBLICRWXF	No	Word
DFHREADTIMESTAMP	Read Timestamp	No	Word
DFHREADTZ	READTZF	No	Integer
DFHRECORDFORMAT	RECORDFORMATINFOF	No	Integer
DFHRECORDTYPE	RCRDTYPE	No	Integer
DFHRELEASEID	RELEASEID	No	Byte list‡
DFHRESTRICTIONMASK	Access Restriction Mask	No	Word
DFHROWADDRESSWORD	Area Address Word	Yes	Word
DFHROWFAMILYINDEX	FAMILYINDEXF	Yes	Integer
DFHROWSIZE	ROWSIZEF	No	Integer
DFHROWTAIL	ROWTAILF	No	Integer
DFHSAVEFACTOR	SAVEFACTORF	No	Integer
DFHSECTORADDRESS	SEGADDRESSF	Yes	Integer
DFHSECTORSIZE	SECTORSIZEF	No	Integer
DFHSECURITYGUARD	SECURITYGUARD	No	Byte list□
DFHSECURITYMODE	SECURITYMODEF	No	Word
DFHSECURITYTYPE	SECURITYCODEF ■	No	Integer

Table C–1. Disk File Header Attributes

Attribute Mnemonic	Field Name or Item in Header	Row Number Required	Type
DFHSECURITYUSE	READWRITEF ■	No	Integer
DFHSENSITIVEDATA	SENSITIVEDATAF	No	Boolean
DFHSERVICELIST	SERVICELIST◂	No	Byte list▾
DFHSINGLEPACK	SINGLEF	No	Boolean
DFHSIZEMODE	SIZEMODE	No	Integer
DFHSIZEOFFSET	SIZEOFF	No	Integer
DFHSIZE2	SIZESZ	No	Integer
DFHSUSPICIOUSEOF	SUSPICIOUSEOFF	No	Boolean
DFHSYSTEMFILE	SYSTEMFYLEF	No	Boolean
DFHTIMESTAMP	Time Stamp	No	Word
DFHTITLE	File Name	No	Byte list□
DFHTOBECRUNCHED	WILLCRUNCHF	No	Boolean
DFHTOTALSECTORS	TOTALSECTORS information	No	Word
DFHTRAINID	TRAINID attribute	No	Integer
DFHTRANSINPUT	Transform parameters	No	Std Form
DFHTRANSLIB	Transform library	No	Std Form
DFHTRANSLIBSL	Transform SL library	No	Boolean
DFHTRANSNAME	Transform name	No	Std Form
DFHUNITS	RCDUNTS	No	Boolean
DFHUSERINFO	USERINFO	No	Word
DFHVERSION	GENVERSN	No	Integer
DFHWARNINGS	WARNINGS	No	16-bit byte list
DFHWROTELASTROW	WROTELASTROWF	No	Boolean

† The physical mode optional attribute is returned if it is set; otherwise PHYSICALMODEF is returned.

‡ Special standard form name is returned. Special standard form names have a single name node that can have a length of up to 251 characters.

□ Standard form name is returned. Refer to Appendix D for a description of the standard form name.

▲ The first byte of the list is a binary number indicating the number of bytes that follow. In the case of a usercode, the following bytes are the usercode name. In cases where the attribute was set by a process not running under a usercode, the two bytes that follow the length byte are 48"00" 8"*" or 48"005C".

▼ The first two bytes (bytes 0, 1) of the list are a version signature of 4 "0227". The next two bytes (bytes 2, 3) of the list are a binary number indicating the number of bytes in the list, including these header bytes. The next two bytes (bytes 4, 5) are a filler with the value 0 (zero). The next byte (byte 6) of the list is a binary number indicating the number of names in the list. The following bytes (bytes 7 . . n) form a list of names in substandard form.

■ This field exists in version 6 headers only. For version 7 headers the value is derived from SECURITYMODEF.

◄ For more information on the SERVICELIST header attribute, see the MP (Mark Program) system command in the *System Commands Reference Manual*.

► The Alternate Groups Standard Form is structured as follows:

Character	Description
1	Total number of characters in the whole string (self-inclusive)
2	Version byte (currently 0)
3	Number of Alternate Group identifiers
4	Group identifiers, each preceded with one permission character followed by a length character (not self-inclusive) Permission Character values: [2:1] read permission [1:1] write permission [0:1] execute permission

For example, the Alternate Groups string

```
"GROUP1 : RW, GROUP2 : RWX"
```

would have the following standard form:

```
48"130002" 48"0606" 8"GROUP1" 48"0706" 8"GROUP2"
```

Disk File Header Versions

Prior to the SSR 42.1 release, the system used disk file headers that were in version 6 format. In the SSR 42.1 release, a new version 7 disk file header format was introduced.

Version 7 disk file headers contain some additional information that is not contained in version 6 headers. This information includes:

- Six new Timestamp Words:
  - Read Timestamp
  - Execute Timestamp
  - Attribute Modify Timestamp
  - Copy Source Timestamp
  - Copy Destination Timestamp
  - Backup Timestamp
- Time zone valued attributes for the nine header Timestamps:
  - ACCESSTZ
  - ALTERTZ
  - ATTMODIFYTZ
  - BACKUPTZ
  - COPYDESTTZ
  - COPYSOURCETZ
  - CREATIONTZ
  - EXECUTETZ
  - READTZ
- GROUP usercode
- SECURITYMODEF and related information

You will find a complete description of version 6 and version 7 disk file headers later in this appendix.

Disk Families

Each disk family has a family header version, which controls the format of the disk file headers written to that family. The MCP currently supports only families with a family header version of 7.

Note: *Under some circumstances, the system operator can use the CONVERTHEADERS system utility to change the family header version of a family. Refer to the* ClearPath MCP Migration Guide *for more information about using the CONVERTHEADERS system utility.*

Library Maintenance Tapes

The system can create library maintenance tapes that contain either version 6 or version 7 disk file headers. The version of the disk file headers written to a specific tape is version 7 by default. Use SW6 to create version 6 disk file headers.

When a disk file header is written to a library maintenance tape in version 6 format, the values of the new version 7 header attributes are discarded. Consequently, the values returned by these attributes may not be reliable for files that have been copied from library maintenance tapes that have version 6 disk file headers.

Notes:

- *A "long file name" refers to a file name that has more than 12 nodes (excluding the user code and family name) or that has a node with more than 17 characters (excluding quotes (" ")).*
- *If you are copying a file with a long file name to tape, library maintenance marks the header as being version 8. Version 8 disk file headers are in version 7 format except that they can not be copied from tape by MCPs prior to SSR 44.2.*
- *If you are copying a file with a long file name from one host to another with network file transfer, the source host marks the disk file header as version 8. The file will fail to copy if the destination host's MCP level is prior to SSR 44.2.*

Version 6 Disk File Header Layout

This subsection describes the word layouts of the Version 6 disk file header. The comments included with each field definition are a general description of the usage of the field and should not be taken as exact definitions. Additionally, many of the fields related to the maintenance of disk file headers by the MCP are valid only under limited circumstances. In general, user programs should use only those header fields that describe the header itself (for example, HDRBLOCKLENGTHF, HDRFIXEDSIZEF, VERSIONF, and so on) or that correspond to file attributes.

WORD 0 All Directory Records - Validity, Location, and Size

Name	Field	Description
MARKERF	[47:16]	Validity marker
INUSEMARK	4'3F3F'	Header or catalog block
AVAILMARK	4'3C3C'	Available record
BADAREAMARK	4'3A3A'	Bad record
MCPUSEMARK	4'3939'	MCPINFO and other records
HDRBLOCKLENGTHF	[31:11]	Total size of record in words. (In memory, this field has the current header size.)
HDRLOCATIONF	[20:21]	Directory location (record number)

WORD 1 Header Information

Name	Field	Description
OPENCOUNTF	[47:12]	Number of logical files using header
FILEKINDF	[35:12]	FILEKIND attribute
PERMANENCYF	[23:01]	0 = Temporary file 1 = File header is in the directory
WRITTENONF	[22:01]	1 = File written on
WROTELASTROWF	[21:01]	1 = Last allocated area of a protected file has been written to
HEADERCHANGEDF	[20:01]	1 = Header has been altered but not written to the directory
HDRFIXEDSIZEF	[19:08]	Size in words of fixed part of header
HDRAVAILSPACEF	[11:12]	Number of available words in the header data area

WORD 2 Record Format and Miscellaneous Information

Name	Field	Description
Reserved	[47:01]	
PROTECTIONF	[46:01]	1 = Protected file
CHECKEOFF	[45:01]	1 = Must find EOF of protected file
SYSTEMFYLEF	[44:01]	1 = Nonremovable file
APLMAF	[43:01]	1 = APL file control
PHYSICALMODEF	[42:03]	EXTMODE at creation. If this field has the value 5 (ASCII) and the physical mode optional attribute is set, the physical mode optional attribute contains the actual EXTMODE value.
RECORDFORMATINFOF	[39:40]	Record layout information
RCDUNTS	[39:01]	UNITS attribute
RCRDTYPE	[38:04]	FILETYPE attribute
SIZEMODE	[34:03]	SIZEMODE attribute
SIZEOFF	[31:16]	SIZEOFFSET attribute
SIZESZ	[15:16]	SIZE2 attribute

WORD 3 Disk Blocking Information

Name	Field	Description
BLOCKSIZEF	[47:16]	Physical BLOCKSIZE
MINRECSIZEF	[31:16]	Physical MINRECSIZE
MAXRECSIZEF	[15:16]	Physical MAXRECSIZE

WORD 4 Timestamp; Time(6) Format

WORD 5 Area Information and Miscellaneous Information

Name	Field	Description
VERSIONF	[47:04]	Header format version
HDRLEVEL3	3	Releases 2.7 through 3.2
HDRLEVEL4	4	Releases 3.3 through 3.5
HDRLEVEL5	5	Release 3.6
HDRLEVEL6	6	Release 3.6 and later
PRIVUSERF	[43:01]	1 = Privileged file
SENSITIVEDATAF	[42:01]	1 = Must scrub areas before disk space is returned
SECURITYCODEF	[41:02]	Class of security; SECURITYTYPE file attribute; attribute values differ from header values
SEC_PUBLICV	0	Attribute value = 1
SEC_GUARDEDV	1	Attribute value = 2
SEC_CONTROLLEDV	2	Attribute value = 3
SEC_PRIVATEV	3	Attribute value = 0
READWRITEF	[39:02]	Read/write security; SECURITYUSE file attribute; attribute values differ from header values
SEC_IOV	0	Attribute value = 3
SEC_INV	1	Attribute value = 1
SEC_OUTV	2	Attribute value = 2
SEC_SECUREDV	3	Attribute value = 0
Reserved	[37:01]	
INRESERVEF	[36:01]	1 = Reserve or squash in process for this header
MODEF	[35:02]	Access mode for the file
SHARED	0	
EXCLUSIVE	1	
NUMROWSF	[33:10]	Number of area address words
ROWSIZEF	[23:24]	Sectors in each area

WORD 6 Genealogy Information

Name	Field	Description
SAVEFACTORF	[47:10]	Save factor
GENEALOGYF	[37:22]	Genealogy information
CYCLE	[37:14]	CYCLE attribute
GENVERSN	[23:08]	VERSION attribute
FILEFAMILYINDEXF	[15:08]	FAMILYINDEX attribute for entire file (nonzero if set by user)
FILEORGANIZATIONF	[07:03]	FILEORGANIZATION attribute
NOTRESTRFO	0	
RELATIVEFO	1	
INDEXEDFO	2	
INDEXEDNOTRESTRFO	3	ISAM file with relative keys
PLIISAMFO	4	
KEYEDIOIIFO	5	
KEYEDIOIISETFO	6	
DUPLICATEDBIT	[04:01]	1 = File is duplicated ***Note:*** *Duplicated files are not supported by 46.1 and later releases.*
COPYNUMBERF	[03:04]	Number of this copy ***Note:*** *Duplicated files are not supported by 46.1 and later releases.*

WORD 7 Disk Pack Information

Name	Field	Description
Reserved	[47:01]	
HDRCATF	[46:01]	1 = File is cataloged
WILLCRUNCHF	[45:01]	1 = File will be crunched when open count is 0
CRUNCHEDF	[44:01]	1 = File is crunched
TIMESTAMPSYNCF	[43:01]	0 = MCP will timestamp the catalog block the next time the file or header is changed to specify that the file has changed since it was last backed up
NUMOPTIONALATTSF	[42:11]	Number of optional attribute words
EOFSCRUBBEDF	[31:01]	1 = EOF area has been scrubbed

Name	Field	Description
FLATHDRLENGTHF	[30:11]	For permanent files, size in words of header in the directory (valid only while header is in memory)
KRBRESF	[19:01]	RESTRICT KERBEROS restricted file access
KRBMPF	[18:01]	MP + KERBEROS privileged program
JOBORDERF	[19:20]	Job files only (maintained by controller)
CPJOBFILEIBF	[19:01]	Saved value of job file header IB field (FILEKIND=CPJOBFILE only)

WORD 8 Title Information

Name	Field	Description
INTERCHANGEF	[47:01]	INTERCHANGE file attribute; not supported by Mark 4.1 and later releases, but might be set in headers created by an operating system that has a release level earlier than the Mark 4.1 release
SINGLEF	[46:01]	1 = All areas must be on the same pack
Reserved	[45:01]	Unused
CYLMODEHDRF	[44:01]	CYLINDERMODE file attribute; not supported by the Mark 3.9 release, but might be set in headers created by an operating system that has a release level earlier than the Mark 3.9 release
HDRTITLESIZEF	[43:08]	Size of title in bytes
HDRTITLEINDEXF	[35:12]	Word index of title from start of header (title is in header data area)
BASEUNITF	[23:12]	Unit number of family base unit
LASTF	[11:12]	Used by GETUSERDISK

WORD 9 EOF Information

Name	Field	Description
EOFU	[47:20]	Number of bits (0 or more) in use in any partial in-use sector following the last full sector in the file
EOFV	[27:28]	Number of full sectors in the file; all areas, up to and including the one containing the EOF, are counted as though they are allocated areas

WORD 10 Creation TIMESTAMP; TIME(6) Format

Name	Field	
CREATIONDATEF	[47:16]	
CREATIONTIMEF	[31:32]	

WORD 11 Alter TIMESTAMP; TIME(6) Format

Name	Field	
ALTERDATEF	[47:16]	
ALTERTIMEF	[31:32]	

WORD 12 Access TIMESTAMP; TIME(6) Format

Name	Field	
ACCESSDATEF	[47:16]	
ACCESSTIMEF	[31:32]	

WORD 13 Multiuse Word

BD Information Word for Backup Files

Name	Field	Description
BDSTKNUMF	[11:12]	Stack number of stack with which the BD file is to be associated

DM TIMESTAMP for Unisys e-@ction Enterprise Database Server for ClearPath MCP Data Files

TIME(6) format

Next Area Word for System Directory Files

Name	Field	Description
ALLOCATINGF	[47:01]	1 = GETAROW running for this directory header
	[39:10]	Next FASTIOV area
	[29:10]	Next VSIOV area
	[19:10]	Next CATIOV area
	[09:10]	Next FLATIOV or VOLIOV area

WORD 14 Core Index Word (Memory Only)

Name	Field	Description
ASYNCHEADERWRITEF	[47:01]	1 = Header will be written to disk asynchronously
INHEADERCACHEF	[46:01]	1 = Header is in the disk file header memory cache
WASPERMF	[45:01]	1 = Header has been, and might still be, in the directory
Reserved	[44:01]	
ORIGHEADERVERSIONF	[43:04]	Original version of this header
CONTENDORSF	[39:12]	Number of programs waiting for exclusive file
HEADERCACHESLOTF	[27:12]	Index in header cache table
COREINDEXF	[15:16]	Index of header in DISKFILEHEADERS array in memory

WORD 15 File Structure

Name	Field	Description
FILESTRUCTURE_SOURCEF	[47:02]	0 = File was created before the FILESTRUCTURE attribute was introduced 1 = File was created with direct I/O 2 = File was not created with direct I/O and the FILESTRUCTURE value was not explicitly specified 3 = File was not created with direct I/O and the FILESTRUCTURE value was explicitly specified

Name	Field	Description
ROWTAILF	[45:18]	Number of unused bytes in the last sector of each area of a file with the FILESTRUCTURE value of STREAM
SECTORSIZEF	[27:20]	Sector size in bytes (A value of 0 implies 180)
FILESTRUCTUREF	[07:08]	FILESTRUCTURE attribute
ALIGNED180FS	0	ALIGNED180
STREAMFS	1	STREAM
BLOCKEDFS	5	BLOCKED

WORD 16 DRC System Header Information (Memory Only)

Name	Field	Description
DRCUSERCOUNTF	[47:12]	Number of MCP DRC functions using the header
TEMPFILEOWNERSNRF	[35:12]	For temporary files, stack number charged for sectors controlled by the file
FAMILYLOCF	[23:24]	For permanent files, family locator of family entry charged for the file

WORD 17 Miscellaneous

Name	Field	Description
LOCKEDFILEF	[47:01]	1 = File has a LOCKEDFILE value of TRUE
CLEARAREASF	[46:01]	1 = File has a CLEARAREAS value of TRUE
Reserved	[46:46]	Unused

Area Address Words for Version 6 Headers

The area address words follow the fixed data area; one word for each area (no more than 1000). NUMROWSF in the fixed data area gives the total number of area address words that the header has (zero or more). ALLOCATEDROWF in each area address word indicates if space has been allocated to an area.

Name	Field	Description
DMLOCKBITSF	[47:02]	DMS usage
DMREADLOCKF	[47:01]	
DMWRITELOCKF	[46:01]	

Name	Field	Description
DKIADORWLOF	[45:01]	1 = Write lock-out
DKDELETEDF	[44:01]	1 = Family member on which the area resides has been deleted by an RC
ALLOCATEDROWF	[43:01]	1 = Space allocated for the area
INDEXWASSETF	[42:01]	1 = Family index was set
FAMILYINDEXF	[41:08]	Family index of family member on which the area is or will be located
ACTIVEROWF	[33:01]	1 = Family member on which the area resides is online
SEGADDRESSF	[32:33]	Sector address

Optional Attribute Words for Version 6 Headers

The optional attribute words (zero or more) follow the area address words. Depending on the size of the attribute value, an optional attribute word either contains the value of the attribute or points to an area in the header data area where the attribute value can be found. The number of optional attribute words in the header is given by NUMOPTIONALATTSF in the header fixed data area. The optional attribute words area can contain unused words (unused words have HAAVAILF set). The unused words are included in the value stored in NUMOPTIONALATTSF.

Name	Field	Description
HAAVAILF	[47:01]	1 = Word is available
HAHOWSTOREDF	[46:05]	How ATTRIBUTE is stored
HAFIELDV	0	Field; value in HAVALUEF
HAWORDLISTV	1	Word list; size in words in HASIZEF
HABYTELISTV	2	Byte list; size in bytes in HASIZEF
HA16BITBYTELISTV	3	16-bit byte list; size in units of 16 bits in HASIZEF

Name	Field	Description
HANUMF	[41:10]	Attribute number
HASECGUARDV	2	Security guard (byte list)
HAUSERINFOV	3	USERINFO attribute (field or one-word list)
HANOTEV	4	NOTE attribute (byte list)
HAWARNINGSV	5	WARNINGS attribute (16-bit byte list)
HAACCESSRESTRICTIONV	6	File access restriction mask (field or one-word list). Refer to the Header Data Area for further information.
HALICENSEKEYV	7	LICENSEKEY attribute (byte list)
HARELEASEIDV	8	RELEASEID attribute (byte list)
HANAMEDATTRIBUTESV	9	User-defined disk file attributes (UDDFAs) (byte list). Refer to the Header Data Area for further information.
HAPHYSICALMODEV	10	New physical mode (EXTMODE attribute) (field)

Name	Field	Description
HANFTRECV	11	30-sector record number that a Native File Transfer (NFT) copy has reached (field or one word list)
HANFTFILEKINDV	12	For NFT, original FILEKIND (field or one word list)
HADOCUMENTTYPEV	13	DOCUMENTTYPE attribute (field)
HAPERMITTEDACTIONSV	14	PERMITTEDACTIONS attribute (field)
HADOCUMENTTYPEPARAMSV	15	Document type optional parameters (byte list)
HANFTCSUMV	16	Checksum for NFT (one word list)
HACODEFILEINFOV	17	Copy of code file SEG0 record (word list)
HACODFILESECURITYV	18	Code file security information (word list). Refer to Header Data Area for further information.
HACODEFILEHANDLINGV	19	Special handling information for code files (field or one-word list). Refer to Header Data Area for further information.
HAIDENTITY V	20	Code file identity (byte list)
HACCSVERSIONV	21	CCSVERSION file attribute (field)
HAKEYEDIOINFOV	22	KEYEDIOII information (byte list)
HAVALUEF	[31:32]	Attribute value for field type
HASIZEF	[31:16]	Number of entries in list for list type attributes; HOWSTOREDF gives the units.
HAINDEXF	[15:16]	Word index of the start of the list in the header data area for list type attributes (the index is from the start of the header)

Header Data Area for Version 6 Headers

The header data area (which might not be present) contains the values for those optional attributes whose values do not fit into the attribute words themselves. Additionally, the TITLE attribute, if present, is stored in this area. Attributes can appear in any order and are not necessarily in the same order that their attribute words appear in the optional attribute area. The area can contain unused words. If so, they will be the last words in the area. The number of unused words is given by HDRAVAILSPACEF in the header fixed data area.

The layouts of selected optional attribute values that are stored in the header data area follow.

ACCESSRESTRICTION Optional Attribute

Reserved	[47:45]	
READONLYFILEF	[02:01]	File is read-only.
RESTRICTEDFILEF	[01:01]	File is restricted.
MUSTBESECURITYADMINF	[00:01]	Access is restricted to security Administrator.

User-Defined Disk File Attributes (UDDFAs)

Count	Number of UDDFAs plus 1 (2 bytes)
UDDFA structure	Length of UDDFA (2 bytes) Characteristics (2 bytes) [13:1] WasRead [12:2] Read access [10:1] WasWritten [09:2] Write access [07:1] HasDefault [06:4] DataType [02:1] Boolean value [01:1] Default Boolean value [00:1] Unused Length of the attribute name (1 byte) Name of attribute (number of bytes specified by name length) Default value length (2 bytes, if any) Default value (if any) Value length (2 bytes, if any) Value (number of bytes specified by value length, if any)
Additional UDDFA structures (if any)	
End marker	Value = 0 (2 bytes)

For additional information about the preceding fields and how to use UDDFAs, refer to the *File Attributes Reference Manual*.

CODEFILESECURITY Optional Attribute

WORD 0		
Reserved	[47:46]	Unused
HDRTASKINGPROGRAMF	[02:01]	1 = Program has tasking privileges.
HDRSECADMINPROGRAMF	[01:01]	1 = Program has security administrator privileges.
HDRPRIVILEGEDPROGRAMF	[00:01]	1 = Program is privileged.
WORD 1		
Reserved	[47:46]	Unused
HDRTASKINGTRANSPARENTF	[02:01]	1 = Program has tasking with transparent privileges.
HDRSECADMINTRANSPARENTF	[01:01]	1 = Program has transparent security administrator privileges.
HDRTRANSPARENTPRIVF	[00:01]	1 = Program has transparent privileges.

CODEFILEHANDLING Optional Attribute

Reserved	[47:41]	Unused
HDRONEONLYIF	[06:01]	1 = Program has been marked as ONEONLY.
HDRCONTROLPROGRAMF	[05:01]	1 = Program is a control program.
HDRAUTOSUPPRESSF	[04:01]	1 = Automatically suppress program MIX display.
HDRRESIDENTPROGRAMF	[03:01]	1 = Program is a resident program.
HDRSDIHALTF	[02:01]	1 = A code file is targeted for a gamma SDI halt.
HDRLOCKPROGRAMF	[01:01]	1 = Program should be locked when initiated.
CODEFILETODFHF	[00:01]	1 = Code file handling bits move to header from SEG0.

CHECKSUM for Version 6 Headers

The last word in disk file headers (HDR[HDRBLOCKLENGTH(HDR) –1]) contains the header CHECKSUM. It is valid only for headers on disk.

Version 7 Disk File Header Layout

This subsection describes the word layouts of the version 7 disk file header.

Version 7 disk file headers have a maximum size of 65,536 words.

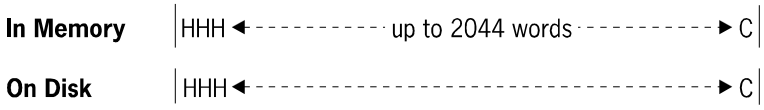
Regardless of its size, a disk file header being used by the system in memory is a monolithic data structure. Similarly, a header is always returned by GETSTATUS directory requests as a monolithic structure.

However, a version 7 disk file header larger than 2,048 words is segmented into multiple directory records when it is written to a disk directory file, including directory files created by certain GETSTATUS directory requests.

Refer to the *GETSTATUS/SETSTATUS Reference Manual* for information about Request Type 3 (Directory Calls).

When a header is segmented, additional words of control information are added at segment boundaries. These words are discarded when a segmented header is reassembled into a monolithic structure.

Figure C–1 illustrates the structure of an unsegmented header in memory and in a disk directory.

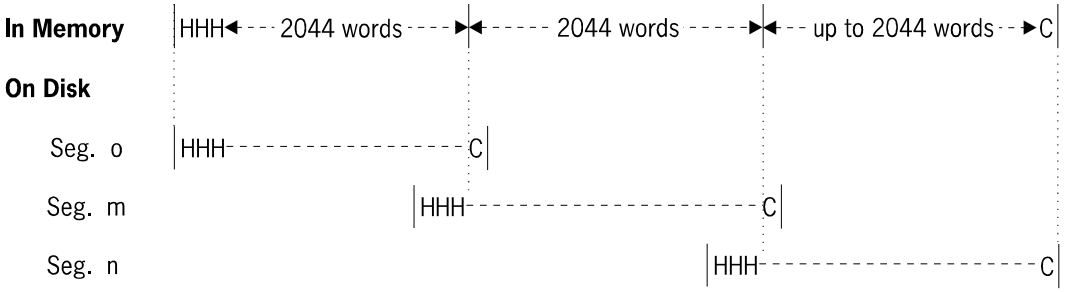


Notes:

1. "HHH" denotes the three words of control information at the start of the header.
2. "C" denotes the checksum word at the end of the header.
3. The header on disk occupies a maximum of 69 180-byte sectors (2048 words).

Figure C–1. Unsegmented Header

Figure C–2 shows the structure of a segmented header in memory and in a disk directory.



Notes:

1. "HHH" denotes the three words of control information at the start of each disk header.
2. "C" denotes the checksum word at the end of each segment.
3. Each disk segment, except possibly the last (segment n) which may be smaller, occupies 69 180-byte sectors (2048 words).
4. For larger headers there will be additional occurrences of segment m.

Figure C–2. Segmented Header

Disk file headers larger than 2,048 words are written as multiple records to library maintenance tapes, but without the additional words of control information used for segmented headers.

In the following layouts, comments included with each field definition are a general description of the usage of the field and should not be taken as exact definitions. Additionally, many of the fields related to the maintenance of disk file headers by the MCP are valid only under limited circumstances. In general, user programs should use only those header fields that describe the header itself (for example, HDRBLOCKLENGTHF, HDRFIXEDSIZEF, VERSIONF, and so on) or that correspond to file attributes.

WORD 0 All Directory Records - Validity, Location, and Size

Name	Field	Description
MARKERF	[47:16]	Validity marker
INUSEMARK	4'3F3F'	Header, FAST entry, or catalog block
AVAILMARK	4'3C3C'	Available record
BADAREAMARK	4'3A3A'	Bad record
MCPUSEMARK	4'3939'	MCPINFO and other records
CONTMARK	4'3737'	Continuation segment of a header
HDRBLOCKLENGTHF	[31:11]	Total size of record in words; must NOT be used to obtain the total size of a header. (See HDRLENGTHF and FLATHDRLENGTHF)
HDRLOCATIONF	[20:21]	FLAT location of this record (file relative sector number).

WORD 1 Segmented Header Control 1

This contains information that links to next segment and only exists in header in memory for first segment.

Name	Field	Description
HDRCYCLEF	[47:04]	Incremented modulo 16 each time header is written so that READER, etc. can detect cases where not all segments of a header were successfully written to disk.
HDRSEGMENTSF	[43:08]	First header segment only - Total number of segments in header.
HDRSEGMENTNUMBERF	[43:08]	Continuation segments only - The segment number of this segment.
	[35:04]	Reserved for future use
NEXTSEGLENGTHF	[31:11]	Length of next segment in words.
NEXTSEGLOCF	[20:21]	FLAT location of next segment (file relative sector number).

WORD 2 Segmented Header Control 2

First Header Segment

Name	Field	Description
HDRTYPEF	[47:08]	Type of header
FLATTYPE	0	Header for file in FLAT
JOBFILETYPE	1	Job file header in JOBDESC file
DIRSTUBTYPE	2	POSIX directory stub
REDIRECTTYPE	3	POSIX redirection header
	[39:07]	Reserved for future use
FLATHDRLENGTHF	[32:17]	For permanent files, HDRLENGTHF is the length in words of the header the last time it was read from disk or from a library maintenance tape or written to disk.
HDRLENGTHF	[15:16]	Size of header in words. For segmented headers this is the size of the assembled header and excludes the intermediate control and checksum words. This field is valid for headers in memory, in FLAT directories, and on Library Maintenance tapes.

Continuation Header Segments

This information does not exist in header in memory.

Name	Field	Description
	[47:27]	Reserved for future use
HDRSEG0LOCF	[20:21]	Back pointer to first segment of header (file relative sector number).

WORD 3 Disk Blocking Information

Name	Field	Description
BLOCKSIZEF	[47:16]	Physical BLOCKSIZE
MINRECSIZEF	[31:16]	Physical MINRECSIZE
MAXRECSIZEF	[15:16]	Physical MAXRECSIZE

WORD 4 Timestamp; Time(6) Format Currently Not Used.

WORD 5 Header Version and Structure Information

Name	Field	Description
VERSIONF	[47:04]	Header format version
MCPHDRLEVEL	7	The header used in memory by this MCP
NUMOPTIONALATTSF	[43:12]	Number of optional attribute words
DATAAREAIXF	[31:16]	Word index of start of header data area
HDRAVAILSPACEF	[15:16]	Number of available words in header data area

WORD 6 Header Fixed Size and Title Pointer

Name	Field	Description
HDRFIXEDSIZEF	[47:08]	Size in words of fixed part of header
HDRFIXEDSIZEV	36	Current fixed size
NUMBEROFTITLESF	[39:08]	Number of TITLEs or links in header
HDRTITLESIZEF	[31:16]	Size of title area in words
HDRTITLEIXF	[15:16]	Word index from start of header of title area

WORD 7 Permanent Disk File Information

Name	Field	Description
FILESTRUCTURE_SOURCEF	[47:02]	Used to determine if the file was created with a changeable default FILESTRUCTURE. (For files that were not created with direct I/O, the default value of FILESTRUCTURE varies by MCP release.)
FSS_PREIMPLEMENTATION	0	File was created before FILESTRUCTURE_SOURCE was recorded.
FSS_DIRECT	1	File was created with direct I/O.
FSS_NONDIRECT_DEFAULT	2	File was not created with direct I/O and a FILESTRUCTURE value was not explicitly specified.
FSS_NONDIRECT_EXPLICIT	3	File was not created with direct I/O and a FILESTRUCTURE value was explicitly specified.

Name	Field	Description
FILESTRUCTUREF	[45:06]	FILESTRUCTURE attribute value (SECTORSTREAMFS and DEPENDENTFS are used by direct I/O and are never found in headers).
ALIGNED180FS	0	
STREAMFS	1	
BLOCKEDFS	5	
BLOCKSTRUCTUREF	[39:08]	BLOCKSTRUCTURE attribute. Note that catalog blocks still store FILETYPE.
FILEORGANIZATIONF	[31:08]	FILEORGANIZATION attribute
NOTRESTRFO	0	
RELATIVEFO	1	
INDEXEDFO	2	
INDEXEDNOTRESTRFO	3	
PLIISAMFO	4	
KEYEDIOIIFO	5	
KEYEDIOIISETFO	6	
KEYEDIOIISFFO	7	
HDR_RCDUNTSF	[23:01]	UNITS attribute 0 = WORDS 1 = CHARACTERS
BITSPERMODEF	[22:07]	Number of bits in a PHYSICALMODE (EXTMODE) frame
PHYSICALMODEF	[15:16]	EXTMODE attribute

WORD 8 Expiration, Genealogy, and File Kind Information

Name	Field	Description
SAVEFACTORF	[47:14]	SAVEFACTOR attribute. (Not enforced by the MCP.)
HDR_GENEALOGYF	[33:22]	Corresponds to GENEALOGYF in the LEB GEN1 word.
HDR_CYCLE	[33:14]	CYCLE attribute
HDR_GENVERSN	[19:08]	VERSION attribute
FILEKINDF	[11:12]	FILEKIND attribute

WORD 9 Security Information

Name	Field	Description
SYSTEMFILEF	[47:01]	1 if nonremovable, and so forth, system file.
LOCKEDFILEF	[46:01]	1 if nonremovable user file. (LOCKEDFILE attribute)
PRIVUSERF	[45:01]	1 if system file with special restrictions on CHANGE, REMOVE, and so forth, and on some file attribute operations.
NONPRIVJOBF	[44:01]	1 if initiating MCS forbids privileged user or security administrator status for job.
NONSYSTEMUSERJOBF	[43:01]	1 if initiating MCS forbids system user status for job.
SENSITIVEDATAF	[42:01]	1 if must scrub rows before disk space is returned to the system.
EOFSCRUBBEDF	[41:01]	1 if the row in which EOF lies has been scrubbed.
PERMITTEDACTIONSF	[40:11]	PERMITTEDACTIONS attribute.
PUBLICWXF	[29:03]	Read, write, execute, search permissions from which SECURITYUSE attribute value is derived.
KRBRESF	[26:01]	RESTRICTED KERBEROS restricted file access
KRBMPF	[25:01]	MP + KERBEROS privileged program
	[24:09]	Reserved
SECURITYMODEF	[15:16]	SECURITYMODE attribute.
	[15:02]	Reserved
GUARDOWNERF	[13:01]	Guard file also applies to owner.

Name	Field	Description
USEGUARDFILEF	[12:01]	Check Guard file permission.
SETUSERCODEF	[11:01]	Set USERCODE of Task to the OWNER value of the file on execution.
SETGROUPCODEF	[10:01]	Set GROUPCODE of Task to the GROUP value of the file on execution.
	[09:01]	Reserved
OWNERRWXF	[08:03]	Owner read, write, execute, or search permissions.
OWNERRF	[08:01]	Owner read permission.
OWNERWF	[07:01]	Owner write permission.
OWNERXF	[06:01]	Owner execute or search permission.
GROUPRWXF	[05:03]	Group read, write, execute, or search permissions.
GROUPRF	[05:01]	Group read permission.
GROUPWF	[04:01]	Group write permission.
GROUPXF	[03:01]	Group execute or search permission.
OTHERRWXF	[02:03]	Other read, write, execute, or search permissions.
OTHERRF	[02:01]	Other read permission.
OTHERWF	[01:01]	Other write permission.
OTHERXF	[00:01]	Other execute or search permission.

WORD 10 Creation Timestamp; TIME(6) Format

Name	Field	Description
CREATIONDATEF	[47:16]	yyyyddd - 1970000
CREATIONTIMEF	[31:32]	Time of day in 2.4 microseconds DIV 16

WORD 11 Alter Timestamp; TIME(6) Format

Name	Field	Description
ALTERDATEF	[47:16]	yyyyddd - 1970000
ALTERTIMEF	[31:32]	Time of day in 2.4 microseconds DIV 16

WORD 12 Access Timestamp; TIME(6) Format

Name	Field	Description
ACCESSDATEF	[47:16]	yyyyddd - 1970000
ACCESSTIMEF	[31:32]	Time of day in 2.4 microseconds DIV 16

WORD 13 Read Timestamp; TIME(6) Format

Name	Field	Description
TS_TIME6_DATEF	[47:16]	yyyyddd - 1970000
TS_TIME6_TIMEF	[31:32]	Time of day in 2.4 microseconds DIV 16

WORD 14 Execute Timestamp; TIME(6) Format

Name	Field	Description
TS_TIME6_DATEF	[47:16]	yyyyddd - 1970000
TS_TIME6_TIMEF	[31:32]	Time of day in 2.4 microseconds DIV 16

WORD 15 Attribute Modify Timestamp; TIME(6) Format

Name	Field	Description
TS_TIME6_DATEF	[47:16]	yyyyddd - 1970000
TS_TIME6_TIMEF	[31:32]	Time of day in 2.4 microseconds DIV 16

WORD 16 Copy Source Timestamp; TIME(6) Format

Name	Field	Description
TS_TIME6_DATEF	[47:16]	yyyyddd - 1970000
TS_TIME6_TIMEF	[31:32]	Time of day in 2.4 microseconds DIV 16

WORD 17 Copy Destination Timestamp; TIME(6) Format

Name	Field	Description
TS_TIME6_DATEF	[47:16]	yyyyddd - 1970000
TS_TIME6_TIMEF	[31:32]	Time of day in 2.4 microseconds DIV 16

WORD 18 Backup Timestamp; TIME(6) Format

Name	Field	Description
TS_TIME6_DATEF	[47:16]	Yyyyddd - 1970000
TS_TIME6_TIMEF	[31:32]	Time of day in 2.4 microseconds DIV 16

WORD 19 through 21 Currently Not Used

WORD 22 Miscellaneous Fields

Name	Field	Description
PHYSICALMODESIZEF	[47:06]	Size of PHYSICALMODE unit in bits
	[41:42]	Reserved

WORD 23 Miscellaneous Booleans and Row Size Information

Name	Field	Description
	[47:01]	Reserved
WILLDOWNSIZEAREAF	[46:01]	1 if closed with DOWNSIZEAREA option.
HDRCATF	[45:01]	1 if file is cataloged.
APLMAF	[44:01]	1 if APL attribute is set.
SINGLEF	[43:01]	1 if all rows must be on the same family member.
PROTECTIONF	[42:01]	1 if PROTECTION attribute value is PROTECTED.
CHECKEOFF	[41:01]	1 if the EOF must be found for a protected file.
WROTELASTROWF	[40:01]	1 if the last allocated row of a protected file has been written to. Set to 0 by RELEASEHEADER when the file is closed properly. It is also set to 0 when the file is crunched or when CHECKEOFF is set.
CRUNCHEDF	[39:01]	1 if the file is crunched.
WILLCRUNCHF	[38:01]	1 if the file will be crunched when the last user releases the header.
AREASIZESETF	[37:01]	1 if user explicitly set the AREASIZE and AREALENGTH file attributes.
PERMANENCYF	[36:01]	0 = Temporary file 1 = File header is in the directory
CLEARAREASF	[35:01]	1 = To force scrubbing for the file

Name	Field	Description
SUSPICIOUSEOFF	[33:01]	A value of 1 indicates that the end-of-file position should be treated with suspicion, because the file might have been closed improperly when created or last updated.
ROWSIZEF	[32:33]	Number of sectors in each area

WORD 24 Duplicated File Information, and Row Tail and Sector Size Information

Name	Field	Description
DUPLICATEDBIT	[44:01]	1 if file is duplicated (DUPLICATE attribute)
COPYNUMBERF	[43:04]	If duplicated, copy number of this file
ROWTAILF	[39:20]	Number of unused bytes at the end of the last sector of each full row for files with a STREAM value for FILESTRUCTURE.
SECTORSIZEF	[19:20]	For files on disk, the actual sector size, in bytes. On library tapes, the sector size in bytes for which the file is oriented.

WORD 25 Family Index, Number of Areas, and EOF Information

Name	Field	Description
FILEFAMILYINDEXF	[47:08]	FAMILYINDEX attribute for entire file (nonzero if set by user).
NUMROWSF	[39:16]	Number of row address words.
EOFU	[23:24]	Bit offset of EOF in any partial last sector (zero if EOFU is at a sector boundary; see also EOFV).

WORD 26 EOFV

Number of full sectors in the file assuming that all rows up to and including the row containing EOF are allocated. Stored as an integer; see also EOFU.

WORD 27 Reserved for POSIX File Serial Number

WORD 28 Time Zones

Name	Field	Description
ACCESSTZF	[47:08]	Time zone for Access timestamp
ALTERTZF	[39:08]	Time zone for Alter timestamp
ATTMODIFYTZF	[31:08]	Time zone for Attmodify timestamp
BACKUPTZF	[23:08]	Time zone for Backup timestamp
COPYDESTTZF	[15:08]	Time zone for Copy destination timestamp
COPYSOURCETZF	[07:08]	Time zone for Copy source timestamp

WORDS 29 Time Zones

Name	Field	Description
CREATIONTZF	[47:08]	Time zone for Creation timestamp
EXECUTETZF	[39:08]	Time zone for Execute timestamp
READTZF	[31:08]	Time zone for Read timestamp

WORDS 30 through 32 Owner Usercode

Usercode of the OWNER of the file or 48"02005C" for files without an OWNER.

WORD 33 Currently Not Used

WORD 34 Multiuse Word

BD Information Word for Backup Files

Name	Field	Description
DESTCONTROLF	[45:22]	See PIB path control word.
BDNAMEDF	[23:01]	Backup file explicitly named flag
BDSTKNUMF	[20:20]	Stack number of job

Job Information Word For Job Files

Name	Field	Description
JOBVALIDITYF	[47:01]	0 if from MAKEJOBFILE 1 if from WFL
JFBIT	[46:01]	1 if there is a job file to print
PBBITS	[45:06]	The location of the backup files for the job; relative bit positions must match those in BDPBBITS.
LPWHEREF	[45:03]	45->LPDK, 44->LPSRPK, 43->LPNAMEDPK
PBREMF	[39:01]	1 If a REMLP file was opened.
WFLFLAGSF	[38:03]	Flags returned by WFL COMPILER and its task attribute value
RESF	[38:01]	1 if RESOURCE attribute set for job
BF	[37:01]	1 if job has INSTRUCTION statement
FSF	[36:01]	1 if job has FETCH statement
JOBSUMF	[35:04]	JOBSUMMARY value
DEFAULT	0	
CONDITIONAL	1	
SUPPRESSED	2	
UNCONDITIONAL	3	
ABORTONLY	4	
NOJOBSUMMARYIOF	[31:01]	1 if NOJOBSUMMARYIO is set.
NOSUMMARYOVRDF	[30:01]	1 if any of tasks of the job terminated when HISTORY is not equal to NORMALEOT.
JSTITLESETF	[29:01]	1 if JOBSUMMARYTITLE is set.
JFPNOPDF	[28:01]	1 if the job file print should not use print defaults to disk.
CPRESTARTF	[27:01]	1 if there is a checkpoint that reflects current state of job.
JFCOPIEDF	[26:01]	1 if a CPJOBFILE has been created for job
CCRESTARTF	[25:01]	1 if job is being rerun with the RERUN statement.
MANRESTARTF	[24:01]	1 if job is to be restarted as the result of a RESTART system command.

Name	Field	Description
JOBPHASEF	[23:04]	Current job phase
INJOBQPHASE	0	
RESTARTPHASE	1	
RUNPHASE	2	
PRINTPHASE	3	
JOBMIXNOF	[19:20]	Mix number of job

Pbit I/O Count For Code Files

Name	Field	Description
SLCOUNTF	[38:08]	Count of support libraries references to the file
D1STACKLINKF	[30:15]	SNR of first D1 stack (code files).
PBITIOHIATUSF	[15:01]	Set to 0 when decrement PBITIOCOUNT to 0.
PBITIOCOUNTF	[14:15]	Count of pbit I/Os in process against file.

Next Row Word For Directory Files

Name	Field	Description
	[39:10]	Next FASTIOV row
	[29:10]	Next VSIOV or ARCIOV row
	[19:10]	Next CATIOV row
	[09:10]	Next FLATIOV or VOLIOV row
NOROWV	1023	Flag to indicate row needed

WORD 35 Memory-Only Items

This word contains fields used by the MCP to maintain headers in memory that, for various reasons, must be in the header and not in the parallel structure used for memory-only header related items.

This word is zeroed before the header is written to the directory or returned to users.

Name	Field	Description
ORIGHEADERVERSIONF	[35:04]	Original version of header read from disk or tape
BASEUNITF	[31:16]	Unit number of family base unit
COREINDEXF	[15:16]	Index of header in DISKFILEHEADERS

Area Address Words for Version 7 Headers

The area address words follow immediately after the fixed portion of the header, one word per area. FIRSTROWINDEX(HDR) gives the index of the first area address word.

Name	Field	Description
DMLOCKBITSF	[47:02]	DMS usage
DMREADLOCKF	[47:01]	
DMWRITELOCKF	[46:01]	
DKWLOF	[45:01]	1 if area is write locked-out
DKDELETEDF	[44:01]	1 if family member was deleted by an RC system command
ALLOCATEDROWF	[43:01]	1 if space has been allocated for the area
INDEXWASSETF	[42:01]	1 if family index was explicitly set
FAMILYINDEXF	[41:08]	Index of family member on which space is (or will be) allocated
ACTIVEROWF	[33:01]	1 if family member on which the area resides is online
SEGADDRESSF	[32:33]	Sector address of start of area

Optional Attribute Words for Version 7 Headers

The optional attribute words (zero or more) follow the area address words. Depending on the size of the attribute value, an optional attribute word either contains the value of the attribute or points to an area in the header data area where the attribute value can be found. The number of optional attribute words in the header is given by NUMOPTIONALATTSF in the header fixed data area. The optional attribute words area can contain unused words (unused words have HAAVAILF set). The unused words are included in the value stored in NUMOPTIONALATTSF. All maintenance of optional attribute words is done by SETHEADERATTRIBUTE. Additional information about optional attributes is contained in the description of the WHICH parameter of GET/SETHEADERATTRIBUTE that follows:

Name	Attribute Number or Field	Description
HAAVAILF	[47:01]	1 if word is available
HAHOWSTOREDF	[46:05]	How the attribute is stored
HAFIELDV	0	Field; value in HAVALUEF
HAWORDLISTV	1	Word list; size in words in HASIZEF
HABYTELISTV	2	Byte list; size in bytes in HASIZEF
HA16BITBYTELISTV	3	16-bit byte list; size in units of 16 bits in HASIZEF
HANUMF	[41:10]	Attribute number; see "Optional Attribute Numbers" later in this description.
HAVALUEF	[31:32]	Attribute value for field type
HASIZEF	[31:16]	Number of entries in list for list type attributes; HOWSTOREDF gives the units.
HAINDEXF	[15:16]	Word index in header data area of start of list for list type attributes

Optional Attribute Numbers

The following are used as values for HANUMF. Where appropriate, the layout of an optional attribute value is given following its optional attribute number.

Name	Attribute Number or Field	Description
HASECGUARDV	2	SECURITYGUARD attribute (byte list)
HAUSERINFOV	3	USERINFO attribute (field or one-word list)

Name	Attribute Number or Field	Description
HANOTEV	4	NOTE attribute (byte list)
HAWARNINGSV	5	WARNINGS attribute (16-bit byte list)
HAACCESSRESTRICTIONV	6	File access restriction mask (field or one-word list)
KERBEROSACCESSF	[03:01]	Value of KERBEROSACCESS file attribute
READONLYFILEF	[02:01]	1 if file is read only
RESTRICTEDFILEF	[01:01]	1 if file is restricted
MUSTBESECURITYADMINF	[00:01]	1 if access is restricted to the security administrator
HALICENSEKEYV	7	LICENSEKEY attribute (byte list)
HARELEASEIDV	8	RELEASEID attribute (byte list)
HANAMEDATTRIBUTESV	9	User-defined disk file attributes (UDDFAs) (byte list)
HAPHYSICALMODEV	10	EXTMODE attribute (field) (Version 6 header only)
HANFTRECV	11	30-sector record number that a Native File Transfer (NFT) copy has reached (field or one-word list)
HANFTFILEKINDV	12	Original value of FILEKINDF (NFT COPY usage only) (Field or one-word list)
HADOCUMENTTYPEV	13	DOCUMENTTYPE attribute (field)
HAPERMITTEDACTIONSV	14	PERMITTEDACTIONS attribute (field) (Version 6 header only)
HADOCUMENTTYPEPARAMSV	15	Parameters associated with DOCUMENTTYPE (byte list) (FTAM usage only)
HANFTCSUMV	16	Checksum for NFT COPY & VERIFY (one word list)
HACODEFILEINFOV	17	Copy of code file SEG0 record (word list)

Name	Attribute Number or Field	Description
HACODEFILESECURITYV	18	Code file security information (word list)
CODEFILESECURITYSIZEV	2	Size of attribute value in words.
Word 0		
HDRKERBEROSMPF	[04:01]	1 = Program is marked with MP + KERBEROS privilege.
HDRTASKINGPROGRAMF	[02:01]	1 = Program has tasking privileges.
HDRSECADMINPROGRAMF	[01:01]	1 = Program has security administrator privileges.
HDRPRIVILEGEDPROGRAMF	[00:01]	1 = Program is privileged.
Word 1		
HDRTASKINGTRANSPARENTF	[02:01]	1 = Program has tasking with transparent privileges.
HDRSECADMINTRANSPARENTF	[01:01]	1 = Program has transparent security administrator privileges.
HDRTRANSPARENTPRIVF	[00:01]	1 = Program has transparent privileges.
HACODEFILEHANDLINGV	19	Code file handling information (field or one-word list)
HDRONEONLYF	[06:01]	
HDRCONTROLPROGRAMF	[05:01]	
HDRAUTOSUPPRESSF	[04:01]	
HDRRESIDENTPROGRAMF	[03:01]	
HDRSDIHALTF	[02:01]	
HDRLOCKPROGRAMF	[01:01]	
CODEFILETODFHF	[00:01]	
HAIDENTITYV	20	Identity of code file (byte list)
HACCSVERSIONV	21	CCSVERSION file attribute (field) Values for CCSVERSION attribute are defined as mnemonics. Refer to CCSVERSION in the *File Attributes Reference Manual.*

Name	Attribute Number or Field	Description
HAKEYEDIOINFOV	22	KEYEDIOII information (byte list)
HASIZEINFOV	23	Information when BLOCKSTRUCTURE value is VARIABLEOFFSET (one-word list)
HDR_SIZEMODE	[47:16]	SIZEMODE attribute
HDR_SIZEOFF	[31:16]	SIZEOFFSET attribute
HDR_SIZESZ	[15:16]	SIZE2 attribute
HAJOBORDERV	24	Used by CONTROLLER to order JOBTABLE during JOBDESC file complement (field)
JOBORDERF	[19:20]	
HAJOBORGUNITV	25	Originating unit of job (field)
HACHECKPOINTNUMV	26	Checkpoint number (field)
HAJOBFILEPPBV	27	For job file, PPB with job attributes (word list)
HAUSERCODESV	28	Usercodes for OWNER and GROUP (word list - obsolete)
HABADINFOV	29	Sector and row information about errors that occurred when library maintenance copied file from tape to disk (word list) There are separate formats for 24-byte and 12-byte attributes.

If the size of the attribute is 24 bytes, then the format is as follows:

Word 0	Total number of errors library maintenance encountered
Word 1	Total number of sectors that were not copied to disk correctly
Word 2	Number of the area in which the first error occurred
Word 3	Sector number relative to start of area where the first error occurred

If the size of the attribute is 12 bytes, then the format is described as follows:

Word	[47:24] [23:24]	Count of errors Count of sectors in error
Word 1	[47:16] [31:32]	Row number of row with first error Sector number within row of first error

Name	Attribute Number or Field	Description
HAGROUPV	31	GROUP usercodes (or 48"02005C" for files without a group usercode)
HAPRODUCTV	32	PRODUCT attribute (byte list)
HASYSTEMTYPEV	33	System type (byte list)
HACAPABILITYV	34	Capability list (word list)
HASMBLIMITSV	35	SMB access limits (field)
HAMIXEDCASEIDV	36	SMB mixed case id (byte list)
HAGRANULATEDPRIVV	37	Code file granulated privilege information (2-word list).

Name	Attribute Number or Field	Description
GRANULATEDPRIVSIZEV	2	Size of attribute in words
Word 0		Granulated privilege
	[47:1]	1 = Program has the LOCALCOPY privilege.
	[46:1]	1 = Program has the CREATEFILE privilege.
	[45:1]	1 = Program has the REMOVE privilege.
	[44:1]	1 = Program has the WRITE privilege.
	[43:1]	1 = Program has the READ privilege.
	[42:1]	1 = Program has the EXECUTE privilege.
	[41:1]	1 = Program has the CHANGE privilege.
	[40:1]	1 = Program has the CHANGESEC privilege.
	[39:1]	1 = Program has the USERDATA privilege.
	[38:1]	1 = Program has the GETSTATUS privilege.
	[37:1]	1 = Program has the SETSTATUS privilege.
	[36:1]	1 = Program has the LOGINSTALL privilege.
	[35:1]	1 = Program has the LOGOTHERS privilege.
	[34:1]	1 = Program has the SYSTEMUSER privilege.
	[33:1]	1 = Program has the IDC privilege.
	[32:1]	1 = Program has the GSDIRECTORY privilege
Word 1		Granulated privilege transparency (same layout as word 0.)

Name	Attribute Number or Field	Description
HASERVICELISTV	38	SERVICELIST attribute (byte list) format: byte 0,1: *Version signature = 4"0224".* byte 2, 3: *Length in bytes (inclusive).* byte 4, 5: *Filler = 4"0000".* byte 6 *Number of service names.* Byte 7. .n: *List of names using substandard form.*
HAALTERNATEGROUPSV	42	Alternate Groups attribute in Alternate Groups Standard Form (byte list)
HAPROPSECTOFILESV	43 [31:32]	PropagateSecurityToFiles attribute (field) 0=Don't propagate 1=Propagate
HAPROPSECTODIRSV	44 [31:32]	PropagateSecurityToDirs attribute (field) 0=Don't Propagate 1=Propagate
HAPBDATAV	50	Printerbackupdata attribute (word list)

Header Data Area for Version 7 Headers

The header data area contains the value of those optional attributes whose values do not fit into their attribute words. Attribute values can appear in any order in this area and are not necessarily in the same order in which their attribute words appear in the optional attribute area. The area can contain unused words. If so, they will be the last words in the area. The number of unused words is given by HDRAVAILSIZEF in the fixed portion of the header. Unused words are always zero. DATAAREAIXF points to the start of the data area. The header data area is maintained by SETHEADERATTRIBUTE.

A file with a version 7 disk file header cannot be converted to a version 6 disk file header if the end-of-file exceeds the end-of-file permissible for a version 6 disk file header.

CHECKSUM for Version 7 Headers

The last word in directory records contains the record checksum. The only exception is available records, which have their checksums in Word 1. For segmented headers, the checksum word exists in the header in memory only for the last segment. Checksums are valid only for headers on disk or Library Maintenance tapes. The procedure WRITER has an example of how the checksum is computed.

Appendix D
Format of Library Maintenance Tapes

Library maintenance tapes are created in compact form. Library maintenance can read two forms: standard tape label form and compact form. The standard tape label form has ANSI69 or ANSI87 multi-file, multi-volume labels. The compact form does not comply with the ANSI multi-file label standards.

For both forms of library maintenance tapes, the first file on the tape is the *tape directory*. This tape directory has a file name of the form *<tapename>/FILE000* and is always written with standard ANSI69 or ANSI87 labels.

The first word of all blocks except the label records (that is, directory blocks, disk file header blocks, fixed blocks, and disk file data blocks) written to library maintenance tapes contains a bit code to indicate which form the tape was written in. If bit 45 of the first word of the first block of the tape directory is OFF, the tape was written completely with standard labels. If bit 45 of the first word of the first block of the tape directory is ON, the tape was written in compact form.

The format of a library maintenance tape can also be determined from the value stored in the USYST (tape type) field of the VOL1 (volume header) label.

Value	Meaning	Type of Tape
3	The tape is a library maintenance tape with standard labels.	Open-reel previous to Mark 4.1 release
6	The tape is a library maintenance tape in compact form.	Half-inch cartridge, 8-mm, open-reel, or digital linear tape (DLT)
7	The tape is a fixed-block library maintenance tape in compact form.	Quarter-inch cartridge tape

For library maintenance tapes, the BLOCKSIZE file attribute value in words is located in the HDR2 label of the tape file that follows the tape directory. When library maintenance is writing to a tape, the BLOCKSIZE value it uses depends on three things:

- If you specify a BLOCKSIZE value for that tape in the WFL "COPY" statement then library maintenance uses that value.
- If the installation has used the SYSOPS LMBLOCKSIZE system command to establish a non-zero default blocksize then library maintenance uses that value.
- Library maintenance uses a BLOCKSIZE value depending on the density of the tape.

Any specified BLOCKSIZE value that is not a multiple of 900 will be rounded up to the next multiple of 900. The BLOCKSIZE value actually used will be the rounded value plus one. This BLOCKSIZE value is stored in the HDR2 label of the tape file that follows the tape directory. However, if that value is 901 words, a value of 1024 is stored in the label.

The following table lists the densities and their corresponding default BLOCKSIZE values:

Density	Default Value	BLOCKSIZE Value in Label
800 BPI (Reel)	900	1024
1250 BPI (Quarter-inch cartridge)	2700	2701
1600 BPI (Reel)	900	1024
6250 BPI (Reel)	2700	2701
11000 BPI (8-millimeter)	4500	4501
38000 BPI (Half-inch cartridge)	4500	4501
FMT36TRK (Half-inch cartridge)	4500	4501
FMTAIT	4500	4501
FMTAIT2	4500	4501
FMTDLT10	4500	4501
FMTDLT20	4500	4501
FMTDLT35	4500	4501
FMTDLT40	4500	4501
FMTST9840	4500	4501
FMTDDS	4500	4501
FMTDDS2	4500	4501
FMTDDS3	4500	4501
FMTDDS4	4500	4501

The contents of volume header labels are illustrated in Appendix E, "Standard Tape Label Formats."

Note: *The format of the library maintenance tapes can be changed without notice. You should not rely on these specifications to read or duplicate the library maintenance tapes.*

Format of Library Maintenance Tapes with Standard Labels

The basic format of a library maintenance tape with standard labels is an ANSI multifile file of *n + 1* files, where *n* is the number of files copied. If the tape name is MTA, the files are named MTA/FILE000, MTA/FILE001, and so forth, with MTA/FILE000 being the tape directory.

The format of a library maintenance tape with standard labels is as follows:

```
Volume label or labels

Header labels

    tapemark

Tape directory (one or more blocks of 901 words or less)

    tapemark

End-of-file labels

    tapemark

Header labels

    tapemark

First copied file.  The first block contains the disk header.
The file data follows in 901-word or 2701-word blocks, except
that blocks corresponding to the ends of disk rows (areas)
might be shorter.

tapemark

End-of-file labels

    tapemark
    .
    .
    .

More copied files.
    .
    .
    .

EOV1 (if going to another volume)

    tapemark

    tapemark
```

The first word in all records is a transaction number. In the tape directories, the number in the first record is –1. In each succeeding record, the number is decremented by 1. The first record (the disk file header) in each of the copied files has a transaction number of 1. In each succeeding record, the number is incremented by 1.

Format of Library Maintenance Tapes in Compact Form

The beginning of a library maintenance tape in compact form is similar to that of a library maintenance tape with standard labels. The tape directory has standard labels, and the first disk file copied is preceded by standard HDR labels with the file name in the form *<tapename>/FILE001*.

However, the remaining files copied to the tape (except the last one) are separated only by a single tapemark; EOF and HDR labels do not appear. The last file on the tape is terminated by either EOV or EOF labels, depending on whether or not another volume is needed.

If a tape is a fixed-block tape and the disk file header is longer than 84 words (504 bytes), two bytes of filler are written between bytes 504 and 505 of the header (where the bytes of the header are numbered starting with 1).

The format of a library maintenance tape in compact form is as follows:

```
Volume labels

Header labels

    tapemark

Tape directory (one or more blocks of 901 words or less)

    tapemark


End-of-file labels

    tapemark

Header labels

    tapemark

First copied file.  The first block contains the disk file
header, and can be up to 2049 words long. The remaining blocks
(if any) contain the data for the file. These blocks range in
size from 31 words up to the maximum BLOCKSIZE indicated in the
HDR2 label for FILE001.

    tapemark

Next copied file (header and data)

    tapemark

Next copied file (header and data)
```

```
    .
    .
    .

(More copied files and tapemarks)
    .
    .
    .

    tapemark


End-of-file labels if no more volumes, otherwise end-of-volume
label

    tapemark

    tapemark
```

The first word of all directory, disk file header, and disk file data blocks is a code word. Bit 46 has the value 1 if the block is from the tape directory, or the value 0 if the block is not from the tape directory. Bit 45 has the value 1 if the tape is written in the compact format, or the value 0 if the tape is written with standard labels.

The field [44:20] contains a file number code. In directory blocks, the file number code is one less than the number of the file that follows the directory. This field always has the value 0 in the directory on the first reel or volume of library maintenance tapes. In disk file header and disk file data blocks, this field contains the file number corresponding to *nnn* in the file name *<tapename>/FILEnnn* for standard label library maintenance tapes.

The field [24:25] contains the block number of the block in that file or directory. The block numbers for the directory and for each file start from 1.

Note: *A program can read the tape directory of a library maintenance tape as a labeled tape file. However, the data files or their disk file headers on a library maintenance tape in the compact form can be read only if the program assigns the LABEL attribute a value of OMITTEDEOF.*

Format of the Tape Directory

The tape directory consists of one or more records. Each record is up to 901 words long and consists of a one-word block number and one word of link information, followed by up to 899 words of file names in standard form. All records are completely packed; a file name can be split across directory records. The link word is used only for multivolume library copies and is described later in this appendix.

The external file name is written in standard form, which is structured as follows:

Character	Description
1	Total number of characters in the whole string (self-inclusive)
2	Security byte Bits [1:2]: 2 System file ("*" specified) 3 Usercode specified (The first identifier is the usercode.)
3	Number of identifiers in external file name
4	Identifiers, each preceded by one character giving the length of that identifier (not self-inclusive)

For example, the external file name SYMBOL/MCP becomes the following in standard form, where *48* means to construct the following string from 4-bit input, but treat the results as 8-bit:

```
48"0E020206" 8"SYMBOL" 48"03" 8"MCP"
```

The end of the file names in the directory is flagged by a name character count field of 0 (zero).

The tape directory of library maintenance tapes produced by the Mark 4.1 release and later System Software Releases (SSRs) also contains a list of disk family names that indicate the original disk families from which the files are copied. This list of family names starts in Word 1 of the directory record that follows the file name list. The family name list consists of one or more four-word entries. The family name list is terminated by a word of all zeroes. Each family name entry is formatted as follows:

Word	Meaning
0	Count of files copied from the named family.
1–3	Name of family in substandard form. If the name is all zeroes, the original family name is unknown—the file was copied to this tape by Native File Transfer, or from a CD-ROM, or from a tape which was created by a Mark 4.0 or earlier MCP.

The order and counts in the family name list correspond to the file names appearing at the beginning of the directory. For example, if files A and B were copied from DISK, R was copied from PACK, and X was copied from DISK, the file names would appear in the order A, B, R, X, 0 (zero), and the family name list would appear as 2 DISK, 1 PACK, 1 DISK, 0 (zero).

To optimize reloading of specified files, each new volume contains a partial copy of the tape directory, which contains entries for those files not copied on the preceding reels. Thus, a COPY of a file copied to the third volume, for example, can be started on the third volume because library maintenance can determine where the requested file is from the partial directory on the third volume.

The link information word in Word 1 of the first block in the directory contains two fields, as follows:

```
CHARCNT    [47:16]
SKIPF      [25:10]
```

CHARCNT and SKIPF are needed because the directories of the succeeding volumes are copied from the pertinent records of the master directory. That means that the first record in any but the first directory begins, in general, with some names not pertinent to this volume. CHARCNT indicates how many characters to jump over (because the directory records are often split in the middle of a standard form external file name) to get to the beginning of the first complete file name. SKIPF specifies how many complete file names to skip over to access the name of the first complete file copied to this volume.

This next example is for a multivolume COPY operation with the following conditions:

- The COPY operation copied *n* files.
- Two volumes were used.
- The (*k*—1)th file was being written at the time of volume change.
- Two directory records were used—one for the name list and one record for the family name list.
- File *k* is the third complete name in the second directory record, but there are six characters ending the last standard form name in the first directory record before the first complete name.

```
Reel 1

    file 0

        1st directory record     Names:  1 through start of k-3
        2nd directory record     Names:  end of k-3 through n
        3rd directory record     Family name list

    file 1
    .
    .
    .
```

```
    file k-1 (start)
Reel 2

    file k-1 (end)

        2nd directory record     Names:  end of k-3 through n
        3rd directory record     Family name list (excludes family
                                 names for files on first reel).

    file k
    .
    .
    .
    file n
```

Note that the tape directory on any but the first volume is never the first tape file on that volume. Reel switch is sensed within a file, and the remainder of that file is written onto the new volume before the tape directory for that volume. The split file is logically considered to be on the volume on which it was begun.

The directory on succeeding volumes is again called MTA/FILE000, but the subsequent files are numbered sequentially, continuing where the preceding volume left off.

Note: *Files contained entirely on a succeeding volume have an indication in their label records that this is their first FILESECTION. Only files actually split over two volumes have a FILESECTION 2 designation in their label records.*

Appendix E
Standard Tape Formats

This appendix presents the standard tape formats used on MCP systems.

The figures in this appendix use the following notations:

Notation	Meaning
*	Tape mark
**	Double tape mark separated by an interrecord gap
EOF	End-of-file code
EOV	End-of-volume code
RFE	Reserved for expansion
RFS	Reserved for standard

Unlabeled Tapes

Figure E–1 shows the formats of an unlabeled single-file tape and an unlabeled multifile tape.

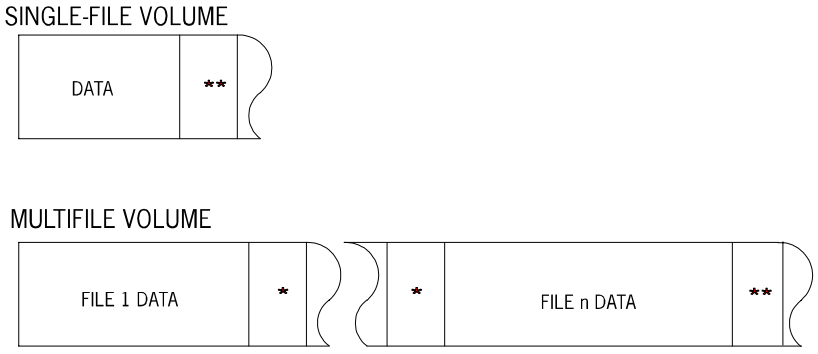


Figure E–1. Unlabeled Single-File Volume and Unlabeled Multifile Volume Formats

ANSI X3.27-1969 (ANSI69) Tapes

Figures E–2 through E–9 show the formats of the following types of ANSI X3.27-1969 (ANSI69) tape labels:

- Single-file, single-volume tape
- Multivolume-file and multifile-volume tape
- Multifile, multivolume tape
- Volume header
- File header 1
- File header 2
- User header and trailer labels

Figure E–10 illustrates the format of an ANSI69 scratch tape.

User header labels can appear immediately after HDR2, and user trailer labels can appear after either EOF2 or EOV1.

ANSI69 tapes are written in EBCDIC code.

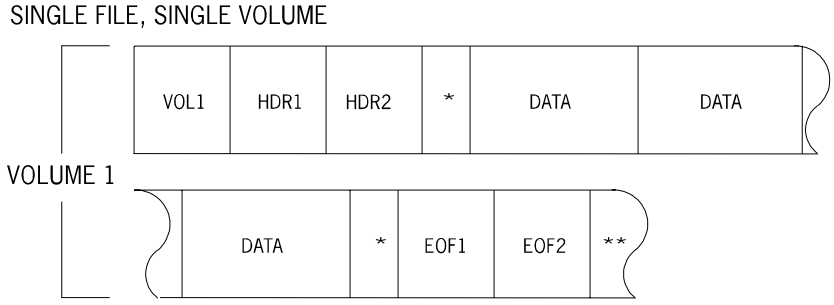


Figure E–2. ANSI69 Single-File, Single-Volume Format

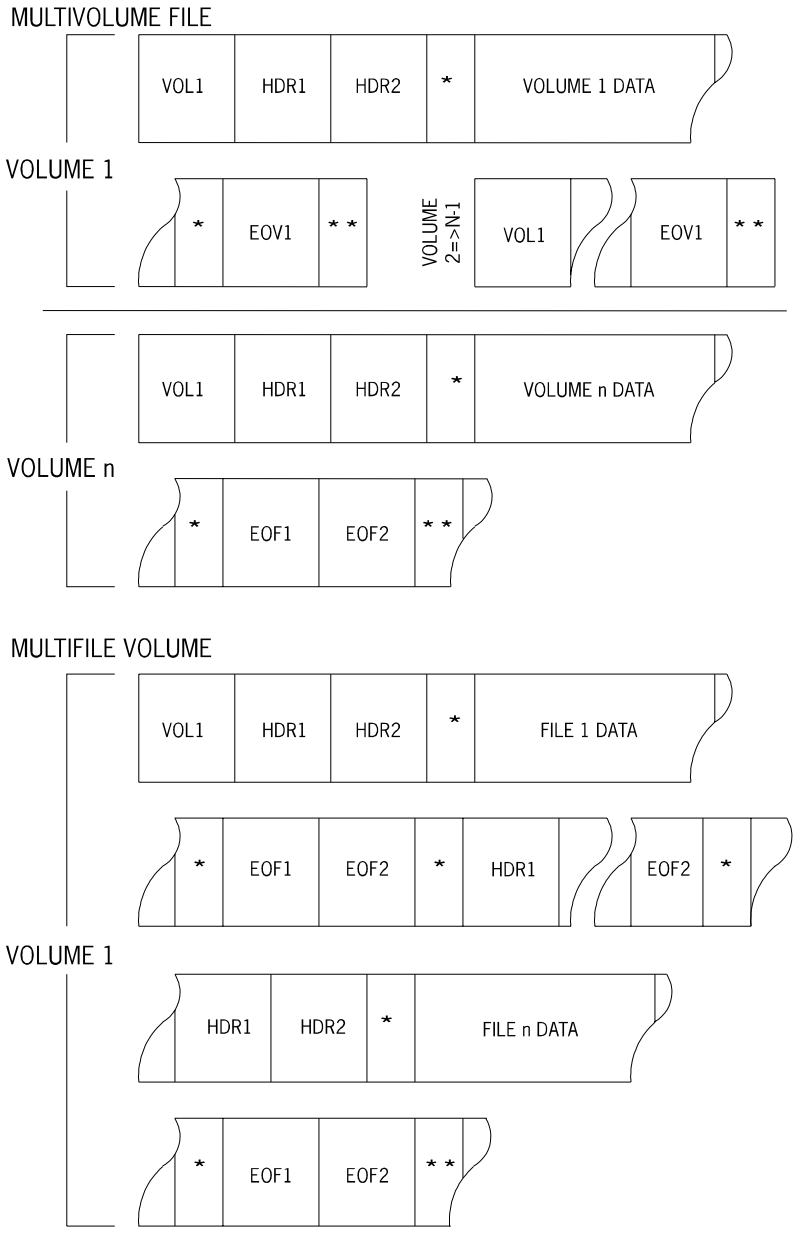


Figure E–3. ANSI69 Multivolume-File and Multifile-Volume Formats

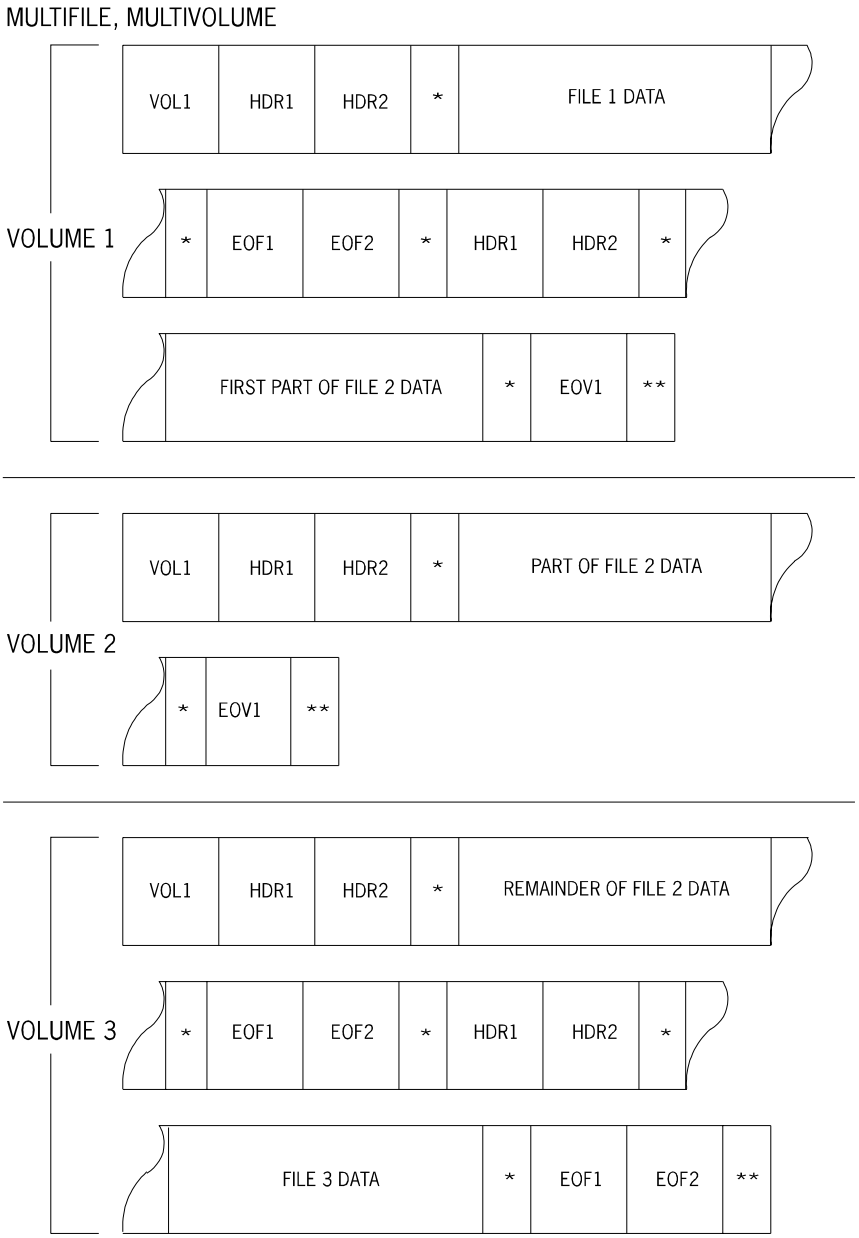


Figure E–4. ANSI69 Multifile, Multivolume Formats

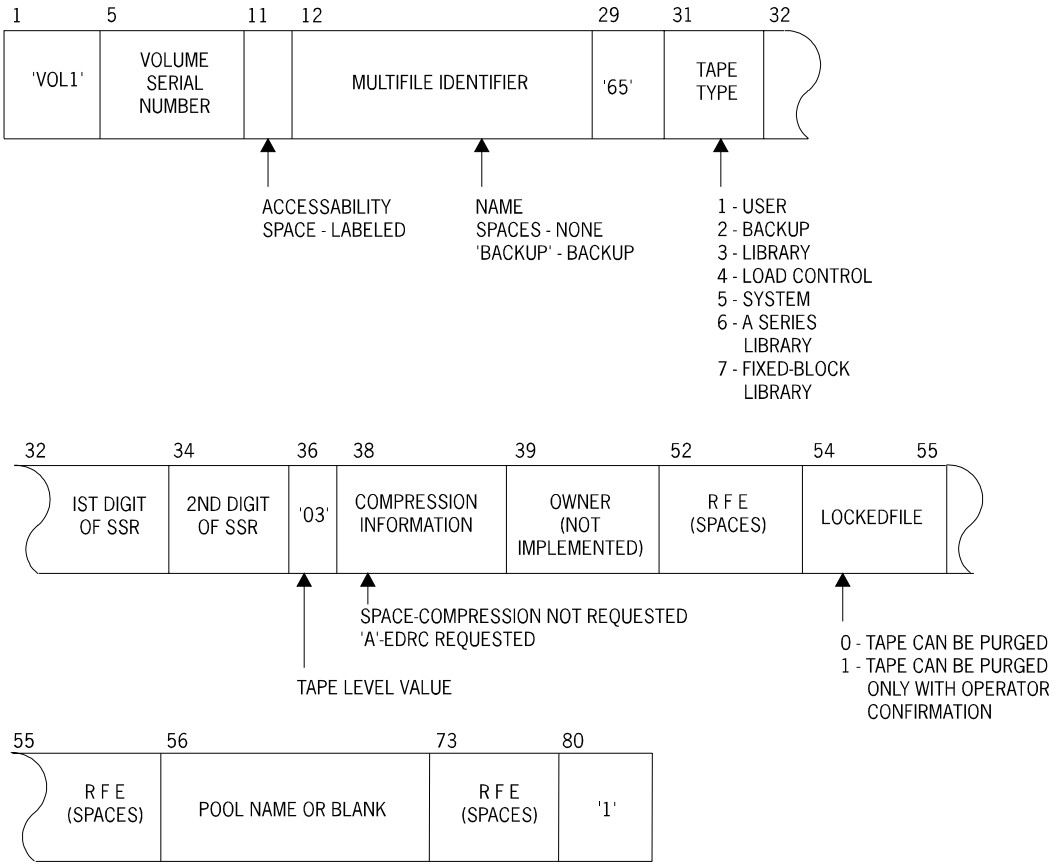


Figure E–5. ANSI69 Volume Header—Non-Scratch

Note: *The four characters of the SSR digit and SSR level number fields combine to identify the SSR level of the MCP that was active when the tape was created. For example, a tape created on a system running a 41.2 MCP stores the following values in characters 32 through 35 of its volume header: "0 4 0 1".*

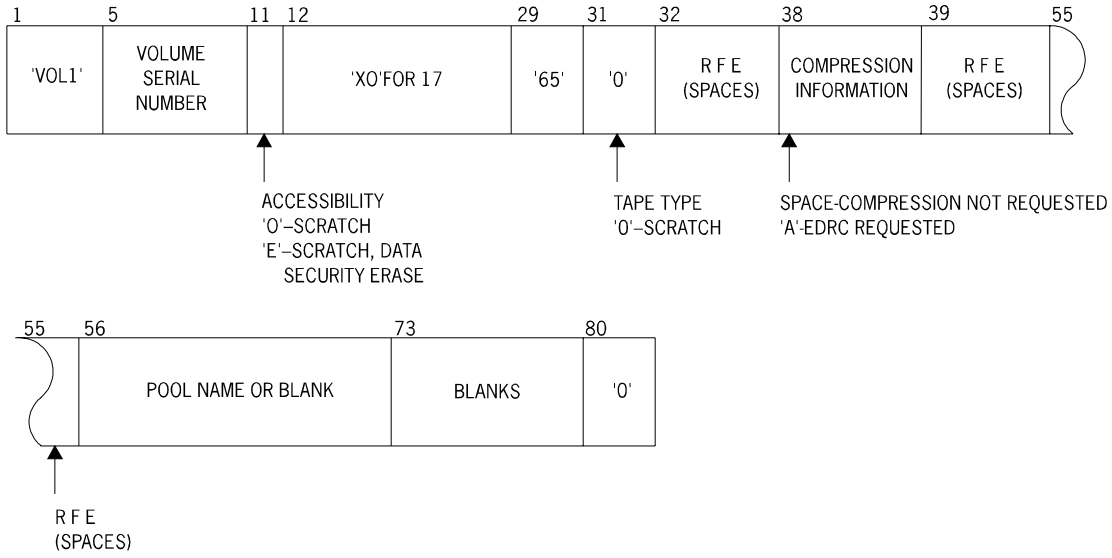


Figure E–6. ANSI69 Volume Header—Scratch

Note: *The MCP determines whether a tape is a scratch tape by examining character 80 of the volume header. If character 80 is "0," the MCP treats the tape as a scratch tape. In non-scratch ANSI69 volume headers, the character 80 is "1."*

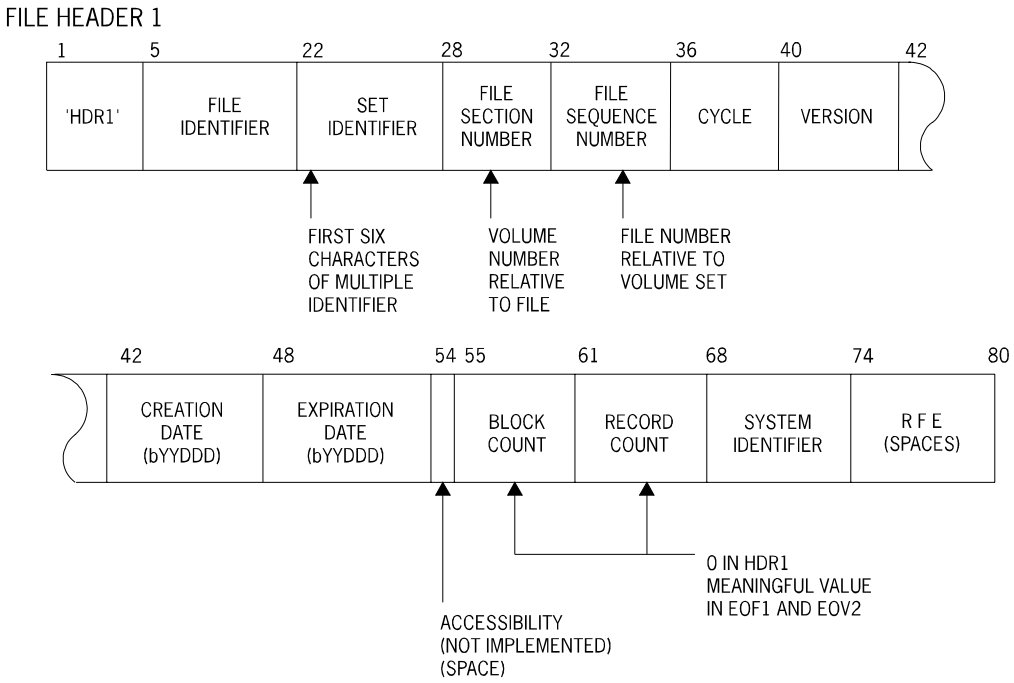


Figure E–7. ANSI69 File Header 1 Format

A first end-of-file label is the same as a first file header, except that the first four characters contain the letters EOF1 and the block and record count contain meaningful information.

An end-of-volume label is the same as the first end-of-file label, except that the first four characters contain the letters EOV1 and the block and record counts contain meaningful information.

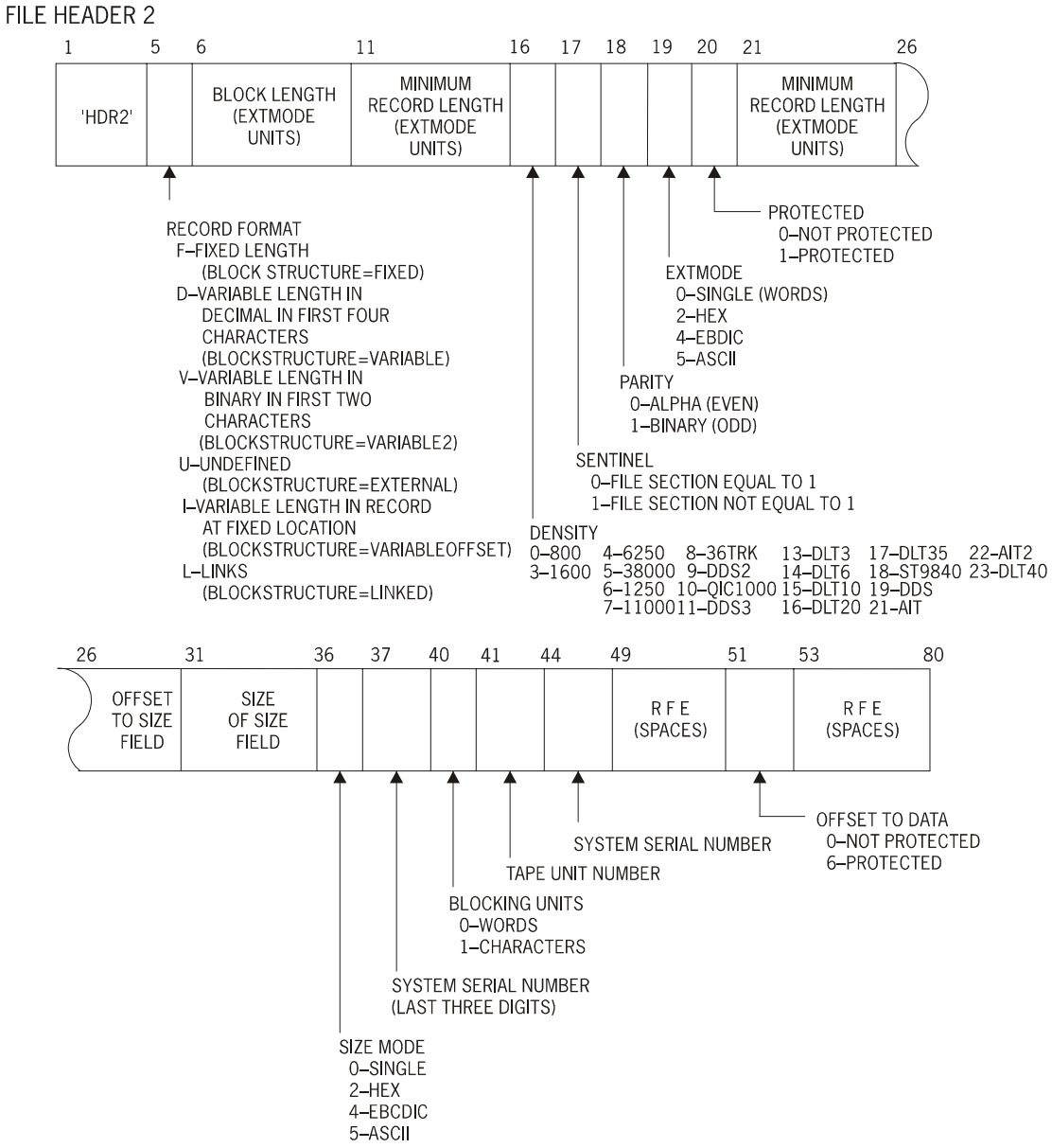


Figure E–8. ANSI69 File Header 2 Format

A second end-of-file label is the same as the second file header, except that the first four characters contain the letters EOF2.

Figure E–9 illustrates the formats of user header and trailer labels on ANSI69 tapes. User header labels can appear immediately after the HDR2 label record and before the tape mark, and user trailer labels can appear after either EOF2 or EOV1 and before the tape mark.

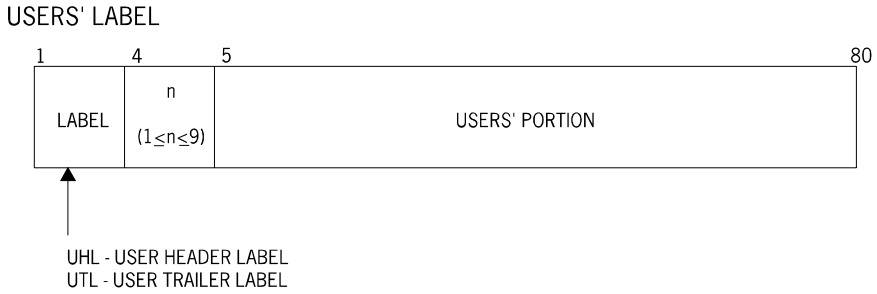


Figure E–9. ANSI69 User Header and Trailer Label Formats

Figure E–10 shows the format of an ANSI69 scratch tape.

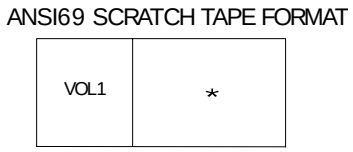


Figure E–10. ANSI69 Scratch Tape Format

B 3500 USASI Tapes

Figures E–11 and E–12 show the volume header format and the file header 1 format for B 3500 USASI tape labels.

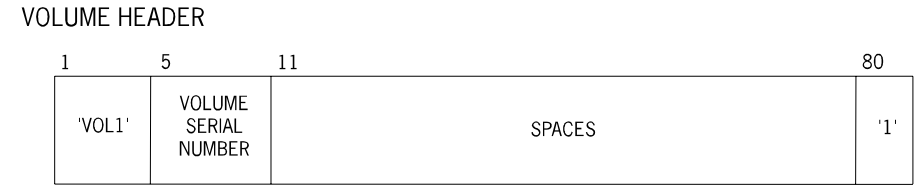


Figure E–11. B 3500 Volume Header Format

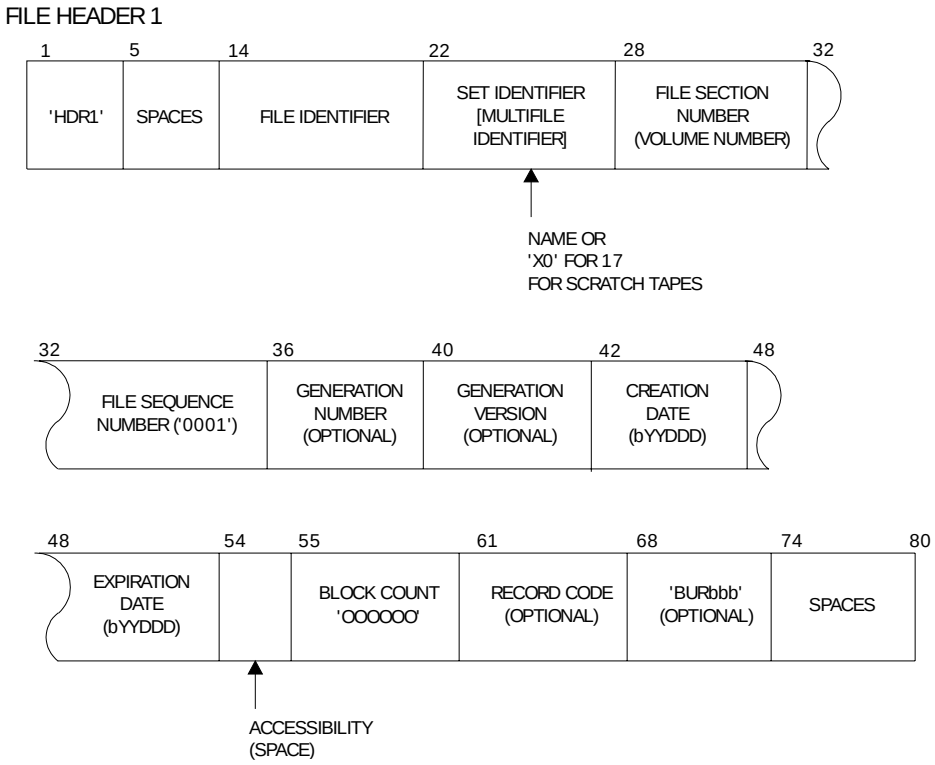


Figure E–12. B 3500 File Header 1 Format

ANSI87 Tapes

Figures E–13 through E–24 show the ANSI87 formats of the following types of tape labels:

- Multivolume file
- Multifile volume
- Volume header 1
- Volume header 2
- Volume header 3
- Volume Header 4
- Volume Header 5
- File header 1
- File header 2
- File header 3
- Scratch Tape

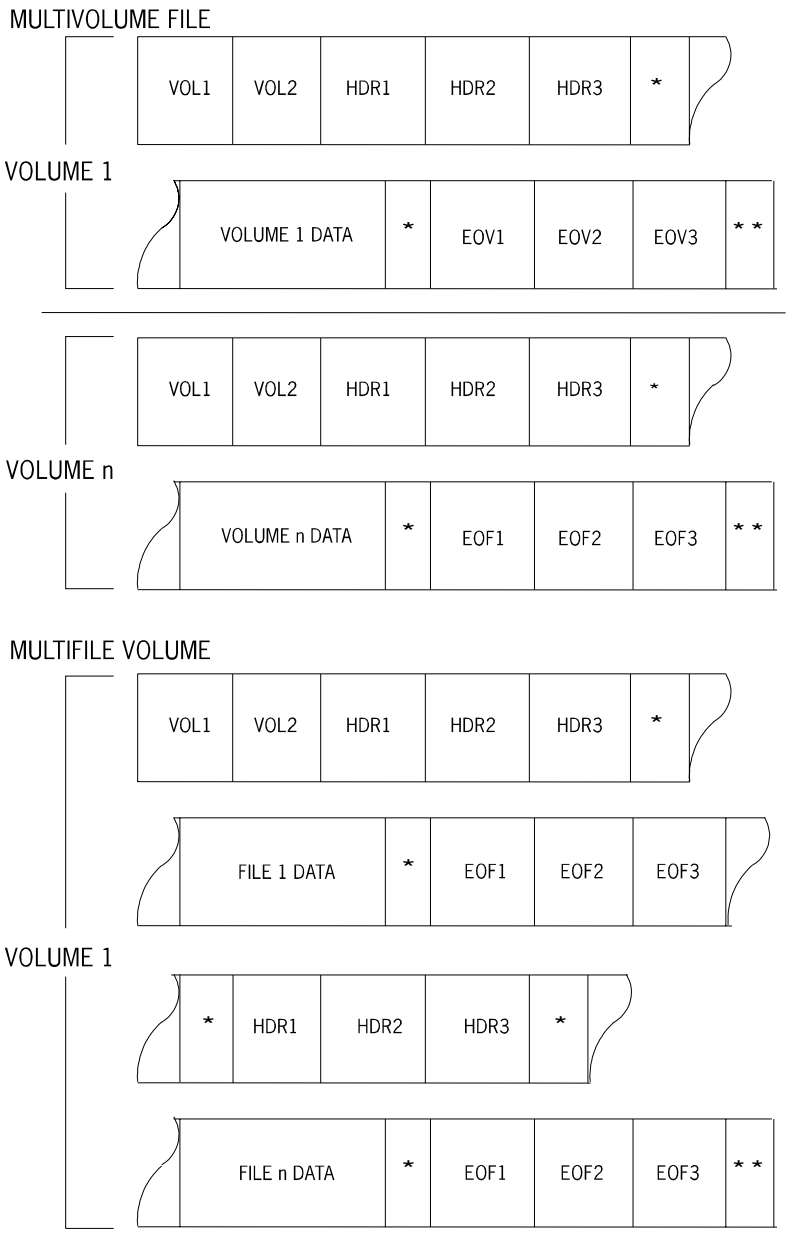


Figure E–13. ANSI87 Multivolume-File Format

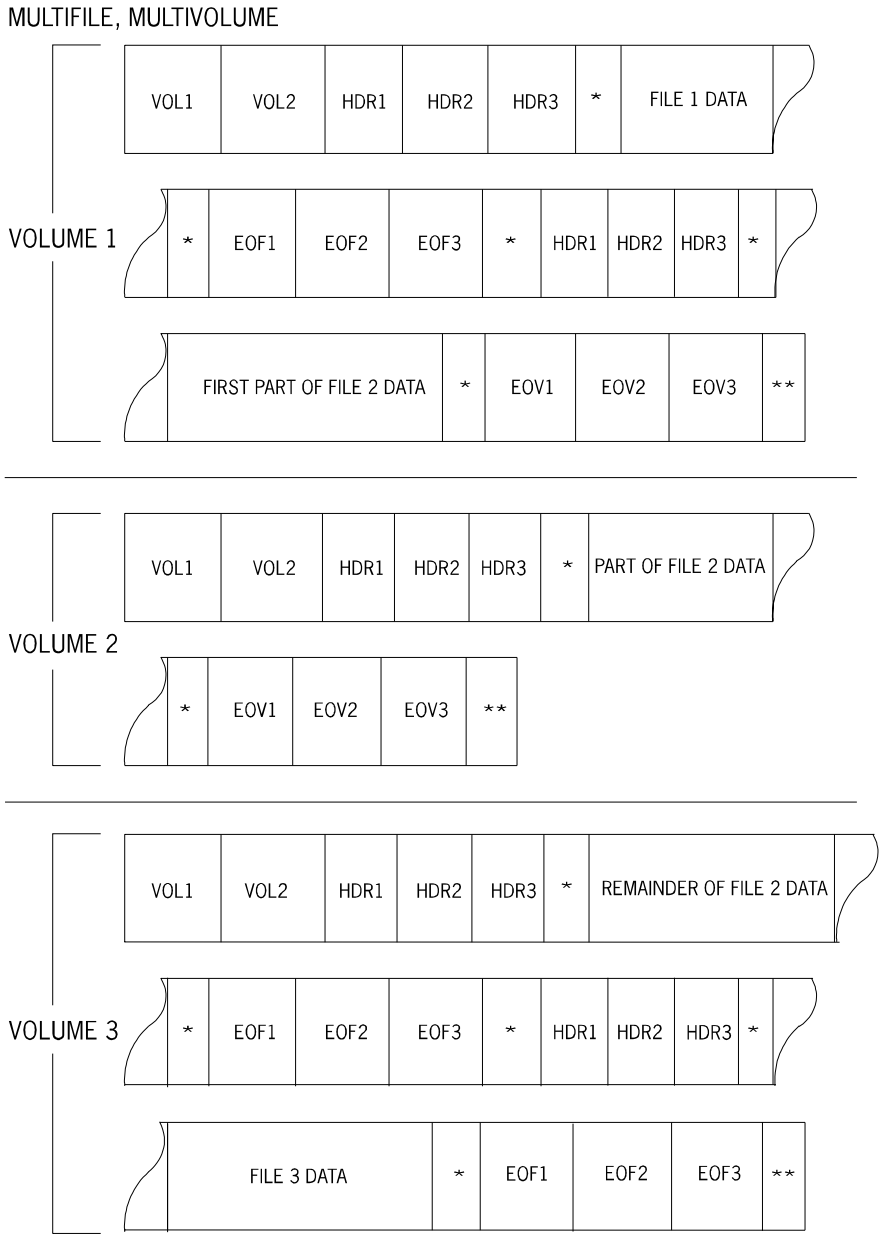


Figure E–14. ANSI87 Multifile, Multivolume Format

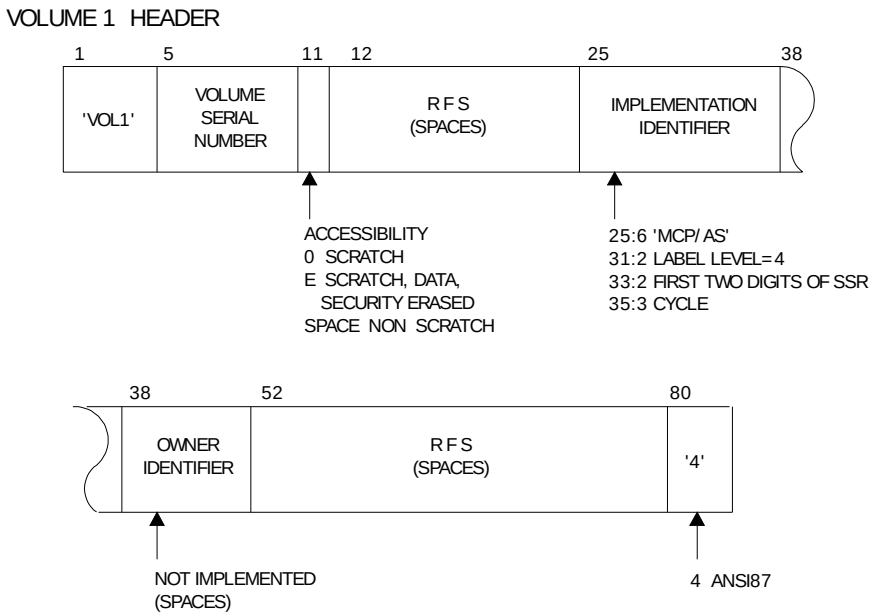


Figure E–15. ANSI87 Volume Header 1 Format

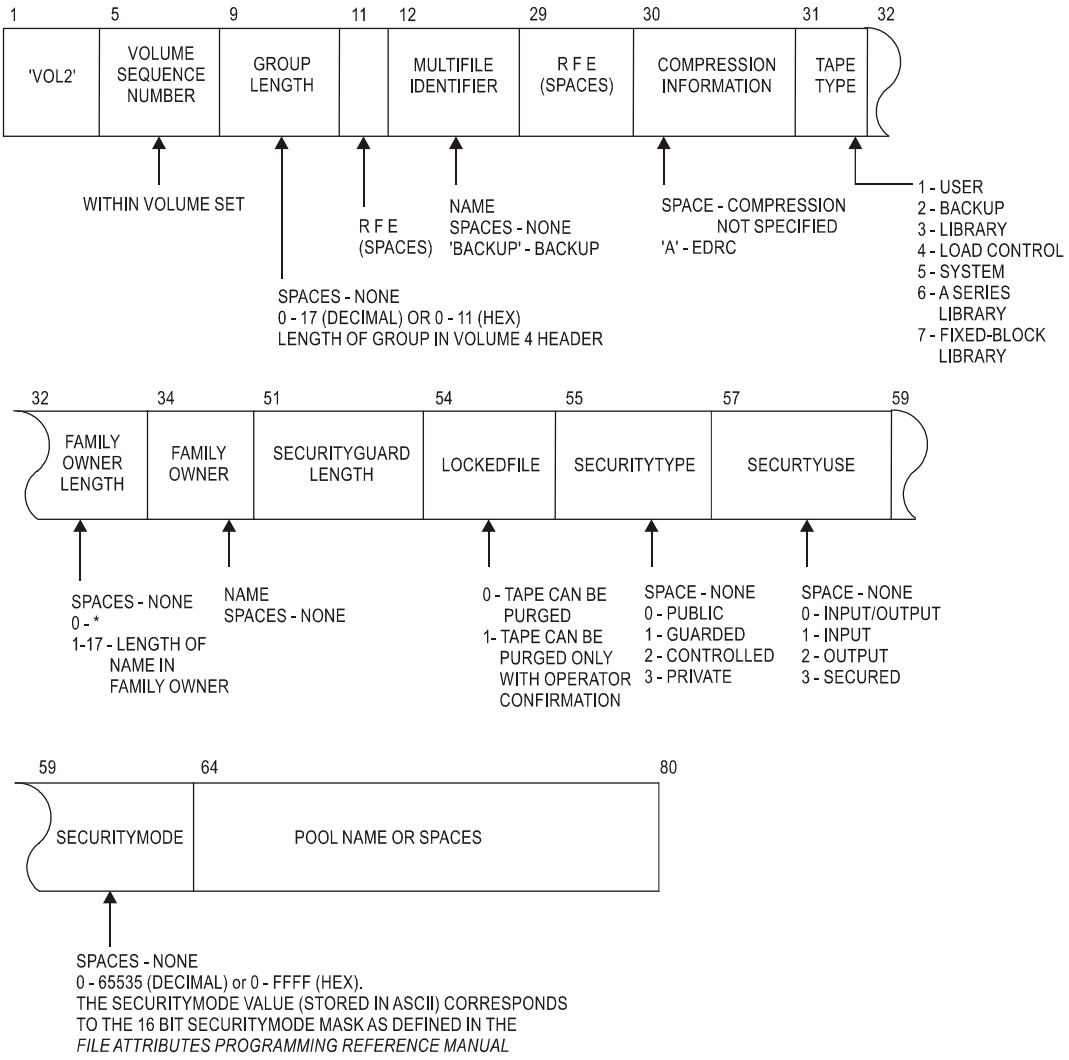


Figure E–16. ANSI87 Volume Header 2–Non-Scratch

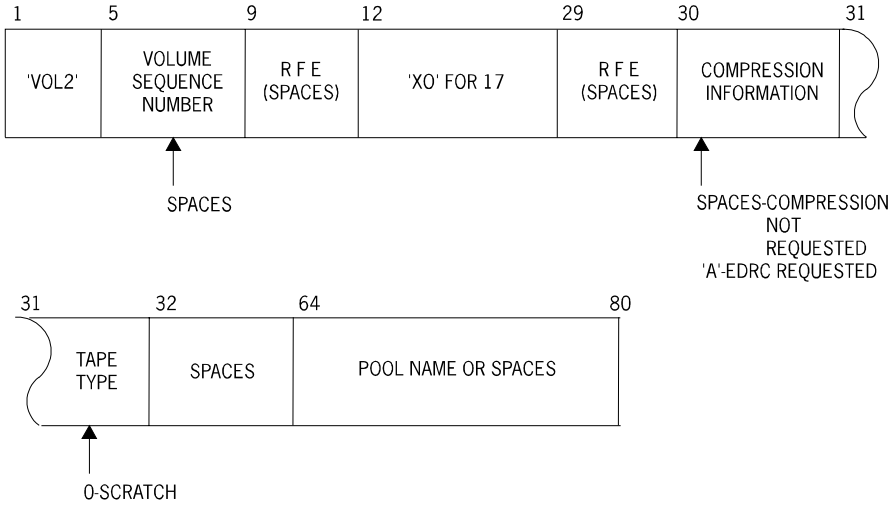


Figure E–17. ANSI87 Volume Header 2–Scratch

Note: *If the FAMILY OWNER LENGTH field (characters 32 and 33) of volume header 2 contains spaces or a zero, the MCP ignores the FAMILY OWNER field (characters 34–50).*

Figure E–18. ANSI87 Volume Header 3–Non-Scratch

Note: *If characters 51–53 of the volume 2 label are 0 but the system needs to store information in subsequent volume labels, such as the volume 5 label, then the volume 3 label will contain 76 blank characters. A tape includes a volume-header-3 label if a nonzero value appears in the SECURITYGUARD LENGTH field (characters 51–53) of volume header 2. A volume-header-3 label can vary in length from a minimum of 80 characters to a variable maximum number of characters that depends on the length of the SECURITYGUARD field. The length of the SECURITYGUARD field varies to accommodate the number of characters in the guard file title specified with the SECURITYGUARD file attribute.*

Figure E–19. ANSI87 Volume Header 4–Non-Scratch

Note: *If characters 9-10 of the volume 2 label are 0 but the system needs to store information in subsequent volume labels, such as the volume 5 label, then the volume 4 label will contain 76 blank characters. A tape includes a volume-header-4 label if a nonzero value appears in the GROUP LENGTH field (characters 9–10) of volume header 2. The tape includes an empty volume-header-3 label when a volume-header-4 label is required and the optional information contained in the volume-header-4 label is not set. This occurs when the GROUP attribute of the Tape volume is set but the SECURITYGUARD attribute has been specified.*

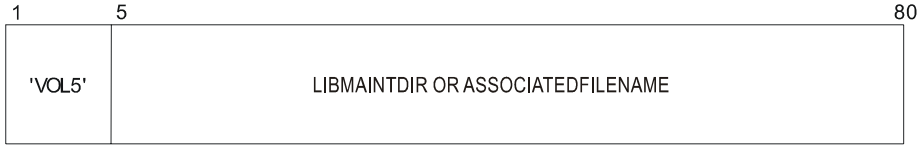


Figure E–20. ANSI87 Volume Header 5

Note: *If a library maintenance tape has a LIBMAINTDIR tape directory disk file associated with it or a non-library maintenance tape has an ASSOCIATEDFILENAME disk file associated with it, the name of that disk file in display form appears in the volume 5 label of that tape and any subsequent tape volumes in the case of multivolume library maintenance tapes.*

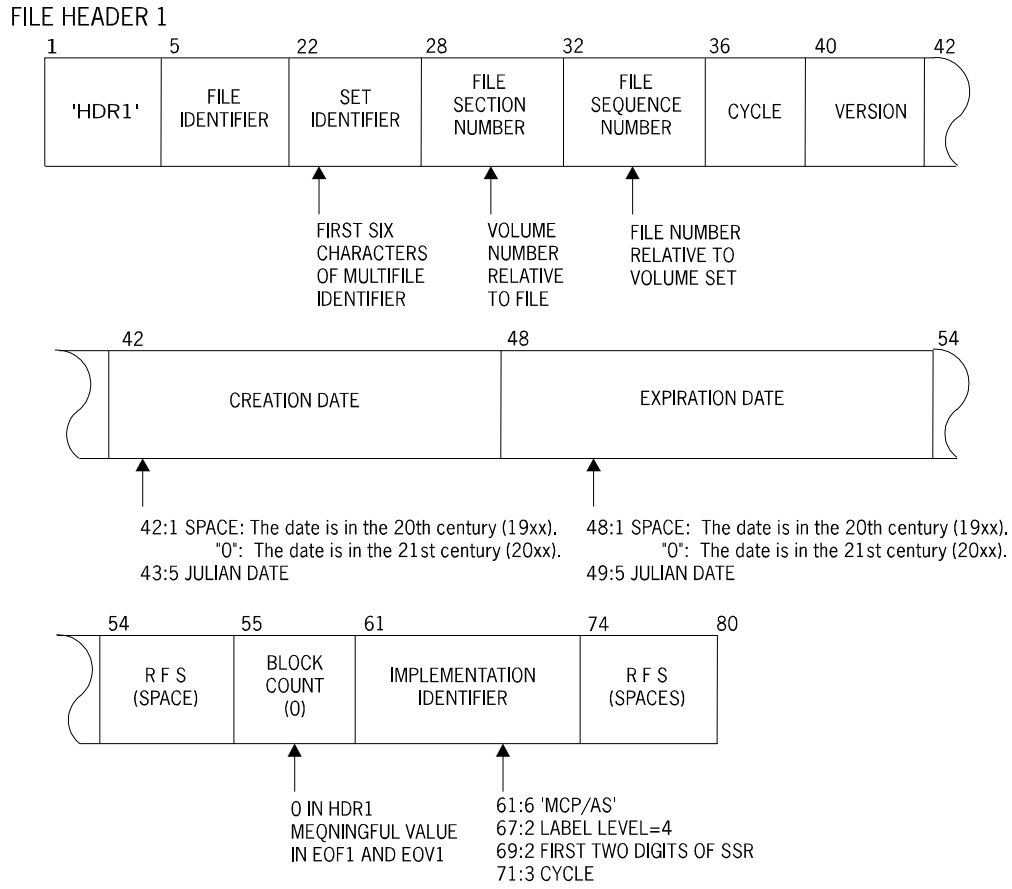


Figure E–21. ANSI87 File Header 1 Format

Notes:

- *A first end-of-file label is the same as a first file header, except that the first four characters contain the letters EOF1 and the block count contains meaningful information.*
- *A first end-of-volume label is the same as the first end-of-file label, except that the first four characters contain the letters EOV1.*

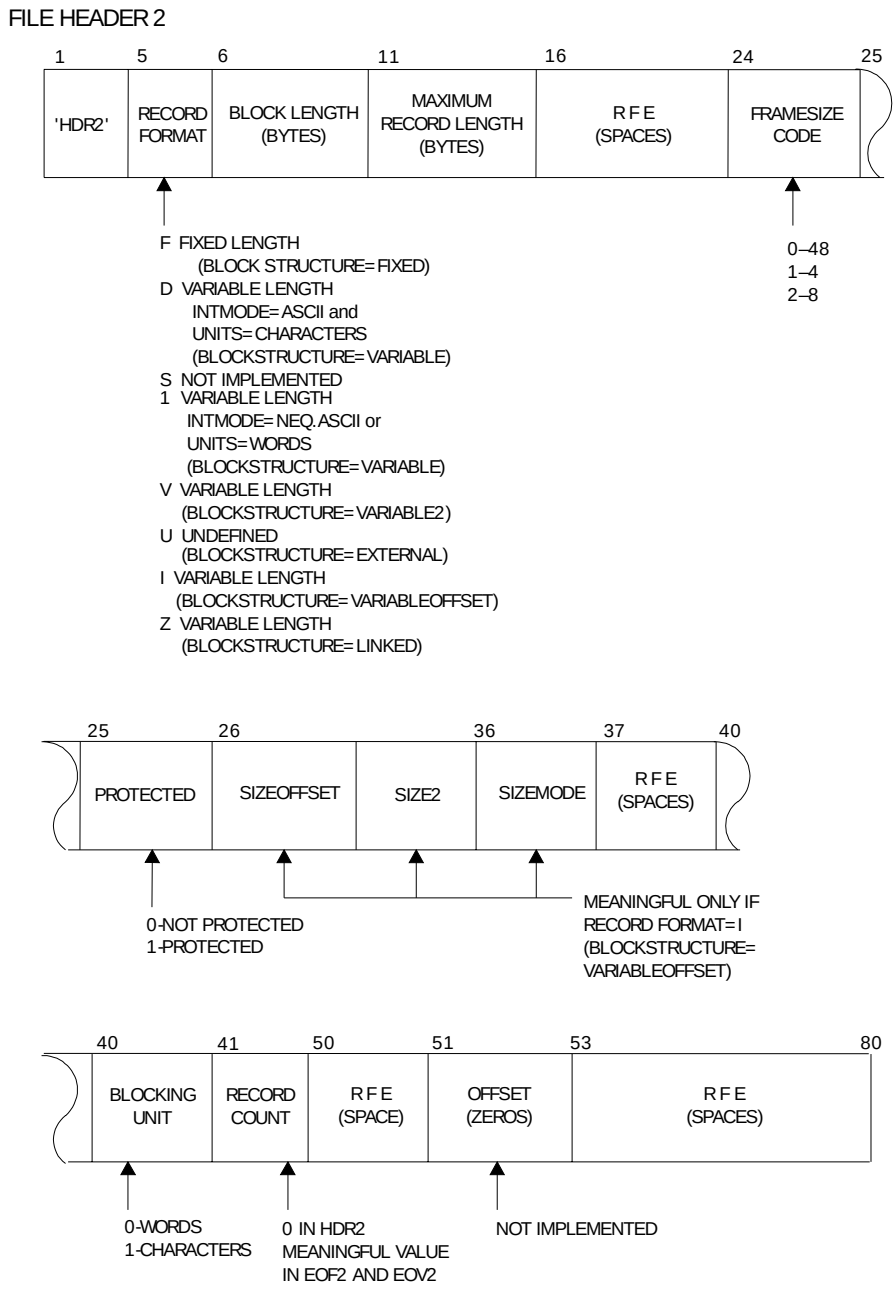


Figure E–22. ANSI87 File Header 2 Format

Notes:

- *A second end-of-file label is the same as a second file header, except that the first four characters contain the letter EOF2 and the record count contains meaningful information.*
- *A second end-of-volume label is the same as the second end-of-file label, except that the first four characters contain the letters EOV2.*

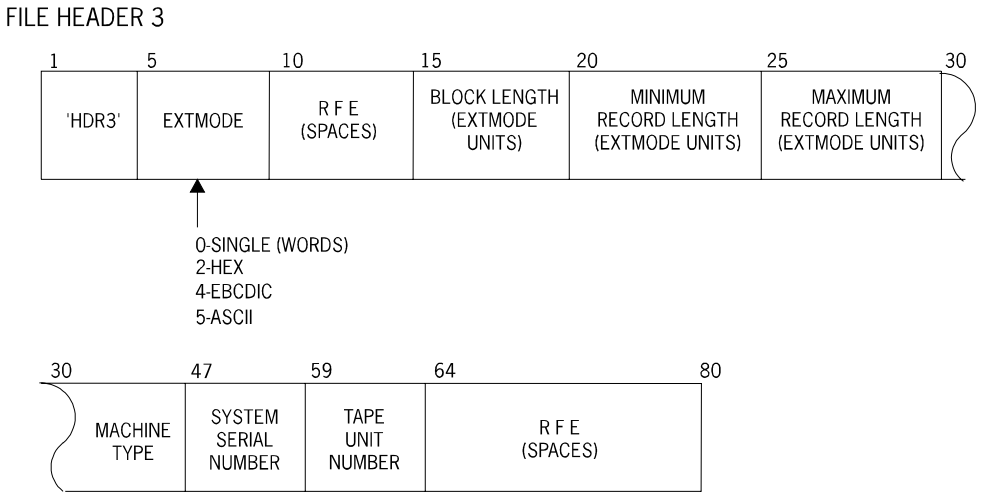


Figure E–23. ANSI87 File Header 3 Format

Notes:

- *A third end-of-file label is the same as a third file header, except that the first four characters contain the letter EOF3.*
- *A third end-of-volume label is the same as the third end-of-file label, except that the first four characters contain the letters EOV3.*

There are user header and trailer labels on ANSI87 tapes. User header labels can appear immediately after the HDR3 label record and before the tape mark, and user trailer labels can appear after either EOF3 or EOV3 and before the tape mark.

Figure E–24 shows the format of an ANSI87 scratch tape.

Figure E–24. ANSI87 Scratch Tape Format

B 5500 Tapes

Table E–1 describes the BCL characters that are common to all B 5500 tape labels.

Table E–1. BCL Characters for B 5500 Tape Labels

Word	Character (Word)	Character (Record)	Field (Description)
1	1–8	1–8	Must contain the word LABEL in upper case letters.
2	1	9	Must be 0.
2	2–8	10–16	Multifile ID.
3	1	17	Must be 0.
3	2–8	18–24	File ID.
4	1–3	25–27	Reel number (within file).
4	4—8	28–32	Date written (creation date).
5	1–2	33–34	Cycle number (to distinguish among identical runs on the same day).
5	3–7	35–39	Purge date (date this file can be destroyed).
5	8	40	Sentinel.
			1 = end-of-volume.
			0 = end-of-file.
6	1–5	41–45	Block count.
6–7	6–8 1–4	46–52	Record count.
7	5	53	Memory dump key. If value is 1 the memory dump follows the label.
7–8	6–8 1–2	54–58	Physical tape number.

ALGOL Files

Table E–2 describes the remaining information that is contained in a B 5500 tape label when the file is an ALGOL file.

Table E–2. B 5500 Tape Label for ALGOL Files

Word	Character (Word)	Character (Record)	Field (Description)
8	3	59	Blocking indicator
			3 = blocked
			0 = not blocked
8	4–8	60–64	Buffer size (number of words)
9	1–5	65–69	Maximum record size (number of words)
9	6–8	70–72	Zeros

COBOL Files

Table E–3 describes the remaining information that can be contained in a B 5500 tape label for a COBOL file.

Table E–3. B 5500 Tape Label for COBOL Files

Word	Character (Word)	Character (Record)	Field (Description)
8	3–8	59–64	Reserved for file-control routine. Not currently used.
9–end	1–end	65–end	Users' portion. Can be of any format desired by user and can be up to 8120 characters in length for tape files, up to 16 characters in length for card files, and up to 56 characters in length for printer files.

Appendix F
FORTRAN77 Programs

FORTRAN77 programs behave slightly differently from programs written in other languages.

The following rules for the TITLE attribute are implemented only for FORTRAN77 programs (FORTRAN77 refers to this attribute as *FILE*):

- If an attempt is made to set the file name with the TITLE attribute, then an implicit close is performed if the file is already open and the name of the file is different from the current name.
- If the name of the file remains the same and the file is open, then no error is given.

An implicit CLOSE operation results in the closing of the physical file and a title change in the logical file. Any subsequent OPEN operation creates a new physical file with the new file TITLE.

An attribute error is suppressed if the following attributes are modified by a FORTRAN77 program while the file is open, and if the resulting action does not change the value of the attribute:

- MAXRECSIZE
- NEWFILE
- PROTECTION
- UNITS

Appendix G
Controlling the Distribution of Application Programs

Your company can protect files that are sent to your customers by using the LICENSEKEY file attribute. If a file is assigned a LICENSEKEY value and that file is copied to a *conditioned* tape, the recipients of the tape must have the correct key and password combination in the SYSTEM/KEYSFILE on their systems before that file can be copied from the tape.

The SYSTEM/KEYSFILE file contains the keys associated with the software that your installation has purchased from Unisys. If a COPY operation is executed for a directory of files by using the "=" syntax, files that are guarded by keys that do not have matching keys in the SYSTEM/KEYSFILE file are not copied. If all files in the directory are not copied, an error message is issued. The SYSTEM/KEYSFILE file can be copied using the library maintenance ADD and COPY commands, and can be removed using the REMOVE command. The SYSTEM/KEYSFILE file can be accessed only by the operating system and resides on the halt/load unit.

The following procedure identifies the tasks you must perform to protect a file with a key:

1. Assign a disk file a license key by using the LICENSEKEY attribute. To make this assignment, the process performing the assignment must be privileged or must be run under a privileged usercode.

   The following ALGOL program extract assigns the *39-MYPRODUCT* key to the file F, as well as associating a password of *PASSWORDFORKEY* with the key:

   ```
   FILE F(KIND=DISK,TITLE = "MY/PROGRAM.");
   F.OPEN:=TRUE
   REPLACE F.LICENSEKEY BY "39-MYPRODUCT:PASSWORDFORKEY.";
   CLOSE(F);
   ```

   The key before the colon (:) can consist of one or more alphanumeric fields that contain no embedded blanks; each field can be separated by a hyphen (-). The password after the colon can consist of 10 to 17 characters with no embedded blanks. The total length of the key, the colon, and the password cannot exceed 150 characters.

   Note: *Use caution when assigning a key. If the password is forgotten or incorrectly entered, it cannot be changed and the file cannot be copied from a conditioned tape to disk.*

2. Copy the keyed file to a conditioned tape by entering the following WFL statement:

   ```
   COPY <file name> TO <tape name>; SW8=TRUE;
   ```

   The SW8=TRUE syntax makes the tape a conditioned tape.

3. Add the key to the SYSTEM/KEYSFILE by entering the following command:

   ```
   IK ADD <key>:<password>
   ```

   If you were adding the key assigned in the previous ALGOL example, you would use 39-MYPRODUCT:PASSWORDFORKEY.

Appendix H
Structure of Backup Files

Naming Conventions

The system assigns file names to backup files on disk according to the following default naming conventions. For printer backup files, the standard naming format is

```
*BD/<job number>/<task number>/<modified file name>
```

The standard prefix is *BD (backup disk). The asterisk is replaced by the usercode of the process that created the backup file if either of the following conditions is true:

- The PRINTDISPOSITION file attribute has a value of DONTPRINT.
- The system security option USERCODEDBACKUP has a value of TRUE. This option is controlled by the SECOPT (Security Options) system command.

The job number node records the mix number of the originating job or the session number of the originating CANDE or MARC session. The number is padded with zeros on the left to bring the total length to 7 digits.

The task number node records the mix number of the task that created the backup file. The number is padded with zeros on the left to bring the total length to 7 digits. The task number node is omitted if the backup file has been created directly by a job. Alternatively, multiple task number nodes appear if the backup file was created by a task at two or more levels removed from its ancestor job.

The format of the modified file depends on the setting of the system option MOREBACKUPFILES as follows:

- If MOREBACKUPFILES is FALSE, the modified file name is a single node consisting of 3 digits, ranging from 000 through 999, followed by the file name of the program output file. The 3-digit number is incremented each time a task or subtask opens a backup file and is intended to prevent duplicate file names.
- If MOREBACKUPFILES is TRUE, the modified file name is two nodes
  - The first node is a 12-digit number that is incremented each time a task or subtask opens a backup file and is intended to prevent duplicate file names.
  - The second node is the file name of the program output file.

Examples of Standard Names

The following example shows the standard backup file names generated by a job consisting of one task that produces three printer output files having the internal names LINE, PRNT, and OUT:

```
*BD/0004567/0002983/000LINE

*BD/0004567/0002983/001PRNT

*BD/0004567/0002983/002OUT
```

In the next example, the standard backup file names have been generated by a job consisting of four tasks, each of which produce one printer output file with the internal file name of LINE:

```
*BD/0002468/0005551/000LINE

*BD/0002468/0005552/000LINE

*BD/0002468/0005553/000LINE

*BD/0002468/0005554/000LINE
```

Overriding Standard Names

You can override the standard backup file-naming convention by using either the BDNAME task attribute to change the default prefix, or by using the USERBACKUPNAME file attribute and either the FILENAME or TITLE file attribute to rename the backup file entirely.

For example, the BDNAME task attribute is commonly used to make file identification easier; that is, you might want to replace the *BD prefix with a prefix such as the name of the program, a usercode, or a department or section name.

The most common use of the USERBACKUPNAME and FILENAME or TITLE file attributes is to rename a backup file when you save it. For more information, refer to the discussions of the BDNAME task attribute, and the USERBACKUPNAME, FILENAME, and TITLE file attributes in Section 5.

Naming Tape Files

Backup files created on magnetic tape are labeled as BACKUP/<file name>, where the file name is the name of the file as declared in the program that created it.

File Format

The following file format applies only to backup files with a FILEKIND value of BACKUPPRINTER and not to backup files with a FILEKIND value of PRINTFILE.

A backup file is constructed of fixed-length records, each of which is considered a *backup block*. Each backup block consists of a variable number of variable-length *backup records*. All but the first of these backup records represent the printer file records written by the program that created the backup file. The first record of the first block is the backup file control record. Only the first block of the file begins with a control record. The last two words of each backup block contain the number of printer records in the block and the number of the first record in the block. A backup file logical record, including the control word, must not exceed 256 words in length.

From the point of view of the I/O subsystem, each backup block is a record of a FIXED BLOCKSTRUCTURE file with a FRAMESIZE of 48 and a FILESTRUCTURE of ALIGNED180 or BLOCKED. Currently, the values of the MAXRECSIZE and BLOCKSIZE file attributes for a backup file are both 300, but either might change in the future. A program is insulated from changes to these attributes if it opens the backup file with DEPENDENTSPECS = TRUE and interrogates the MAXRECSIZE attribute to determine the size of a backup block.

Each backup record, including the control record, begins with a control word that contains information about the record and a pointer to the next record. The information in the control word cannot be printed. The control word is optionally followed by one or more words of data. The data portion of a block ends with a terminal control word composed entirely of binary zeros, which indicates that no more valid records appear in that block.

Records cannot be split over a backup block boundary, so a backup block can have unused words. For instance, if a record ends at word 290 of a block and the next record is 12 words long, the 12-word record is written as the first record of the next block. Word 291 becomes the terminal record.

Figure H–1 shows the typical structure of a backup file. In the diagram, CR means control record, R1 and so on mean data records, and W means a word. The rows of asterisks are unused words, and the number signs (#) indicate the division between blocks. Note that only the first block contains a control record.

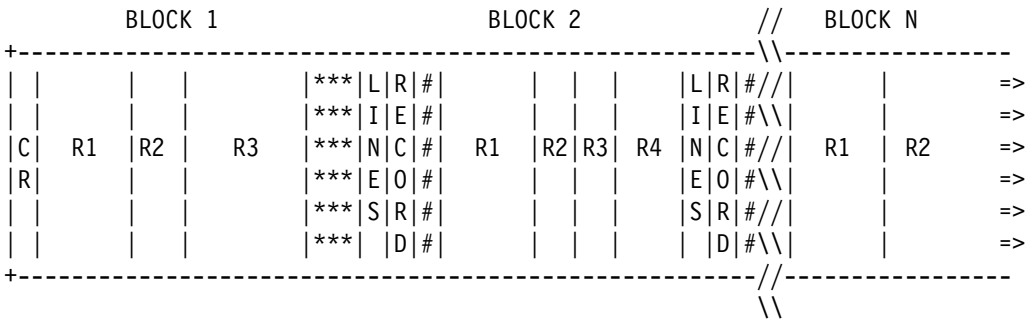


Figure H–1. Structure of a Backup File

Figure H–2 shows the structure of a backup block. The block shown is Block 1 of a backup file; all other backup blocks in the backup file begin with a data record.

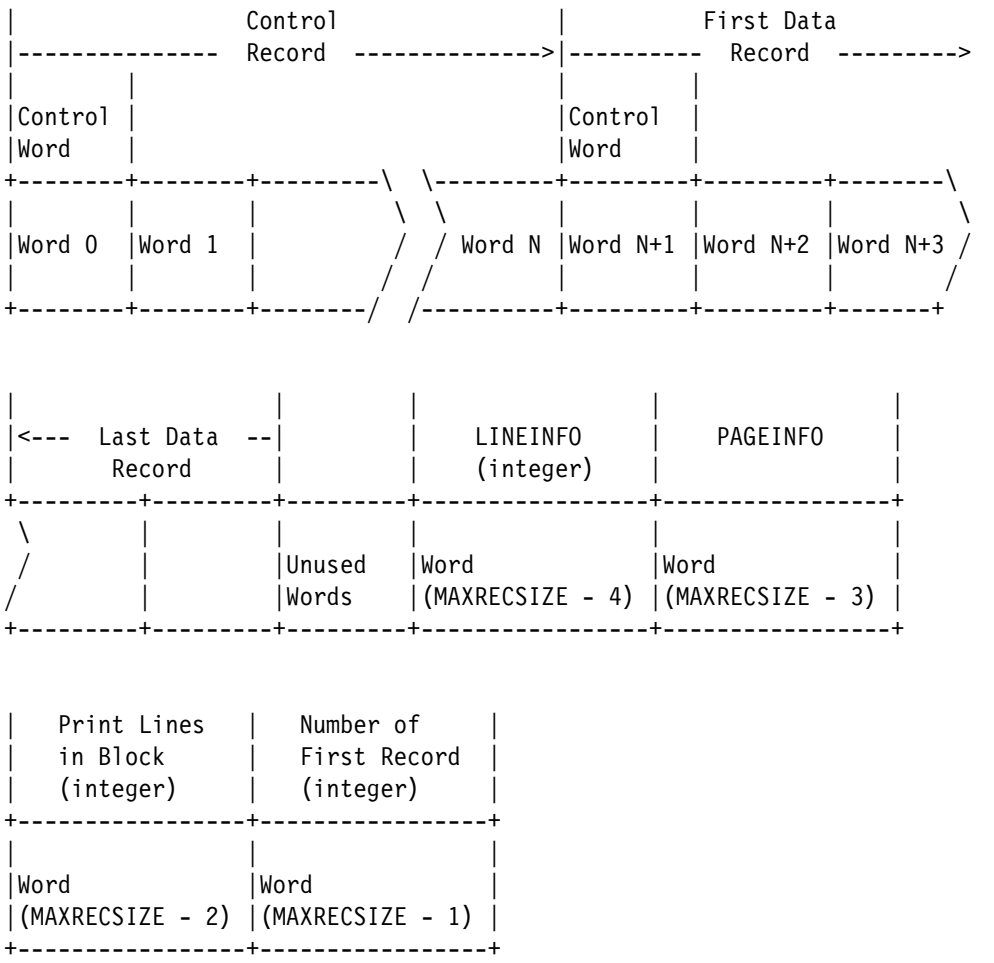


Figure H–2. Format of a Backup Block

The LINEINFO word contains the number of lines required to print the contents of the backup file through the current block. An overprinted record is counted as a single print line.

The PAGEINFO word contains two fields. Field [47:09] contains the maximum number of lines required between two skip-to-channel-1 operations (through the current block), and field [38:39] contains the number of skip-to-channel-1 operations encountered so far.

The backup file control record (the first record in the file) is at least 13 words long. The control record stores the values of the permanent file attributes and other information about the file. Table H–1 shows the format of the backup file control record.

Table H–1. Format of the Backup File Control Record

Word	Contents
0	Control word
1	Block character control word
2	Logical file kind word
3	Path control word
4	TRAINID file attribute
5	EXTMODE file attribute
6	Label type
7	I/O mask
8	Job number
9	Level
10	TRIMBLANKS file attribute
11	NOTE, FORMID, and TRANSFORM file attributes
12 –N	Values of variable-length attributes

Control Record Word Descriptions

A word can consist of one or more fields. Each field in a word is described by two numbers separated by a colon and enclosed in brackets. The first number specifies the starting bit of the field and the second number specifies the number of bits composing the field.

The diagram in Figure H–3 shows how bits are numbered within a word.

```
+---+---+---+---+---+---+---+---+---+---+---+---+
| 47| 43| 39| 35| 31| 27| 23| 19| 15| 11| 7 | 3 |
+---+---+---+---+---+---+---+---+---+---+---+---+
| 46| 42| 38| 34| 30| 26| 22| 18| 14| 10| 6 | 2 |
+---+---+---+---+---+---+---+---+---+---+---+---+
| 45| 41| 37| 33| 29| 25| 21| 17| 13| 9 | 5 | 1 |
+---+---+---+---+---+---+---+---+---+---+---+---+
| 44| 40| 36| 32| 28| 24| 20| 16| 12| 8 | 4 | 0 |
+---+---+---+---+---+---+---+---+---+---+---+---+
```

Figure H–3. Diagram of Numbering Bits within a Word

For example, assume you have the field [43:03]. The starting bit of the field is 43, and the field is comprised of three bits. Using the diagram in Figure H–3, you find that the field extends from bit 43 to bit 41.

Each word in a backup file control record is described on the following pages. Each word description contains the fields that are used and their locations, the possible values for each field (if applicable), and an explanation of the contents of each field. In the diagrams of each word, areas filled with reverse slashes (\) are reserved.

Table H–2 and Figure H–4 describe the control word (word 0). This word contains information that cannot be printed.

Table H–2. Fields of the Control Word (Word 0)

Field	Value	Contents
[43:01] (A)	0	Data is transferred from memory.
	1	Data is not transferred from memory.
[40:01] (B)	0	Data is to be sent to the printer.
	1	Data represents a dynamic change to the PAGECOMP file attribute.
[35:04] (C)		Contains the channel number to skip to after printing.
[31:02] (D)		The number of blank lines to add after printing.
[19:03] (E)	0	The record to be printed consists of complete words.
	nonzero	Character count residue of the current record. Value range is 1 through 5.
[16:17]		Word count for data in the current record in full words, not counting the control word.

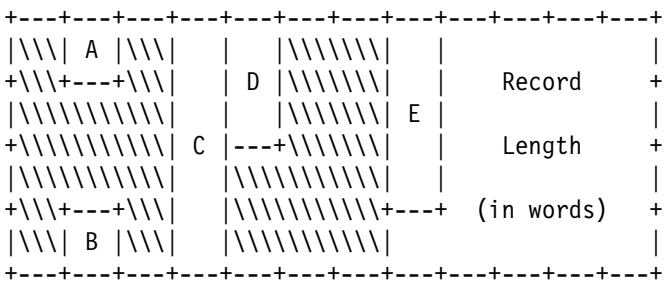


Figure H–4. Diagram of the Control Word (Word 0)

Table H–3 and Figure H–5 describe a block character control word (word 1) of the control record.

Table H–3. Fields of the Block Character Control Word (Word 1)

Field	Value	Contents
[47:08]	nonzero	Index to FORMID file attribute
	0	No FORMID
[39:08]		Index to NAME task attribute
[31:08]		Index to CHARGECODE task attribute
[23:08]		Index to USERCODE task attribute. This is the USERCODE value of the process that created the backup file. This value could be different from the USERCODE of the originating job or session. This value could also be different from the usercode of the print request that this backup file is part of.
[03:01] (S)	1	The file is a backup disk file, and the label type of the file is STANDARD.
	0	Label entries are not present.
[02:01] (J)	1	A JOBNUMBER is to be printed.
	0	A JOBNUMBER is not to be printed.
[01:01] (R)	1	The origin is remote.
	0	The origin is not remote.
[00:01] (V)	1	This control word is valid.
	0	This control word is not valid.

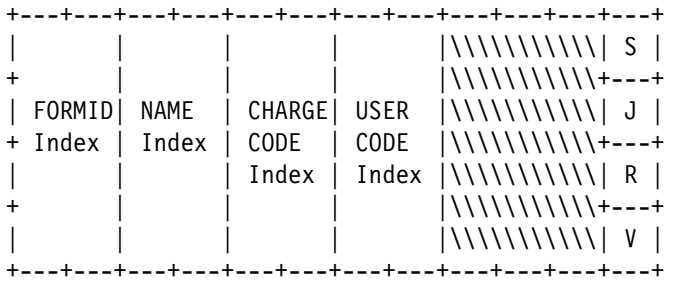


Figure H–5. Diagram of the Block Character Control Word (Word 1)

Table H–4 and Figure H–6 describe the logical file kind word (word 2) of the control record.

Table H–4. Fields of the Logical File Kind Word (Word 2)

Field	Value	Contents
[47:16]		The value of the PRINTERKIND file attribute.
[16:01] (T)	1	Transform is BYFUNCTION.
	0	Transform is BYTITLE.
[15:01] (a)		Validity bit for ALIGNMENT file attribute:
	1	ALIGNMENT attribute is explicitly set at file creation.
	0	ALIGNMENT attribute was never set.
[14:01] (b)		Validity bit for BANNER file attribute:
	1	BANNER attribute was explicitly set at file creation.
	0	BANNER attribute was never set.
[13:01] (A)		The value of the ALIGNMENT file attribute:
	1	The file requires alignment at printing time.
	0	The file does not require alignment at printing time.
[12:01] (B)		The value of the BANNER file attribute:
	1	Banner is printed before the file.
	0	Banner is not printed.
[11:04](P)		Formerly used for the value of the PRINTERKIND file attribute. See field [47:16] for new location.
[07:08]		The unit type of the backup file:
	7	Line printer.

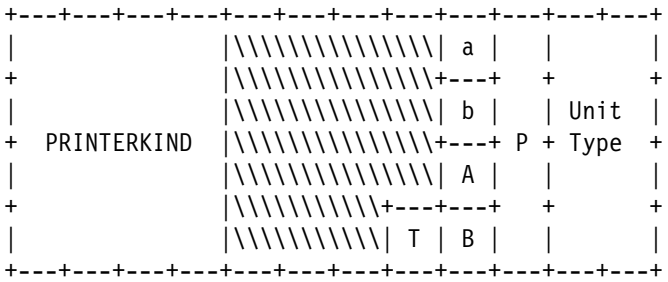


Figure H–6. Diagram of the Logical File Kind Word (Word 2)

Table H–5 and Figure H–7 describe the path control word (word 3) of the control record.

Table H–5. Fields of the Path Control Word (Word 3)

Field	Value	Contents
[45:06]		DESTMCSF. The number of the destination message control system (MCS).
[39:01] (D)		DESTISREMOTEF. Destination is a remote unit.
[38:15]		DESTUNITF. Destination unit number/Logical Station Number (LSN).
[21:06]		The number of the MCS controlling the originating LSN, if the origin was a remote unit.
[15:01] (R)	1	The originating unit is a remote unit.
	0	The originating unit is not a remote unit.
[14:15]		The number of the originating unit of the job that created the file. If the origin was a remote terminal, the number is an LSN. Otherwise, the number is a logical unit number.

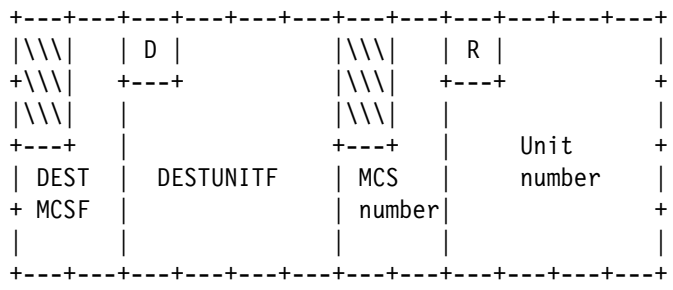


Figure H–7. Diagram of the Path Control Word (Word 3)

Table H–6 describes words 4 through 9 of the control record.

Table H–6. Fields of Words 4 through 9 of the Control Record

Word	Field	Value	Contents
4	Whole word	Nonzero	The value of the TRAINID file attribute, which specifies a train-printer character set
		0	The file is generated as if for a drum printer.
5	Whole word		The value of the EXTMODE file attribute
6	Whole word		The value of the LABEL file attribute
7	Whole word		The I/O mask for page specifications, with bits [09:01] and [00:01] being of interest. (Setting the PAGESIZE attribute changes this word to a nonzero value.)
	[09:01]	1	Suppress end-of-page (EOP) or end-of-file (EOF) handling.
		0	Handle EOP or EOF.
	[00:01]		Exception bit
8	Whole word		Job number of the job being printed
9	Whole word		The level number of the backup file format. On SSR 45.1 the level is 9. If the format changes, the level number is incremented.

Table H–7 and Figure H–8 describe word 10 of the control record.

Table H–7. Fields of Word 10 of the Control Record

Field	Value	Contents
[47:08]		Index to the input string of the transform function
[39:16]		MAXRECSIZE of the backup file, in EXTMODE characters
[23:08]	nonzero	Index to the SOURCENAME task attribute, stored in standard form
	0	Either the task did not have a specified SOURCENAME, or it could not fit in block zero
[00:01] (T)		Value of the TRIMBLANKS file attribute:
	0	TRIMBLANKS file attribute was FALSE when the backup file was created.

Table H–7. Fields of Word 10 of the Control Record

Field	Value	Contents
	1	TRIMBLANKS file attribute was TRUE when the backup file was created. Trailing blank words are omitted.

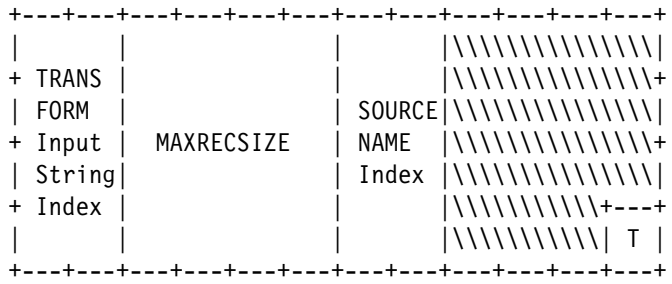


Figure H–8. Diagram of Word 10 of the Control Record

Table H–8 and Figure H–9 describe word 11 of the control record.

Table H–8. Fields of Word 11 of the Control Record

Field	Value	Contents
[47:08]	nonzero	Index to the value of the NOTE file attribute
	0	The NOTE file attribute value was not specified when the file was created.
[39:08]		Index to the value of the ALIGNFILE file attribute
[31:08]	nonzero	Index to the value of the ACCESSCODE task attribute
	0	The ACCESSCODE task attribute was not specified when the file was created.
[23:08]	nonzero	Index to the value of the FORMID file attribute. This index points to the full FORMID. If the FORMID is less than or equal to three words long, the FORMID at this index is the same as the FORMID at the index specified in word 1. If the FORMID is greater than three words long, the FORMID at the index in word 1 is truncated to the first three words.
	0	A FORMID was not specified when the file was created.
[15:08]	nonzero	Index to the TRANSFORM entry point name
	0	A TRANSFORM was not specified when the file was created.
[07:08]	nonzero	Index to the name of the TRANSFORM library
	0	The TRANSFORM library name was not specified when the file was created.

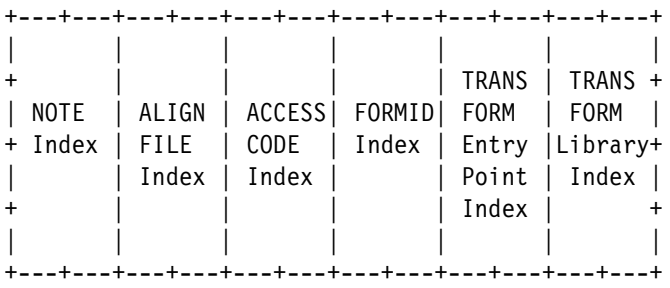


Figure H–9. Diagram of Word 11 of the Control Record

Word 12 of the control record is the beginning of the area that contains the values of the variable-length file and task attributes FORMID, NAME, CHARGECODE, USERCODE, NOTE, TRANSFORM, and so on. The actual locations of these are indicated by words 1 and 11. All these are stored in standard form.

Table H–9 and Figure H–10 describe word 12 of the control record.

Table H–9. Fields of Word 12 of the Control Record

Field	Value	Contents
[16:01	0	CCSVERSION value is positive.
	1	CCSVERSION value is negative.
[15:16}		The value of the CCSVERSION file attribute.

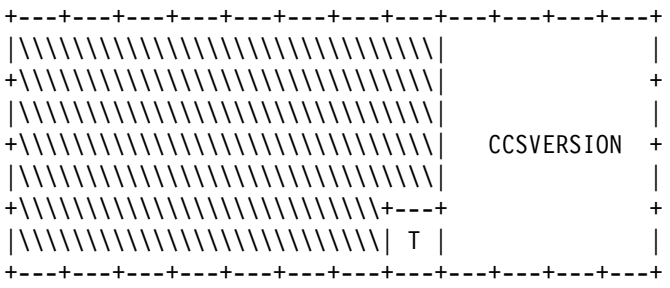


Figure H–10. Diagram of Word 12 of the Control Record

Appendix I
Related Product Information

The following documents provide information that is directly related to the primary subject of this guide.

***MCP/AS Interactive Datacomm Configurator (IDC) Operations Guide* (8600 1880)**

This guide explains how to use IDC, a menu-driven utility used to define and modify data communications networks. It provides information on configuring a data communications network using the IDC menu system and basic constructs, and provides reference information about the commands and attributes. This guide is written for individuals who have a basic knowledge of data communications concepts, but who might not know the physical characteristics of hardware devices in the network.

***MCP/AS KEYEDIOII Programming Reference Manual* (8600 0684)**

This manual describes the KEYEDIOII software. KEYEDIOII is the Unisys indexed sequential access method (ISAM) software for COBOL74 and Report Program Generator (RPG) programming languages. This manual is designed for applications programmers and analysts.

***MCP/AS Networking Operations Reference Manual, Volume 2: Reports and Log Messages* (3787 7925)**

This manual describes in reference format the Operations Interface (OI) reports and log messages for CNS, BNA, NCF, OSI, TCP/IP, and SNMP entities.

***MCP/AS POSIX User's Guide* (7011 8328)**

This guide describes the basic concepts of the POSIX interface, including process control and file management. It also describes specifically how the POSIX.1 interface is implemented and used on the enterprise server. This guide is written for programmers and any user who wants to understand the POSIX interface.

***MCP/AS Security Features Operations and Programming Guide* (8600 0528)**

This guide describes the security features available to users and provides instructions for their use. This guide is written for users who are responsible for maintaining the security of their individual programs and data.

MCP/AS System Administration Guide (8600 0437)

This guide provides the reader with information required to make decisions about system configuration, peripheral configuration, file management, resource use, and other matters related to system administration. This guide is written for users with some, little, or no experience who are responsible for making decisions about system administration.

Unisys e-@ction ClearPath Enterprise Servers Distributed Systems Services Operations Guide (8600 0122)

This guide describes the capabilities and features of distributed systems services and how to use them. It is intended for system operators, system administrators, and general computer users.

Unisys e-@ction ClearPath Enterprise Servers File Attributes Programming Reference Manual (8600 0064)

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 *I/O Subsystem Programming Guide* is a companion manual.

Unisys e-@ction ClearPath Enterprise Servers GETSTATUS/SETSTATUS Programming Reference Manual (8600 0346)

This manual explains how to use the various GETSTATUS and SETSTATUS calls in the Data Comm ALGOL programming language. This manual is written for experienced ALGOL programmers.

Unisys e-@ction ClearPath Enterprise Servers ClearPath MCP Migration Guide (8999 9197)

This guide provides a detailed overview of some of the new features and enhancements available with the ClearPath systems. Specifically, it describes the new features and enhancements that are installed in the MCP environment. It also describes changes in software behavior between the last software release and the current release. The document includes information on software migration, feature matrixes, special installation instructions for specific products, deimplementations and deimplementation warning messages, and known problems. This document is written for people who want to know how upgrading to the new release will affect their existing software.

Unisys e-@ction ClearPath Enterprise Servers MCP System Interfaces Programming Reference Manual (8600 2029)

This manual describes selected library objects exported from the MCPSUPPORT library, and describes the ARCHIVESUPPORT, BILLINGSUPPORT, and TAPEMANAGER libraries. This manual is written for system programmers who want to write programs that interface with the system software.

Unisys e-@ction ClearPath Enterprise Servers MultiLingual System Administration, Operations, and Programming Guide (8600 0288)

This guide describes how to use the MLS environment, which encompasses many products. The MLS environment includes a collection of operating system features, productivity tools, utilities, and compiler extensions. The guide explains how these products are used to create application systems tailored to meet the needs of users in a multilingual or multicultural business environment. It explains, for example, the procedures for translating system and application output messages, help text, and user interface screens from one natural language to one or more other languages; for instance, from English to French and Spanish. This guide is written for international vendors, branch systems personnel, system managers, programmers, and customers who wish to create customized application systems

Unisys e-@ction ClearPath Enterprise Servers Print System and Remote Print System Administration, Operations, and Programming Guide (8600 1039)

This guide describes the features of the Print System and provides a complete description of its command syntax. This guide is written for programmers, operators, system administrators, and other interactive users of Menu-Assisted Resource Control (MARC) and CANDE.

Unisys e-@ction ClearPath Enterprise Servers Security Administration Guide (8600 0973)

This guide describes system-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.

Unisys e-@ction ClearPath Enterprise Servers System Commands Operations Reference Manual (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.

Unisys e-@ction ClearPath Enterprise Servers System Configuration Guide (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.

Unisys e-@ction ClearPath Enterprise Servers System Log Programming Reference Manual (8600 1807)

This manual describes the format and contents of all the Major Type and Minor Type entries of the system log. It also contains information about controlling the log contents and about writing log analysis programs.

Unisys e-@ction ClearPath Enterprise Servers System Operations Guide (8600 0387)

This guide describes concepts and procedures required to operate most Unisys systems. Sections 1 and 2 contain information and procedures that can be done by novice operators. Section 3 contains operations and procedures that require more advanced operations experience. This guide is written for operators responsible for operating the enterprise server, especially operators with little or no experience.

Unisys e-@ction ClearPath Enterprise Servers System Software Utilities Operations Reference Manual (8600 0460)

This manual provides information on the system utilities BARS, CARDLINE, CDFORMAT, COMPARE, DCAUDITOR, DCSTATUS, DUMPALL, DUMPANALYZER, FILECOPY, FILEDATA, HARDCOPY, INTERACTIVEXREF, ISTUTILITY, LOGANALYZER, LOGGER, PATCH, PCDRIVER, PRINTCOPY, RLTABLEGEN, SORT, XREFANALYZER, and the V Series conversion utilities. It also provides information on KEYEDIO support, Peripheral Test Driver (PTD), and mathematical functions. This manual is written for applications programmers, system support personnel, and operators.

Unisys e-@ction ClearPath Enterprise Servers SYSTEMSTATUS Programming Reference Manual (8600 0452)

This manual documents the SYSTEMSTATUS intrinsic of the master control program. The SYSTEMSTATUS intrinsic provides information that can be used to efficiently monitor the performance of a running system. This manual is written for systems programmers.

Unisys e-@ction ClearPath Enterprise Servers TCP/IP Distributed Systems Services Operations Guide (8807 6385)

This guide explains how to use most of the TCP/IP services provided by the DSS products. These products include File Transfer Protocol (FTP), Telnet, Domain Name Services (DNS), and Time Synchronization. The guide also explains how to use the TCP/IP DSS debugging tools. This guide is written for system planners, system programmers, application programmers, and computer operators who use TCP/IP DSSs.

Unisys e-@ction ClearPath Enterprise Servers TCP/IP Implementation and Operations Guide (3787 7693)

This guide provides procedures for configuring a TCP/IP network using the NAU. Included is information on how to operate and troubleshoot a TCP/IP network.

Unisys e-@ction ClearPath Enterprise Servers Work Flow Language (WFL) Programming Reference Manual (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.

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

X

Y

Z


860000056-408