Simulated NOS/VE Program Interfaces

ARH 3125 17 July 80

EXTERNAL REFERENCE SPECFICATION

for

SIMULATED NOS/VE PROGRAM INTERFACES

Submitted: ____________________

Approved: ____________________

____

____

REVISION DEFINITION SHEET

REV	DATE	Original issue.
A	12/20/78	Original release
B	05/15/79	Changed for SES Release 12 implementation
C	07/17/79	Revised for NOS/170 "interchange" file support
D	01/28/80	Revised for new NOS/VE I/O, the NOS/170 INTERFACE, the SES release 13 SIMULATOR, and the new NOS/VE PROGRAM MANAGEMENT INTERFACES.

1.0 INTRODUCTION

The SIMULATED NOS/VE PROGRAM INTERFACES are provided for programs running on the CYBER 180 SIMULATOR. Their intent is to duplicate the CYBIL interfaces available on the CYBER 180 under NOS/VE.

1.1 APPLICABLE DOCUMENTS

NOS/VE ERS - PROGRAM INTERFACE (ARH3610, REV 5)
ERS for CYBIL I/O (ARH2739)
ERS for C180 SIMULATED ENVIRONMENT (ARH1729)

2.0 SIMULATED NOS/VE I/O INTERFACES

2.1 INTRODUCTION

The SIMULATED NOS/VE I/O system (AMP$) consists of a collection of procedures and common data type declarations which provide for the use of a subset of the NOS/VE I/O procedures described in the NOS/VE ERS - PROGRAM INTERFACE, section 6 on FILE MANAGEMENT.

The SIMULATED NOS/VE I/O system provides two distinct methods of performing I/O. The first method uses the CYBER 180 SIMULATOR to simulate NOS/VE programs. This method is hereafter refered to as "SIMULATED NOS/VE I/O". The second method provides for direct execution of the user program on the NOS/170 machine. The second method is hereafter called the "NOS/170 INTERFACE". SIMULATED NOS/VE I/O and NOS/170 INTERFACE together are hereafter referred to as the "SIMULATED NOS/VE I/O system".

SIMULATED NOS/VE I/O is available only as part of the CYBER 180 SIMULATOR on a NOS/170 machine. SIMULATED NOS/VE I/O is compatible only with programs compiled to generate CYBER 180 object code, such as the CYBER 180 CYBIL compiler (CI version) or CYBER 180 Assembly Language.

The NOS/170 INTERFACE is compatible only with programs written in the CYBIL language and compiled with the CC (NOS/170 object code) version of the compiler. The NOS/170 INTERFACE consists of a set of object time routines that emulate NOS/VE I/O calls. This interface is provided as a means of running programs directly on a NOS/170 machine and (optionally) using the CYBIL debugger.

It is assumed that the user is familiar with the CYBIL language and that the user has read the NOS/VE ERS - PROGRAM INTERFACE, section 6 on FILE MANAGEMENT.

2.1.1 USAGE OVERVIEW

The following block diagrams illustrates the generalized usage structure for this product. The SIMULATED NOS/VE I/O diagram is not intended to be complete in detail, rather it is meant as a generalized overview of the capability added to the CYBER 180 SIMULATOR by this I/O product. The method diagrammed for SIMULATED NOS/VE I/O executes the user program by itself on the CYBER 180 simulator. Alternatives to this "stand alone" method, such as using the NOS/VE operating system, are available for program simulation. Further details on these other methods is not herein provided.

The diagram of the NOS/170 INTERFACE shows how one may take a program to be run on NOS/VE and debug the program with the CYBIL debugger by compiling the program and running the program on the NOS/170 machine using the NOS/170 INTERFACE.

BLOCK DIAGRAM OF SIMULATED NOS/VE I/O USAGE OVERVIEW.

```
                +----------+  +----------+  +----------+  +----------+
                ! NOS/170  !  !  USER    !  !  SES.    !  !          !
         USER ->! EDITOR   !->! SOURCE   !->! GENCOMP  !<-! CYBICMN  !
                ! (etc.)   !  !  FILE    !  ! (etc.)   !  !          !
                +----------+  +----------+  +----------+  +----------+
                                                  !
                      +---------------------------+
                      V
                +----------+  +----------+  +----------+  +----------+
                ! COMPILE  !  !  CYBIL   !  !   LGO    !  !          !
                !  FILE    !->!  (CI)    !->!   FILE   !  ! CYBILIB  !
                !          !  !          !  !          !  !          !
                +----------+  +----------+  +----------+  +----------+
                                                  !              !
                                                  +------------+ !
                                                               V V
                +----------+  +----------+  +----------+  +----------+
                ! NOS/170  !  ! NOS/170  !  ! VIRTUAL  !  !   SES.   !
                !  FILE    !<-! EDITOR   !  !ENVIRONMT !<-!  GENCPF  !
                !          !  ! (etc.)   !  !   FILE   !  !          !
                +----------+  +----------+  +----------+  +----------+
                      !                           !
                      V                           V
                +----------+  +----------+  +----------+  +----------+
                !  SES.    !  !  C180    !  !  C180    !->! NOS/170  !
                !  IO180   !->!  INPUT   !->!SIMULATOR !  !INTERCHG. !
                !          !  !  FILE    !  !          !<-! FILE(S)  !
                +----------+  +----------+  +----------+  +----------+
                                              ! ! A ! A
                                    +---------+ ! ! ! +---------+
                                    V           V ! +---+       !
                +----------+  +----------+  +----------+! +----------+
                !  SES.    !  !  C180    !  !  C180    !! !  INPUT   !
                ! FROM180  !<-! OUTPUT   !  !INTERMED. !! !INTERCHG. !
                !          !  !  FILE    !  ! FILE(S)  !! !   FILE   !
                +----------+  +----------+  +----------+! +----------+
                      !                                 +-------+
                      V                                         V
                +----------+  +----------+  +----------+  +----------+
                ! NOS/170  !  !  SES.    !  ! PRINTER  !  !  OUTPUT  !
                !  FILE    !->!  PRINT   !->! LISTING  !<-!INTERCHG. !
                !          !  ! (etc.)   !  !          !  !   FILE   !
                +----------+  +----------+  +----------+  +----------+
```

2.0 SIMULATED NOS/VE I/O INTERFACES
2.1.1 USAGE OVERVIEW

```
BLOCK DIAGRAM OF NOS/170 INTERFACE USAGE                                      !

                 +----------+  +----------+  +----------+  +----------+       !
                 ! NOS/170  !  !   USER   !  !   SES.   !  !          !       !
          USER ->! EDITOR   !->!  SOURCE  !->! GENCOMP  !<-! CYBCCMN  !       !
                 ! (etc.)   !  !   FILE   !  ! (etc.)   !  !          !       !
                 +----------+  +----------+  +----------+  +----------+       !
                                                  !                           !
                      +---------------------------+                           !
                      V                                                       !
                 +----------+  +----------+  +----------+  +----------+       !
                 ! COMPILE  !  !  CYBIL   !  !   LGO    !  !          !       !
                 !  FILE    !->!  (CC)    !->!  FILE    !  ! CYBCLIB  !       !
                 !          !  ! (DEBUG)  !  !          !  !          !       !
                 +----------+  +----------+  +----------+  +----------+       !
                                                  !               !
                                                  +------------+  !           !
                                                               V  V           !
                 +----------+  +----------+  +----------+  +----------+       !
                 ! NOS/170  !  ! NOS/170  !  ! NOS/170  !  !   SES.   !       !
          USER-> ! EDITOR   !->!  FILE    !  !  LOAD    !<-! LINK170  !       !
                 ! (etc.)   !  !          !  !   FILE   !  ! (DEBUG)  !       !
                 +----------+  +----------+  +----------+  +----------+       !
                                    !             !
                      +-------------+             !                           !
                      V                           V                           !
                 +----------+  +----------+  +----------+  +----------+       !
                 !  SES.    !  !  INPUT   !  ! NOS/170  !  !  DEBUG   !       !
                 !  IO170   !->!INTERFACE !->!EXECUTION !<-! LIBRARY  !       !
                 !          !  !  FILE    !  ! (DEBUG)  !  !          !       !
                 +----------+  +----------+  +----------+  +----------+       !
                                               A ! A !                        !
                      +------------------------+ ! !  +----------+           !
                      V                          V !             V           !
                 +----------+                +----------+  +----------+       !
                 !  DEBUG   !                !INTERMED. !! ! OUTPUT   !       !
          USER ->!COMMANDS  !                !INTERFACE !! !INTERFACE !       !
                 !          !                ! FILE(S)  !! !  FILE    !       !
                 +----------+                +----------+! +----------+       !
                                                                 !
                      +------------------------------------------+            !
                      V                                                       !
                 +----------+  +----------+  +----------+  +----------+       !
                 !  SES.    !  ! NOS/170  !  !   SES.   !  ! PRINTER  !       !
                 ! FROM170  !->!  FILE    !->!  PRINT   !->! LISTING  !       !
                 !          !  !          !  !  (etc.)  !  !          !       !
                 +----------+  +----------+  +----------+  +----------+       !
```

2.1.2 FILE TYPES

The SIMULATED NOS/VE I/O system (AMP$) provides by default the record access level on unlabeled, sequential or byte addressable files with "W" type records on mass storage. This file structure is identified as having a block type of "amc$system_specified" and a record type of "amc$variable". SIMULATED NOS/VE I/O also provides the record access level for unlabeled, sequential files on mass storage only with the "interchange" file format, which provides for access of NOS/170 files directly by a NOS/VE program. The "interchange" file format is identified as having a block type which is "amc$user_specified" and a record type which is "amc$undefined".

Segment access and physical (read/write) access are not provided. No other record types, such as Ansi Fixed are permitted. Only mass storage files may be accessed by this product, with the exception of the interactive terminal files. User supplied exit procedures are not supported either.

The NOS/170 INTERFACE provides an equivalent interface for programs compiled with the CYBIL CC compiler, generating object code for a CYBER 170 machine. Only the "amc$variable" record_type is provided by this interface.

2.1.2.1 DEFAULT (CONTROL WORD) TYPE NOS/VE FILES

A control_word type file (also called W-type) has a record type ordinal identifier of "amc$variable". "Amc$variable" record type is distinguished by the fact that each record in the file is preceded by a record_header. For SIMULATED NOS/VE I/O the record_header is eight (8) bytes long. For the NOS/170 INTERFACE the record_header is one (1) word long.

This NOS/VE file structure is expected to be the most common NOS/VE file type, being used for batch job and interactive input and output, compilers, editors, and most other other called programs. Input and output to the interactive user terminal or batch job are also provided. A record is the nominal unit of data transfer in NOS/VE I/O, which is a different terminology than is used by NOS/170. For a more complete description of the NOS/VE I/O system refer to the NOS/VE ERS document.

The "amc$variable" file format of SIMULATED NOS/VE I/O does not record data on a NOS/170 file in a format which is directly accessable by any other NOS/170 products, such as editors, compilers, etc. Data conversion is provided only for character data via the conversion utilities (see Section 8 and 9).

2.1.2.1.1 SEQUENTIAL NOS/VE FILES

Sequential files have only sequential access and is identified as having a file_organization of "amc$sequential". Data appears in the file in the order in which it was written, and can only be read in that same order. These files may be positioned at the beginning or end of information. Note that positioning at the beginning and then writing a sequential file implies that all data which was previously present on the file is lost.

2.1.2.1.2 BYTE ADDRESSABLE NOS/VE FILES

Byte addressable files are like sequential files except the file may be positioned randomly by specifying a "byte address". Byte addressable files are identified as having a file organization of "amc$byte_addressable". Note that writing from the beginning of a byte addressable file does not necessarily imply that existing data (which follows the data being written) will be lost as is the case with the sequential files.

Byte addresses returned by the SIMULATED NOS/VE I/O system should be, but may not necessarily be the same as the byte addresses returned by NOS/VE.

2.1.2.2 NOS/170 OR "INTERCHANGE" FILES

The "interchange" file format is provided as a means of directly accessing any NOS/170 file. This file type is identified as having a record type of "amc$undefined".

"Interchange" files are defined to have no record structure. Data from an "interchange" type file is mapped into a NOS/VE program in terms of whole (NOS/170) words. Each 60-bit, NOS/170 word occupies the rightmost 60 bits of eight consecutive bytes (or 64 bits) of virtual memory. The remaining bits are zero filled. When an "interchange" type file is written, only the rightmost 60 bits of every 8 bytes is recorded on the file. For both gets and puts the users working storage length must be specified as an even multiple of 8 bytes, else an error is returned. Also, since only whole NOS/170 words are transmitted the "record_length", "transfer_count", and "byte_address" parameters returned by any of the "amp$get_" procedures will always be even multiples of 8.

SIMULATED NOS/VE I/O transmits data for an "interchange" type file only in terms of Physical Record Units (PRU's), which for mass storage files must be 64 NOS/170 words, or 512 bytes of virtual memory. For this file type SIMULATED NOS/VE I/O provides only for the physical transfer of whole (NOS/170) words of data.

SIMULATED NOS/VE I/O support of "INTERCHANGE" files includes reading of single record files and reading of multi_record files. Reading of "multi_file files" is not supported. SIMULATED NOS/VE I/O support of "INTERCHANGE" files also includes writing single record files and writing multi_record files. Generation of "multi_file files" is not supported.

For all "interchange" files the NOS/VE program must provide any necessary blocking or deblocking of the data.

This file type is not available with the NOS/170 INTERFACE. CYBIL I/O may be used.

2.1.2.2.1 SEQUENTIAL "INTERCHANGE" FILES

Each get_next or put_next transmits whole PRU's, or multiples of 512 (64 * 8) bytes of data. A get_next of less than 512 bytes of data will transmit the requested number of bytes of data and advances the file to the next PRU of data. A get_next of a short PRU with a NOS/170 "end_of_record" or a get_next of a short PRU with a NOS/170 "end_of_file" will cause an "amc$eop" (end_of_partition) status to be returned, in addition to any data returned and the file is positioned at the PRU following the short PRU. Get_partial requests may be used to retrieve lesser amounts of data in each PRU without skipping any data in the PRU. After a get_partial of a short PRU transmits the last word of data in the PRU it operates just like a get_next of a short PRU. A get_partial transmits data up to the end of the PRU but does not cross a PRU boundary.

A put_next writes a whole PRU of data and advances the file to the next PRU. A put_next of less than 512 bytes causes a short PRU with a NOS/170 "end_of_record" to be written. A put_partial with "term" parameter of "amc$start" or "amc$cont" may be used to add data to the last PRU of the file without causing a short PRU to be written and without causing the file to advance to the next PRU, unless the PRU is filled by the put_partial. A put_partial with "term" parameter of "amc$term", which does not fill a PRU with data, will cause a short PRU with a NOS/170 "end_of_record" to be written and the file will advance to the next PRU. After a put_partial is used to put some data in a PRU a put_next may be used to fill the PRU with data, but if the PRU is not filled by the put_next then a short PRU with a NOS/170 "end_of_record" is written and the file advances to the next PRU.

2.1.2.3 NOS/170 FILE ACCESS BY SIMULATED NOS/VE I/O

SIMULATED NOS/VE I/O utilizes the CYBIL I/O procedures for data transmissions with the NOS/170 files. For "amc$variable" type files the CYBER 180 program requests use 8-bit bytes as the unit of data transmission, whereas NOS/170 uses 60-bit words as the unit of data transmission. The object program interface for SIMULATED NOS/VE I/O records information for the amc$variable record_type on the NOS/170 file at seven 8-bit bytes per CYBER 170 CPU word. A record can start at any byte address on the file.

The intended use of the SIMULATED NOS/VE I/O system for byte addressable files is that the file initially be written sequentially and subsequently the file may be accessed randomly and records may be rewritten randomly or new records may be added at end of information. It is not recommended that the user program generate byte addresses for putting records into a file. Any usage of this product, other than that intended, may not necessarily be compatible with NOS/VE.

For "interchange" or amc$undefined record type files accessed by SIMULATED NOS/VE I/O the processing of any internal data structure of each PRU of data transmitted is the exclusive responsibility of the CYBER 180 program.

2.1.2.4 NOS/170 INTERFACE FILES

Only the control_word type of files are provided by this interface. Interactive input/output is also available. The NOS/170 INTERFACE does not record data on a file which is directly accessable by any other NOS/170 products. Conversion is provided only for character data. See Section 9 for details about conversion.

2.2 I/O PROCEDURES

Only the following I/O procedures are supported by the SIMULATED NOS/VE I/O system. The procedures are listed alphabetically.

The user is cautioned that undefined input parameters of the procedures give undefined results. Other procedures in the FILE MANAGEMENT section of the NOS/VE ERS which are not mentioned in this ERS are not supported by the SIMULATED NOS/VE I/O system.

2.2.1 AMP$CLOSE

This procedure closes the file and invalidates the file identifier of the file. This procedure is fully supported by the SIMULATED NOS/VE I/O system.

2.2.2 AMP$FETCH

AMP$FETCH returns the identical file_attribute information as the AMP$GET_FILE_ATTRIBUTES procedure. AMP$FETCH requires the file_identifier of an open file as input, whereas AMP$GET_FILE_ATTRIBUTES uses local_file_name. The caller must initialize all file_attribute keys and AMP$FETCH returns the corresponding file_attribute value. Refer to the section on data types for a complete description of file_attributes returned by the SIMULATED NOS/VE I/O system.

2.2.3 AMP$FETCH_ACCESS_INFORMATION

This procedure returns information about the current status of the file to the caller. The information is returned in the access_information record. The caller must initialize all access_information keys and the SIMULATED NOS/VE I/O system returns the corresponding access_information value. Refer to the section on data_types for a complete explanation of the information returned by the SIMULATED NOS/VE I/O system.

2.2.4 AMP$FILE

This procedure defines the attributes of a file. Although NOS/VE I/O permits this procedure to be executed at any time, the SIMULATED NOS/VE I/O system does not permit this procedure to be executed when the file is open. The AMP$STORE procedure may be used instead.

The caller must initialize the file_attributes parameter, both key and value, before calling AMP$FILE. All file_attributes are accepted by the AMP$FILE procedure as input, but only some file_attributes are used by the SIMULATED NOS/VE I/O system. Refer to the section on data types for a description of the file_attributes parameter. The user must also supply local_file_name as input.

2.2.5 AMP$GET_FILE_ATTRIBUTES

This procedure is normally executed before opening a file to determine the previous definition, if any, of the file. The information is returned in the file_attributes record. The caller must initialize all file_attribute keys and the SIMULATED NOS/VE I/O system will return the corresponding file_attribute value. The user must supply the local_file_name parameter as input. Status of the request is returned in "status".

AMP$GET_FILE_ATTRIBUTES will return default values for all file_attributes if the file is unknown to SIMULATED NOS/VE I/O system. Refer to the subsequent section on data types for a description of the file_attributes parameter.

2.2.6 AMP$GET_DIRECT

This procedure returns data from the location on the file specified by the caller. This procedure is fully supported by the SIMULATED NOS/VE I/O system for files with amc$variable record_type. This procedure is supported with "interchange" type files having a record type of amc$undefined but the byte_address parameter must specify the first byte of a PRU (be a multiple of 512). NOS/VE will not provide this procedure for "interchange" files.

2.2.7 AMP$GET_NEXT

This procedure returns the next sequential record of data from the file to the user program. This procedure is fully supported by the SIMULATED NOS/VE I/O system.

2.2.8 AMP$GET_PARTIAL

This procedure returns a partial record from the next location of the file. Subsequent AMP$GET_PARTIAL requests are used to retrieve the remainder of the total record. This procedure is fully supported by the SIMULATED NOS/VE I/O system.

2.2.9 AMP$OPEN

This procedure opens a file whose local_file_name is input and returns the file_identifier to the caller. The only access_level parameter permitted is "amc$record". This procedure is fully supported by the SIMULATED NOS/VE I/O system, with the exceptions that multiple opens of the file at the same time are not permitted and the local_file_name must conform to the limits of construction imposed by the SIMULATED NOS/VE I/O system. Refer to the section on data types for an explanation of the limits to the construction of local_file_name.

2.2.10 AMP$PUT_DIRECT

This procedure puts a whole record into the file at the byte_address specified by the user. This procedure is fully supported by the SIMULATED NOS/VE I/O system for files with record type of amc$variable. The user may rewrite an existing record on the file, or the user may write a new record at or beyond the end of information currently defined for the file. This means that the byte address specified must point to an existing record on the file (so that amp$put_direct re-writes the existing record) or byte address may point beyond the last record of the file (so amp$put_direct adds a new record to the file).

This procedure is supported for "interchange" type files having a record_type of amc$undefined, but the byte_address parameter must specify the first byte of a PRU. A put_direct may rewrite an existing PRU of the file or it may write a new PRU at end_of_information only. Put_direct can not be used to write beyond end_of_information. NOS/VE will not provide this procedure for "interchange" files.

2.2.11 AMP$PUT_NEXT

This procedure transfers a whole record from the users working storage area to the "next" location in a file. This procedure is fully supported by the SIMULATED NOS/VE I/O system.

2.2.12 AMP$PUT_PARTIAL

This request transfers a partial record from the users working storage area to the "next" location in the file. NOS/VE permits the user to put one or more parts of a record into the file followed by a put partial specifying the start of a new record, but the SIMULATED NOS/VE I/O system does not support this feature. Once the first part of a record has been put into the file the user must do a put partial of the last part of the record before a new record is started, else an error is returned. All other features of put_partial are supported by the SIMULATED NOS/VE I/O system.

2.2.13 AMP$REWIND

This procedure positions a file at beginning of information. This procedure is supported by the SIMULATED NOS/VE I/O system, but the WAIT parameter is not interrogated as OSC$WAIT is assumed.

2.2.14 AMP$SEEK_DIRECT

AMP$SEEK_DIRECT is used to set the current_byte_address of a file to a new value. This procedure may be used only on files opened with a file_organization of amc$byte_addressable.
The caller must insure that the byte_address parameter input points to the start of a record. Amp$seek_direct does verify that the new current_byte_address does point to the start of a record. A subsequent ampget_next, ampget_partial, amp$put_next, or amp$put_partial call that uses current byte address would return an error if an invalid byte_address parameter were input to amp$seek_direct.

2.2.15 AMP$STORE

AMP$STORE procedure is used to set file attributes of an open file identified by the file_identifier parameter. File_attributes to be changed are specified by the store_attributes parameter. The store_attributes parameter is a subset of the file_attributes parameter used by the procedures AMP$FETCH, AMP$FILE, and AMP$GET_FILE_ATTRIBUTES. All store_attributes are permitted by the SIMULATED NOS/VE I/O system. Refer to the section on data types for a complete description of which file_attributes are legal store_attributes and which file_attributes are interpreted by the SIMULATED NOS/VE I/O system.

2.3 DATA TYPES

This section defines the CYBIL language "types" required to interface the SIMULATED NOS/VE I/O system.

The data types necessary for SIMULATED NOS/VE I/O are identical to the data types necessary for the NOS/170 INTERFACE and are a subset of the total capability provided by NOS/VE, yet the user will employ common decks which provide a complete definition of the NOS/VE I/O system. This means that there will be some data types which are not used and some defined values of parameters not allowed by the SIMULATED NOS/VE I/O system. This section describes only those data types used and describes which values of parameters are allowed in the SIMULATED NOS/VE I/O system.

A complete listing of each of the common decks employed by the user is given in Section 5. These common decks are input to the CYBIL compiler, but they contain comments which describe the meaning and usage of the defined data types and procedure parameters. The NOS/VE definitions described in the common decks will always apply to SIMULATED NOS/VE I/O and the NOS/170 INTERFACE, unless this document states otherwise.

The NOS/VE common decks are subject to change. The user is cautioned that the definitions in the common decks will always apply and that as necessary, SIMULATED NOS/VE I/O and the NOS/170 INTERFACE will be updated to reflect such changes in NOS/VE. Problems with usage might arise where an out of date ERS reference is used.

2.3.1 ACCESS_INFORMATION

ACCESS_INFORMATION is an adaptable array of case variant records. Each case variant record holds one piece of access information. The adaptable array allows the caller to obtain as many pieces of access information as needed per call. The caller must initialize all access_information keys and the SIMULATED NOS/VE I/O system returns the corresponding access information value. The ACCESS_INFORMATION parameter is described in the common deck AMDFNFO.
The SIMULATED NOS/VE I/O system accepts all the different access_information cases. The following table shows what pieces of access information are maintained by the SIMULATED NOS/VE I/O system and what pieces of access information have default values returned. The letter 'S' in the value returned column indicates that the current value of the access information kept by the SIMULATED NOS/VE I/O system is returned. The letter 'D' in the value returned column

indicates that a default value is returned and the value is given in the default value column. The COMMON DECK column tells which common deck contains the definition of the corresponding access value "type".

TABLE of ACCESS_INFORMATION

ACCESS_ INFORMATION_	COMMON_ DECK_	VALUE_ RETURNED_	DEFAULT VALUE
access_level	AMDOPEN	D	amc$record
allocation_unit_length	AMDINFO	D	2048
block_number	AMDINFO	D	1
current_byte_address	AMDGLOB	S	
cycle_number	PFDATRB	D	1
device_class	RMDCLAS	S	
eoi_byte_address	AMDGLOB	S	
error_status	OSDSTAT	D	0
file_position	AMDGLOB	S	
global_file_name	AMDINFO	D	0
global_share_mode	PFDATRB	D	null set
last_operation	AMDINFO	S	
last_op_status	AMDINFO	D	amc$complete
local_file_name	AMDNAME	S	
record_header_length	AMDRCDH	D	8
record_length	AMDGLOB	S	
residual_skip_count	AMDINFO	D	1
storage_class	RMDCLAS	D	rmc$temporary_device
transfer_count	AMDGLOB	S	
volume_number	AMDINFO	D	1

2.3.2 ACCESS_LEVEL

This parameter indicates what type of access is desired when the file is opened. This parameter must be set to the ordinal AMC$RECORD when the file is opened else an error is detected. The user must always set this parameter prior to calling AMP$OPEN. Other ordinal values are not supported.

2.3.3 BYTE_ADDRESS

Byte_address is a parameter returned by AMPGET_NEXT, AMPPUT_NEXT, AMP$GET_PARTIAL, and AMP$PUT_PARTIAL. Byte_address is an input parameter to AMP$GET_DIRECT and AMP$PUT_DIRECT.

For a NOS/VE format file byte_address is the number of 8-bit bytes from the beginning of information to the location of the current record in the file.

For a NOS/170 format or "INTERCHANGE" file byte_address is defined as the number of 8-bit bytes from the beginning of information to the location of the first byte in the current word in the file. Because only whole NOS/170 words are transmitted, byte_address for this file type will always be an even multiple of 8.

2.3.4 FILE_ATTRIBUTES

FILE_ATTRIBUTES is an adaptable array of a case variant record. Each case variant record holds one file_attribute. The adaptable array allows the user to process as many attributes as needed per procedure call. File_attributes are returned to the caller by the procedures AMP$GET_FILE_ATTRIBUTES and AMP$FETCH. File_attributes may be specified by the caller of the procedures AMP$FILE and AMP$STORE. Note that for the procedure AMP$STORE the parameter is type AMT$STORE_ATTRIBUTES, but upon closer inspection one discovers that this type is actually a subset of the AMT$FILE_ATTRIBUTES type. The definition of the file_attribute parameter is contained in the common deck AMDFATT and the definition of the store_attribute parameter is contained in the deck AMDSTOR.

The SIMULATED NOS/VE I/O system accepts and returns all the different cases of AMT$FILE_ATTRIBUTES. The following table shows what attributes are legal AMT$STORE_ATTRIBUTES, whether the attribute information is retained by the SIMULATED NOS/VE I/O system, and what value is returned by the SIMULATED NOS/VE I/O system. The table also shows the default value returned by AMP$GET_FILE_ATTRIBUTES when the file is unknown. The letter 'S' in the ACTION TAKEN column indicates the system saves the information in its file tables and returns the same saved information to the caller and the default value is returned only when the file is unknown. The letter 'D' in the ACTION TAKEN column indicates the SIMULATED NOS/VE I/O system defaults, the attribute value is not saved, and the system returns a default value for all files. The COMMON DECK column gives the name of the common deck which defines the corresponding file_attribute "type".

TABLE of FILE_ATTRIBUTES

FILE_ATTRIBUTE_ NAME_	STORE_ ATTRIBUTE_	COMMON_ DECK_	ACTION_ TAKEN_	DEFAULT VALUE
access_mode	no	PFDATRB	S	null set
block_header_length	no	AMDBLKH	D	16

block_type	no	AMDFILE	S	amc$system_specified
character_conversion	yes	AMDFATT	D	FALSE
clear_space	no	OSDCFS	D	FALSE
error_exit_procedure	yes	AMDFILE	D	NIL
error_options	yes	AMDFILE	D	amc$terminate_file
file_command_processing	no	AMDFATT	D	FALSE
file_content	yes	AMDFILE	D	amc$unknown_class, amc$unknown_kind, amc$unknown_nature
file_disposition	no	AMDFILE	D	amc$task_local
file_limit	yes	AMDFILE	S	0
file_organization	no	AMDFILE	S	amc$sequential
horizontal_print_density	yes	AMDFILE	D	amc$ten_cpi
internal_code	no	AMDFILE	D	amc$ascii
job_local_share_mode	no	PFDATRB	D	null set
label_exit_procedure	yes	AMDFILE	D	NIL
label_options	yes	AMDFILE	D	null set
label_type	no	AMDFILE	D	amc$unlabelled
line_number	yes	AMDFILE	D	1,1
max_block_length	no	AMDGLOB	D	1
max_record_length	no	AMDGLOB	S	0
min_block_length	no	AMDFILE	D	1
min_record_length	no	AMDFILE	S	0
open_position	no	AMDOPOS	S	amc$open_at_boi
owncode_procedure	no	PMDNAME	D	NIL
padding_character	yes	AMDFILE	D	blank
page_format	yes	AMDFILE	D	amc$continuous_form
page_length	yes	AMDFILE	D	66
page_width	yes	AMDFILE	D	72
preset_value	no	AMDFILE	D	0
record_type	no	AMDFILE	S	amc$variable
ring_attributes	no	AMDFILE	D	15,15,15
suppress_buffering	yes	AMDFATT	D	FALSE
transfer_unit	yes	AMDFILE	D	amc$t
user_info	yes	AMDFILE	D	blanks
vertical_print_density	yes	AMDFILE	D	amc$six_lpi

The subsequent sections describe those file_attributes which directly affect the use of the SIMULATED NOS/VE I/O system.

2.3.4.1 ACCESS_MODE

Access_mode is a set of 5 attributes. Any combination of access_mode set members is permitted. All set members except PFC$EXECUTE are used by the SIMULATED NOS/VE I/O system.

PFC$READ access_mode by itself means the file can only be read. When pfc$read is the only access_mode set element the open_position of the file must not be open at end of information. When used with other access_mode set elements it means the file can also be read.

PFC$SHORTEN access_mode by itself means the file can be rewritten from beginning of information to end of information only. The file size cannot increase. The file may not be read unless the access_mode set element pfc$read is also present. Each write establishes a new end of information and all data previously in the file beyond end of information is lost. When used with other access_mode set elements it means that when the file is closed a new end of information is established for the file at the highest address written.

PFC$APPEND access_mode by itself permits the file to be written at end of information only and the open_position file_attribute must correspondingly be set to open at end of information. The file may not be read unless the access_mode set includes pfc$read. When used with other access_mode set elements it allows the file size to increase.

PFC$MODIFY access_mode by itself requires an existing file and permits any portion of the file to be written without affecting any subsequent data in the file. The end of information remains unchanged when the file is closed. When used with other access_mode set elements it causes all previous contents of the file to be retained except the area directly written.

The default value for access_mode is the NULL set. When a file is opened with a NULL set for access_mode the access_mode is changed. If the file already exists at the time of the open then access_mode defaults to the set of [PFC$READ]. If a new file is being created the access_mode defaults to the set of [PFC$READ, PFC$SHORTEN, PFC$APPEND, PFC$MODIFY].

For input interactive files, such as 'INPUT', the SIMULATED NOS/VE I/O system requires the access_mode to be the set of [PFC$READ], which defaults for an existing file. An output interactive file, such as 'OUTPUT', must have an access_mode which consists of the set [PFC$APPEND], which does not default for an existing file. The open_position of the file must be amc$open_at_eoi too. The restrictions to the access_mode of interactive files are inherent in the SIMULATED NOS/VE I/O system and are not limitations of NOS/VE.

2.3.4.2 BLOCK_TYPE

This parameter is an ordinal which indicates blocking structure of the file.

AMC$SYSTEM_SPECIFIED indicates the default (control_word) NOS/VE block_type. This is the default file_attribute returned when the file is unknown. !

AMC$USER_SPECIFIED must be used with a record_type or "amc$undefined" for an "interchange" file definition, which accesses a NOS/170 file directly. This block_type is not provided by the NOS/170 INTERFACE. !

2.3.4.3 FILE_ORGANIZATION

This attribute specifies whether the file is sequential, or byte_addressable.

AMC$SEQUENTIAL indicates a sequential file (default value).

AMC$BYTE_ADDRESSABLE indicates a byte_addressable file. This file organization is required if get_direct and put_direct procedures are to be used.

2.3.4.4 OPEN_POSITION

This parameter is an ordinal which indicates what positioning should be performed for the file being opened.

AMC$OPEN_AT_BOI indicates file position is beginning of information. This is the required open position if the access_mode consists only of the set element pfc$read. This ordinal defaults when access_mode is a null set. !

AMC$OPEN_AT_EOI indicates file position is end of information. This is the required open_position when the access_mode is the set of pfc$append only. !

AMC$OPEN_NO_POSITIONING indicates file position left unchanged. !

Other ordinal values are not permitted.

2.3.4.5 RECORD_TYPE

This parameter is an ordinal which indicates the record structure of the data on the file.

AMC$VARIABLE indicates the default (control_word) type NOS/VE file. This ordinal must be used in conjunction with a block_type of "amc$system_default" to specify the control_word type NOS/VE file. This is the default file_attribute value returned when the file is unknown.

AMC$UNDEFINED indicates an "interchange" file. This ordinal must be used in conjunction with a block_type of "amc$user_specified" to provide the "interchange" file definition for access of a NOS/170 file directly by the program. This record_type is not provided by the NOS/170 INTERFACE.

Other ordinal values of record_type are not supported.

2.3.5 FILE_IDENTIFIER

This data type is used when calling any of the AMP$ procedures except AMP$GET_FILE_ATTRIBUTES and AMP$FILE. A file identifier variable must be provided for each file opened. The value of file_identifier for a file is returned by AMP$OPEN for the file being opened. File_identifier is used by all other procedures except AMP$GET_FILE_ATTRIBUTES, AMP$FILE, and AMP$OPEN to identify the file being accessed. Multiple opens of a file in the same program are not supported by the SIMULATED NOS/VE I/O system. The file_identifier of each file open concurrently will be unique. File_identifier remains defined until the file is closed. Under NOS/VE a user program may not define more than 2000 files concurrently. This data type must be left undefined by the user.

2.3.6 FILE_POSITION

This parameter is an ordinal that is returned by AMPGET_DIRECT, AMPGET_NEXT, and AMP$GET_PARTIAL procedures. The parameter may also be returned by AMP$FETCH_ACCESS_INFORMATION. The SIMULATED NOS/VE I/O system currently returns only AMCBOI, AMCMID_RECORD, AMC$EOR, or AMC$EOI with the amc$variable record type. Other ordinal values are not returned. See section 3.3 for further information.

For "interchange" file format the AMC$EOP status can be returned in addition to all the previously mentioned ordinals. A status of AMC$EOP indicates "end_of_record" , or a short PRU of data on the file. When the file position is at the start of a PRU the status returned is AMC$EOR and when the file position is in the middle of a PRU the status returned is AMC$MID_RECORD.

2.3.7 LOCAL_FILE_NAME

LOCAL_FILE_NAME is an input parameter for the procedures AMP$FILE, AMP$OPEN, and AMP$GET_FILE_ATTRIBUTES and it is an output parameter of the procedure AMP$FETCH_ACCESS_INFORMATION. When local_file_name is input it is an adaptable string type, but when it is output it is a fixed string of 31 characters.

NOS/VE requires that the local_file_name conform to the SCL rules for constructing a name. SCL permits a name to be a string of alphabetic characters or decimal digits of 31 characters maximum length and the first character must not be a decimal digit. SCL defines an alphabetic character to be a letter or the special characters # (number_sign), $ (dollar_sign), @ (at_symbol), or _ (underline). NOS/170 only permits letters and decimal digits for file names and the maximum length is 7 characters. This means that the SIMULATED NOS/VE I/O system can only utilize the first 7 characters of local file name. Though NOS/VE permits multiple opens of the same file concurrently in the same program, this feature is not supported by the SIMULATED NOS/VE I/O system. The file may be opened only once before it is closed, but the file may be opened again following the close.

The user must insure that local file names are unique within the first 7 characters and that local_file_name contains only letters and decimal digit characters, else an error will be detected by the SIMULATED NOS/VE I/O system.

There is no default value set for local file name.

2.3.8 RECORD_LENGTH

Record_length is a parameter returned on all AMPGET_DIRECT, AMPGET_NEXT, and AMP$GET_PARTIAL procedure calls.

For files with record_type of AMC$VARIABLE the record_length parameter is the number of 8-bit bytes in the record.

For "INTERCHANGE" files, with record_type AMC$UNDEFINED, record_length parameter is the number of 8-bit bytes into which the data from the start of the current PRU to the last word transmitted are mapped, that is, it is 8 times the number of words from the start of the current PRU to the last word transmitted.

2.3.9 SKIP

The skip parameter is input for AMP$GET_PARTIAL procedure. All ordinal values are permitted.

2.3.10 STATUS

This data type is used when calling any of the AMP$ procedures. Status is a record of data returned by the procedure which indicates whether the operation performed was successful or whether an error was encountered. STATUS.NORMAL should be set TRUE by the user before calling any AMP$ procedure as the procedures do not change STATUS. No other fields of this record will be referenced by the SIMULATED NOS/VE I/O. The NOS/170 INTERFACE will use all fields of this parameter during the course of generating error messages.
Since the SIMULATED NOS/VE I/O system does not provide the user program with error processing capability, a FALSE value of STATUS.NORMAL can not occur. Should SIMULATED NOS/VE I/O detect an error, the simulation run is halted and a diagnostic message is sent to the OUTPUT file (usually the interactive NOS/170 terminal). When the NOS/170 INTERFACE detects an error a diagnostic message is generated and sent to the OUTPUT file and the program aborts. The user may then interrogate the error and restart execution after the error is corrected.

<u>The user is encouraged to incorporate code for status checking into programs using the SIMULATED NOS/VE I/O system in anticipation of running the programs on the CYBER 180.</u>

2.3.11 TERM

Term is an input parameter of AMP$PUT_PARTIAL. All ordinal values are permitted.

2.3.12 TRANSFER_COUNT

Transfer count is a parameter returned by AMPGET_DIRECT, AMPGET_NEXT, and AMP$GET_PARTIAL procedure calls. Transfer_count may also be returned by AMP$FETCH_ACCESS_INFORMATION.

For files with record_type of AMC$VARIABLE transfer_count is the number of 8-bit bytes transferred with this procedure call.

For "INTERCHANGE" files, with record_type AMC$UNDEFINED, transfer_count is defined as the number of 8-bit bytes moved by the current call.

2.3.13 WAIT

The wait parameter is an ordinal which is input for an AMP$REWIND procedure call. The parameter is not interrogated by the SIMULATED NOS/VE I/O system as OSC$WAIT is always assumed. !

2.3.14 WORKING_STORAGE_AREA

Working_storage_area is a pointer to the users data buffer. It is recommended that the pointer be generated through the use of the "#LOC" function.

2.3.15 WORKING_STORAGE_LENGTH

Working_storage_length specifies the users buffer size in bytes for AMPGET_DIRECT, AMPGET_NEXT, and AMP$GET_PARTIAL procedure ! calls. It specifies the number of bytes to transfer for ! AMPPUT_DIRECT, AMPPUT_NEXT, and AMP$PUT_PARTIAL procedure calls. ! It is recommended that the size be generated through the use of the "#SIZE" function for all AMP$GET_ type procedure calls.

For the "interchange" file format this parameter must be a multiple of 8, else an error will be returned.

2.4 USING THE SIMULATED NOS/VE I/O SYSTEM

2.4.1 SOURCE CODE FOR THE SIMULATED NOS/VE I/O SYSTEM

2.4.1.1 CONSIDERATIONS

SIMULATED NOS/VE I/O must coexist with the NOS/VE operating system when both are running on the SIMULATOR. Since NOS/VE does not permit duplicate procedure names that are declared external via XDCL in CYBIL or ALIAS in the cpu assembly language, the SIMULATED NOS/VE I/O names have the "$" (dollar sign) changed to "#" (pound sign).

For example: When using the NOS/170 INTERFACE to call the procedure AMP$OPEN, use the name AMP$OPEN; but when using the CYBER 180 SIMULATOR, use the name AMP#OPEN. If the name AMP$OPEN is used on the SIMULATOR then the NOS/VE procedure is called and not SIMULATED NOS/VE I/O.

2.4.1.2 INCLUDING SOURCE CODE IN A MODULE

To interface the SIMULATED NOS/VE I/O system, the user's CYBIL program must include the relevant procedure call declarations. Each procedure is contained in a separate common deck of code which the user may include in the source program module through the use of *CALL statements. Each procedure contains the necessary nested call (*CALLC) statements to include any required "type" or "constant" definitions. Common deck names are the same regardless of whether the NOS/170 INTERFACE or the SIMULATOR are used. The following table gives the name of each procedure common deck and identifies the defined procedure.

TABLE of the SIMULATED NOS/VE I/O systen COMMON DECKS

COMMON DECK	PROCEDURE NAME
AMXCLSE	AMP$CLOSE
AMXFTCH	AMP$FETCH
AMXFNFO	AMP$FETCH_ACCESS_INFORMATION
AMXFILE	AMP$FILE
AMXGFAT	AMP$GET_FILE_ATTRIBUTES
AMXGETD	AMP$GET_DIRECT
AMXGETN	AMP$GET_NEXT
AMXGETP	AMP$GET_PARTIAL
AMXOPEN	AMP$OPEN
AMXPUTD	AMP$PUT_DIRECT
AMXPUTN	AMP$PUT_NEXT
AMXPUTP	AMP$PUT_PARTIAL
AMXREWD	AMP$REWIND
AMXSEEK	AMP$SEEK_DIRECT
AMXSTOR	AMP$STORE

2.4.1.3 SIMULATED NOS/VE I/O COMMON DECKS

SIMULATED NOS/VE I/O common decks are found on the "CYBICMN" source program library. SES.GENCOMP will find the required common decks when the "CYBICMN" keyword is part of the SES procedure call. Remember that these common decks have AMP# procedure names.

2.4.1.4 NOS/170 INTERFACE COMMON DECKS

The NOS/170 INTERFACE requires a different set of common decks than the SIMULATED NOS/VE I/O interface. These common decks have the same name as their SIMULATED NOS/VE I/O counterparts but are found on the "CYBCCMN" library. These common decks have the AMP$ procedure names. They may be obtained by using the "CYBCCMN" keyword on the SES.GENCOMP statement.

2.4.2 OBJECT PROGRAM LINKAGE TO AMP$

2.4.2.1 SIMULATED NOS/VE I/O OBJECT PROGRAM LIBRARY

The NOS/VE object program is linked and prepared for loading into the CYBER 180 SIMULATOR by the Virtual Environment Linker and the CheckPoint File GENerator. These programs are normally executed by the command SES.GENCPF. This is not the only method to prepare programs for input to the SIMULATOR. The user is directed to the ERS documents for the Virtual Environment Linker (VELINK) and the

Virtual Environment Generator (VEGEN) for additional information.
The object time programs for SIMULATED NOS/VE I/O are found on the "CYBILIB" library. GENCPF has the keyword parameter "CYBILIB" to provide access to this library.

2.4.2.2 NOS/170 INTERFACE OBJECT PROGRAM LIBRARY

The NOS/170 INTERFACE object modules may be obtained from the "CYBCLIB" library. The user should prepare the object program for execution be creating a NOS/170 "load file". This process is performed by the SES procedure SES.LINK170. LINK170 has the keyword "CYBCLIB" available to link CYBIL object time programs. The object programs for the NOS/170 INTERFACE are part of this library.

2.5 NOS/VE I/O COMMON DECK LISTINGS

This section lists the contents of each common deck of source code for the NOS/VE system which is supported by the SIMULATED NOS/VE I/O system.

2.5.1 PROCEDURE COMMON DECKS

These common decks must be included by the user in order to call the simulated procedure.

2.5.1.1 AMXCLSE (AMP$CLOSE)

```
{
{   The purpose of this request is to terminate file access. If the}
{ file_disposition is amc$task_local, the file is returned to the system.}
{ If the file_disposition is amc$job_global, any modified data in memory}
{ is flushed to the assigned device.  Any outstanding no_wait physical}
{ accesses are allowed to complete before control returns to the caller}
{ of this request.}
{   This request is optional. Task termination performs a close on all files}
{ opened by the task.}
{   If the preceding operation on the file was an output and the label_type}
{ is amc$labelled, a standard ANSI EOF label group is written. If the}
{ label_type is amc$unlabelled and the file is assigned to a tape device,}
{ two tapemarks are written.}
{   If a label_exit_procedure has been specified, control will be passed}
{ to this procedure whenever a standard ANSI label matching the label_options}
{ attribute is processed.}
{   If the file has been routed with the defer option, the file will be routed}
{ by amp$close if the file_disposition is amc$task_local; otherwise, the}
{ file is routed at the RETURN command/request or job termination.}
{   The file_identifier established by a preceding amp$open request is}
{ invalidated by this request.
{
{       AMP$CLOSE (FILE_IDENTIFIER, STATUS)
{
{ FILE_IDENTIFIER: (input) This parameter specifies the file access
{       identifier established when the file was opened.
{
{
{ STATUS: (output) This parameter specifies the request status
{}


{ * amxclse}
  PROCEDURE [XREF] amp#close ALIAS 'amxclse' (file_identifier:
    amt$file_identifier;
    VAR status: ost$status);

*callc amdglob
*callc osdstat
```

2.5.1.2 AMXFILE (AMP$FILE)

```
{
{   The purpose of this request is to describe a file prior to access.
{ The description of the file consists of one or more file attributes.
{ Each file attribute has an appropriate default value depending upon
{ the type of file to be accessed. File attributes are preserved with
{ the file. Therefore, file attributes need only be specified when the
{ file is created, and then only if the default is not desired.
{   If the user has control permission on an existing file, this request
{ may be used to permanently change the file's attributes. The change
{ will not take place until the file is successfully opened.
{   If the user does not have control permission on an existing file,
{ this request may be used to override the file's attributes until the
{ task terminates. These changes are not preserved with the file.
{   Repeated amp$file requests may be made against the same local file.
{ The requests are additive. When two requests redefine the same
{ attribute, the most recent definition is used.
{   File attributes defined on a preceding FILE command will override
{ those specified on this request. However, the task can prevent
{ FILE command processing by quoting the file_command_processing
{ attribute on this request.
{
{       AMP$FILE (LOCAL_FILE_NAME, FILE_ATTRIBUTES, STATUS)
{
{ LOCAL_FILE_NAME: (input) This parameter specifies the name of the local
{       file for which attribute values are being supplied.
{
{ FILE_ATTRIBUTES: (input) This parameter specifies one or more file
{       attribute-value pairs.
{
{ STATUS: (output) This parameter specifies the request status.
{


{ * amxfile}
  PROCEDURE [XREF] amp#file ALIAS 'amxfile' (local_file_name:
    amt$file_name;
    file_attributes: amt$file_attributes;
    VAR status: ost$status);
```

```
*callc amdfatt
*callc amdglob
*callc amdname
*callc osdstat
```

2.5.1.3 AMXFNFO (AMP$FETCH_ACCESS_INFORMATION)

```
{
{   The purpose of this request is to retrieve the value of one or
{ more items of file information subsequent to the file being opened.
{
{       AMP$FETCH_ACCESS_INFORMATION (FILE_IDENTIFIER, ACCESS_INFORMATION,
{         STATUS)
{
{ FILE_IDENTIFIER: (input) This parameter specifies the file access
{       identifier established when the file was opened.
{
{ ACCESS_INFORMATION: (input-output) This parameter specifies one or more
{       items of file access information to be returned.
{
{ STATUS: (output) This parameter specifies the request status.
{


{ * amxfnfo}
  PROCEDURE [XREF] amp#fetch_access_information ALIAS 'amxfnfo'
    (file_identifier: amt$file_identifier;
    VAR access_information: amt$access_information;
    VAR status: ost$status);

*callc amdfnfo
*callc amdglob
*callc osdstat
```

2.5.1.4 AMXFTCH (AMP$FETCH)

```
{
{   The purpose of this request is to retrieve the value of one or
{ more file attributes subsequent to the file being opened.  This
{ request is similar to amp$get_file_attributes except the file_identifier
{ is used to distinguish from among what may be several instances
{ of open of the same file.
{
{       AMP$FETCH (FILE_IDENTIFIER, FILE_ATTRIBUTES, STATUS)
{
{ FILE_IDENTIFIER: (input) This parameter specifies the file access
{       identifier established when the file was opened.
{
{ FILE_ATTRIBUTES: (input-output) This parameter specifies one or more
{       attributes whose value is sought.
{
{ STATUS: (output) This parameter specifies the request status.
{


{ * amxftch}
  PROCEDURE [XREF] amp#fetch ALIAS 'amxftch' (file_identifier:
    amt$file_identifier;
    VAR file_attributes: amt$file_attributes;
    VAR status: ost$status);
*callc amdfatt
*callc amdglob
*callc osdstat
```

2.5.1.5 AMXGETD (AMP$GET_DIRECT)

```
{
{   The purpose of this request is to retrieve a record from a file opened}
{ with amc$record access and file_organization of amc$byte_addressable.}
{   The location of the record is specified by the byte_address parameter.}
{   The record is moved from buffers maintained by the access method to the}
{ user's working_storage_area. Data movement always begins at a record }
{ boundary and continues until either the end of the record or the end of}
{ the working_storage_area is encountered, whichever occurs first. If the}
{ entire record is not moved, an abnormal STATUS will be returned. In this}
{ event, subsequent amp$get_partial requests may be issued to obtain the}
{ remainder of the record.}
{
{       AMP$GET_DIRECT (FILE_IDENTIFIER, WORKING_STORAGE_AREA,
{         WORKING_STORAGE_LENGTH, RECORD_LENGTH, TRANSFER_COUNT,
{         BYTE_ADDRESS, FILE_POSITION, STATUS)
{
{ FILE_IDENTIFIER: (input) This parameter specifies the file access
{       identifier established when the file was opened.
{
{ WORKING_STORAGE_AREA: (input) This parameter specifies the user's working
{       storage area.
{
{ WORKING_STORAGE_LENGTH: (input) This parameter specifies the maximum
{       number of bytes to be moved into the working storage area.  The
{       value is typically set at the maximum record size defined for the
{       file.
{
```

```
{ RECORD_LENGTH: (output) This parameter specifies the actual size of the
{       record as it exists on the file.
{
{ TRANSFER_COUNT: (output) This parameter specifies the number of bytes
{       moved to the user's buffer.
{
{ BYTE_ADDRESS: (input) This parameter specifies the file byte address
{       of the record to be retrieved.
{
{ FILE_POSITION: (output) This parameter specifies the position of the
{       file following this request.  It can be used to recognize end
{       of record status.
{
{ STATUS: (output) This parameter specifies the request status.
{


{ * amxgetd}
  PROCEDURE [XREF] amp#get_direct ALIAS 'amxgetd' (file_identifier:
    amt$file_identifier;
    working_storage_area: ^cell;
    working_storage_length: amt$working_storage_length;
    VAR record_length: amt$max_record_length;
    VAR transfer_count: amt$transfer_count;
    byte_address: amt$file_byte_address;
    VAR file_position: amt$file_position;
    VAR status: ost$status);

*callc amdglob
*callc osdstat
```

2.5.1.6 <u>AMXGETN (AMP$GET_NEXT)</u>

```
{
{   The purpose of this request is to retrieve the next record from a}
{ file opened with access_level of amc$record. The location of the }
{ record is the current byte address of the file.}
{   This request may be issued regardless of the file_organization.}
{   The record is moved from buffers maintained by the access method to the}
{ user's working_storage_area. Data movement always begins at a record }
{ boundary and continues until either the end of the record or the end of}
{ the working_storage_area is encountered, whichever occurs first. If the}
{ entire record is not moved, an abnormal STATUS will be returned. In this}
{ event, subsequent amp$get_partial requests may be issued to obtain the}
{ remainder of the record.}
{
{       AMP$GET_NEXT (FILE_IDENTIFIER, WORKING_STORAGE_AREA,
{          WORKING_STORAGE_LENGTH, RECORD_LENGTH, TRANSFER_COUNT,
{          BYTE_ADDRESS, FILE_POSITION, STATUS)
{
{ FILE_IDENTIFIER: (input) This parameter specifies the file access
{       identifier established when the file was opened.
{
{ WORKING_STORAGE_AREA: (input) This parameter specifies the user's working
{       storage area.
{
{ WORKING_STORAGE_LENGTH: (input) This parameter specifies the
{       maximum number of bytes to be moved into the working storage area.
{       The value is typically set at the maximum record size defined for
{       the file.
{
{ RECORD_LENGTH: (output) This parameter specifies the actual size of
{       the record as it exists on the file.
{
{ TRANSFER_COUNT: (output) This parameter indicates to the user the
{       number of bytes moved to the user's working storage area.
{
{ BYTE_ADDRESS: (output) This parameter specifies the file byte address
{       associated with the record retrieved.  This value is only returned
{       for files on mass storage devices.
{
{ FILE_POSITION: (output) This parameter specifies the position of the file
{       following this request.  It can be used to recognize end of record
{       status.
{
{ STATUS: (output) This parameter specifies the request status.
{


{ * amxgetn}
  PROCEDURE [XREF] amp#get_next ALIAS 'amxgetn' (file_identifier:
    amt$file_identifier;
    working_storage_area: ^cell;
    working_storage_length: amt$working_storage_length;
    VAR record_length: amt$max_record_length;
    VAR transfer_count: amt$transfer_count;
    VAR byte_address: amt$file_byte_address;
    VAR file_position: amt$file_position;
    VAR status: ost$status);

*callc amdglob
*callc osdstat
```

2.5.1.7 AMXGETP (AMP$GET_PARTIAL)

```
{
{   The purpose of this request is to retrieve the next portion of a}
{ record from the current location of a file opened with amc$record access.}
{   Repeated requests may be required to move the whole record.}
{   The data is moved from buffers maintained by the access method to the}
{ user's working_storage_area.}
{
{       AMP$GET_PARTIAL (FILE_IDENTIFIER, WORKING_STORAGE_AREA,
{         WORKING_STORAGE_LENGTH, RECORD_LENGTH, TRANSFER_COUNT,
{         BYTE_ADDRESS, FILE_POSITION, SKIP_OPTION, STATUS)
{
{ FILE_IDENTIFIER: (input) This parameter specifies the file access
{       identifier established when the file was opened.
{
{ WORKING_STORAGE_AREA: (input) This parameter specifies the user's working
{       storage area.
{
{ WORKING_STORAGE_LENGTH: (input) This parameter specifies the number
{       of bytes to be moved into the working storage area.
{
{ RECORD_LENGTH: (output) This parameter will specify the actual size of
{       the record as it exists on the file when the last data of the
{       record is transferred.  It will be cumulative for a series of
{       partial record operations within a record.
{
{ TRANSFER_COUNT: (output) This parameter specifies the number
{       of bytes moved to the user's working storage area.
{
{ BYTE_ADDRESS: (output) This parameter specifies the file byte address
{       of the beginning of the record being processed as a series of
{       partial records.  This value is only returned for files which reside}
{       on mass storage devices.  It is not updated to point
{       to the beginning of each partial transfer.
{
{ FILE_POSITION: (output) This parameter specifies the position of the file
{       following this request.  It can be used to recognize end of record
{       status.
{
{ SKIP_OPTION: (input) This parameter specifies whether to advance to the
{       beginning of the next record before transferring data}
{       (amc$skip_to_eor) or to start the transfer at the current position}
{       (amc$no_skip). If the current position is beginning of record}
{       then this option has no effect.}
{
{
{ STATUS: (output) This parameter specifies the request status.
```

```
{


{ * amxgetp}
  PROCEDURE [XREF] amp#get_partial ALIAS 'amxgetp' (file_identifier:
    amt$file_identifier;
    working_storage_area: ^cell;
    working_storage_length: amt$working_storage_length;
    VAR record_length: amt$max_record_length;
    VAR transfer_count: amt$transfer_count;
    VAR byte_address: amt$file_byte_address;
    VAR file_position: amt$file_position;
    skip_option: amt$skip_option;
    VAR status: ost$status);

*callc amdglob
*callc osdstat
```

2.5.1.8 AMXGFAT (AMP$GET_FILE_ATTRIBUTES)

```
{
{     The purpose of this request is to allow a user to interrogate
{ file attributes maintained by the access method for a local file.
{     File attributes may be defined for a file in the following ways:
{     .  FILE command
{     .  amp$file request
{     .  access method supplied default
{     .  amp$store request
{}
{     File attributes, once established, are preserved with the file.
{ These become the default whenever the file is accessed.  Preserved
{ attributes may be redefined as mentioned above.
{     This request returns file attribute values which result from the
{ precedence relationships of FILE command, amp$file request, preserved
{ attributes and access method defaults.
{     This request is intended to be used prior to issuing an amp$file
{ or amp$open request.  Used in this manner, this request allows an
{ application to determine whether a file exists, and if it does exist,
{ whether the known file attributes are consistent with the processing
{ to follow.
{     This request may be issued at any time and will return the file_
{ attributes which would be in effect if the file were to be opened
{ immediately following this request.
{     If the file is not local, the file_attributes parameter is not
{ modified by this request.
{
{        AMP$GET_FILE_ATTRIBUTES (LOCAL_FILE_NAME, FILE_ATTRIBUTES,
{          LOCAL_FILE, EXISTING_FILE, CONTAINS_DATA, STATUS)
```

```
{
{ LOCAL_FILE_NAME: (input) This parameter specifies the name of the local
{       file whose attribute values are sought.
{
{ FILE_ATTRIBUTES: (input-output) This parameter specifies one or more
{       file attributes whose value is sought.
{
{ LOCAL_FILE: (output) This parameter specifies whether the file is
{       local to the task. A file is said to be local if a preceding command,
{       or request, has been issued naming the file.
{
{ EXISTING_FILE: (output) This parameter specifies whether the local file
{         has ever been opened.
{
{ CONTAINS_DATA: (output) This parameter specifies whether an existing
{       file has data (non_null in length).
{
{ STATUS: (output) This parameter specifies the request status.
{


{ * amxgfat}
  PROCEDURE [XREF] amp#get_file_attributes ALIAS 'amxgfat' (local_file_name:
    amt$file_name;
    VAR file_attributes: amt$file_attributes;
    VAR local_file: boolean;
    VAR existing_file: boolean;
    VAR contains_data: boolean;
    VAR status: ost$status);

*callc amdfatt
*callc amdglob
*callc amdname
*callc osdstat
```

2.5.1.9 AMXOPEN (AMP$OPEN)

```
{   The purpose of this request is to prepare a local file for access.}
{}
{   If the file exists and is permanent, this request must be preceded}
{ by an ATTACH or GET command or request to make the file local to the job.}
{}
{   File Attributes:}
{}
{   Requests such as amp$file and amp$get_file_attributes are optional}
{ prior to the call to amp$open. File attributes defined at the time a file}
{ is created are preserved with the file; these become the default when the}
{ file is opened, unless overridden. The FILE command and the amp$file}
{ request may be used to override the default file attributes. At open,
{ the attributes quoted on the FILE command will prevail over those named}
{ on an amp$file request whose attributes will in turn prevail over those}
{ preserved with the file. Use of the FILE command attributes may be}
{ controlled with the amp$file request.}
{}
{   Label Processing:}
{}
{   Standard ANSI labels are verified at open. If the user has specified a}
{ label_exit_procedure with the amp$file request, control will be passed}
{ to this procedure whenever a label matching the label_options attribute}
{ is found.}
{}
{   Instances of file:}
{}
{   An access method request which has local_file_name as the first formal}
{ parameter may be issued regardless of whether the file is open. All other}
{ requests may only be issued after a successful open. Amp$open returns}
{ a FILE_IDENTIFIER which identifies an instances of a file.  Each}
{ open of a file creates a unique instance of a file which is given}
{ a unique file_identifier.}
{   Each instance of a file is described by separate tables to allow}
{ independent, parallel access to multiple instances of the file by a task,}
{ by multiple tasks in the same job, or by tasks in different jobs.}
{   Only permanent files may be shared among jobs in the system. Any local}
{ file may be shared by tasks in the same job.}
{   Restrictions are placed upon a task opening the same file more than once}
{ without an intervening close.  These restrictions depend upon the}
{ device_class assigned to the file and the access_mode requested.}
{ Tape files cannot be multiply opened.  Disk and interactive files}
{ may be multiply opened.}
{   Each instance of a multiply opened disk file shares a common mass}
{ storage allocation. However, each accessor has a unique file_identifier}
{ with which a unique location within the file ( current_byte_address,}
{ file_position, volume_number, etc.) is maintained by the access method.}
{   Each instance of open of the same file must quote compatible}
{ access_level, record_type, and block_type.}
{   Open verifies that the access_level quoted by one instance of open is}
{ compatible with previous instances of open for the file in the same job.}
{ Record_access and segment_access may occur in parallel on the same file,}
{ if the two instances of open are in different tasks. Physical_access can}
{ coexist with neither record_access nor segment_access.}
{   Open verifies that the record_type quoted by one instance of open is}
{ compatible with previous instances of open in the same job. An instance of}
{ open with access_mode of only pfc$read is allowed to specify amc$undefined}
{ regardless of previous record_type specification. No other combinations}
{ of record_type may coexist.}
{   Open also verifies that the block_type quoted by one instance of open is}
{ compatible with previous instances of open in the same job. An instance}
{ of open with access_mode of only pfc$read access is allowed to specify a}
{ block_type other than the prevailing one.}
{   Only permanent files may be shared among jobs in the system.  Any}
{ local file may be shared by tasks in the same job.}
{}
{   File_sharing:}
{}
{   A task may insulate itself from conflict caused by other tasks in the}
{ same job sharing the local file by using the job_local_share_mode attribute}
{ of the amp$file request to restrict shared usage of the file.}
{}
{   Segment_access:}
{}
{   If segment access is quoted on an open request, the task must obtain a}
{ pointer to the segment in order to access it.  See the amp$get_segment_}
{ pointer request.}
{}
{   Open_positioning:}
{}
{   A task may use the open_position attribute of the amp$file}
{ request to cause the file to be positioned by amp$open to BOI,}
{ EOI or a partition boundary.}
{}
{   File_disposition:}
{}
{   Files opened by a task are automatically closed at its termination. Use of}
{ the amp$close request is optional.}
{   Prior to open, the file_disposition attribute of the amp$file request}
{ may be used to specify when file disposition is to occur.   A}
{ file_disposition of amc$task_local will cause disposition to occur}
{ at amp$close or task termination.  A file_disposition of}
{ amc$job_global will delay the disposition of the file until job}
{ termination.  The RETURN command/request causes immediate}
{ disposition and overrides the file_disposition selected.}
{
{       AMP$OPEN (LOCAL_FILE_NAME, ACCESS_LEVEL, FILE_IDENTIFIER, STATUS)
{
{ LOCAL_FILE_NAME: ( input) This parameter specifies the name of the local}
{       file to be opened for access.}
{
{ ACCESS_LEVEL: (input) This parameter specifies the means by which the}
{       file will be accessed.}
{
{ FILE_IDENTIFIER: (output) This parameter specifies the file access}
{       identifier which is assigned to this instance of open of the local}
{       file.}
```

```
{
{ STATUS: (ouput) This parameter specifies the request status.}
{


{ * amxopen}
  PROCEDURE [XREF] amp#open ALIAS 'amxopen' (local_file_name:
    amt$file_name;
    access_level: amt$access_level;
    VAR file_identifier: amt$file_identifier;
    VAR status: ost$status);

*callc amdglob
*callc amdname
*callc amdopen
*callc osdstat
```

2.5.1.10 <u>AMXPUTD (AMP$PUT_DIRECT)</u>

```
{
{   The purpose of this request is to transfer a record from the user's}
{ working storage area to a file opened with amc$record access_level}
{ and amc$byte_addressable file_organization.}
{   If the record_type is amc$ansi_fixed, the record is padding_character}
{ filled, if necessary.}
{   This request will establish a new end of information (EOI) if the sum}
{ of byte_address and working_storage_length exceeds the current EOI.}
{ It is the user's responsibility to ensure that a record being
{ replaced  is the same length as the original.
{
{       AMP$PUT_DIRECT (FILE_IDENTIFIER, WORKING_STORAGE_AREA,
{         WORKING_STORAGE_LENGTH, BYTE_ADDRESS, STATUS)
{
{ FILE_IDENTIFIER: (input) This parameter specifies the file access
{       identifier established when the file was opened.
{
{ WORKING_STORAGE_AREA: (input) This parameter specifies the user's
{       working storage area that contains the record to be transferred.
{
{ WORKING_STORAGE_LENGTH: (input) This parameter specifies the number
{       of bytes to be transferred.
{
{ BYTE_ADDRESS: (input) This parameter specifies the file byte address
{       of where the record is to be placed in the file.
{
{ STATUS: (output) This parameter specifies the request status.
{
```

```
{ * amxputd}
  PROCEDURE [XREF] amp#put_direct ALIAS 'amxputd' (file_identifier:
    amt$file_identifier;
    working_storage_area: ^cell;
    working_storage_length: amt$working_storage_length;
    byte_address: amt$file_byte_address;
    VAR status: ost$status);

*callc amdglob
*callc osdstat
```

2.5.1.11 AMXPUTN (AMP$PUT_NEXT)

```
{
{   The purpose of this request is to transfer a record from the user's
{ working storage area to the "next" location in a file opened with}
{ amc$record access_level.}
{   If the record_type is amc$ansi_fixed, the record is padding_character}
{ filled, if necessary.}
{   This request will unconditionally establish end of information (EOI)}
{ when issued on a file opened with amc$sequential file_organization.}
{}
{       AMP$PUT_NEXT (FILE_IDENTIFIER, WORKING_STORAGE_AREA,
{         WORKING_STORAGE_LENGTH, BYTE_ADDRESS, STATUS)
{
{ FILE_IDENTIFIER: (input) This parameter specifies the file access
{       identifier established when the file was opened.
{
{ WORKING_STORAGE_AREA: (input) This parameter specifies the user's working
{       storage area that contains the data to be transferred.
{
{ WORKING_STORAGE_LENGTH: (input) This parameter specifies the number
{       of bytes to be transferred.
{
{ BYTE_ADDRESS: (output) This parameter specifies the file byte address
{       established for the beginning of the record.  This address
{       may be used on future requests to randomly access the record.
{       This value is only returned for files on mass storage devices.
{
{ STATUS: (output) This parameter specifies the request status.
{


{ * amxputn}
  PROCEDURE [XREF] amp#put_next ALIAS 'amxputn' (file_identifier:
    amt$file_identifier;
    working_storage_area: ^cell;
    working_storage_length: amt$working_storage_length;
    VAR byte_address: amt$file_byte_address;
    VAR status: ost$status);

*callc amdglob
*callc osdstat
```

2.5.1.12 AMXPUTP (AMP$PUT_PARTIAL)

```
{
{   The purpose of this request is to transfer a partial record from the
{ user's working storage area to the "next" location in a file opened}
{ with amc$record access_level.}
{   This request may not be issued if the record_type is amc$ansi_variable}
{ or amc$ansi_fixed.}
{   This request is provided for exceedingly long records which would}
{ otherwise require an impractical working storage length.}
{    The first portion of the record is supplied with the amc$start}
{ term_option.  The amc$start may be issued when the file_position}
{ is amc$mid_record to terminate a previous record while starting a}
{ new one.  This request sets the file_position to amc$mid_record.}
{   Intermediate portions of the record are transferred with a term_option}
{ of amc$continue. This term_option may only be issued when the file_position}
{ is amc$mid_record. File_position remains amc$mid_record.}
{    The final portion of the record is supplied with amc$terminate}
{ term_option.}
{ This term_option may be issued when the file_position is amc$mid_record or}
{ amc$eor. In the latter case, a full record is being supplied. This request}
{ results in an amc$eor file_position.}
{}
{       AMP$PUT_PARTIAL (FILE_IDENTIFIER, WORKING_STORAGE_AREA,
{         WORKING_STORAGE_LENGTH, BYTE_ADDRESS, TERM_OPTION, STATUS)
{
{ FILE_IDENTIFIER: (input) This parameter specifies the file access
{       identifier established when the file was opened.
{
{ WORKING_STORAGE_AREA: (input) This parameter specifies the user's working
{       storage area that contains the data to be transferred.
{
{ WORKING_STORAGE_LENGTH: (input) This parameter specifies the number
{       of bytes to be transferred.
{
{ BYTE_ADDRESS: (output) This parameter specifies the byte address
{       established for the beginning of the record.  This address
{       may be used on future requests to randomly access the record.
{       This value is only returned for files on mass storage devices.
{
{ TERM_OPTION: (input) This parameter specifies which portion of the record}
{       is being supplied.}
{
{ STATUS: (output) This parameter specifies the request status.
{


{ * amxputp}
  PROCEDURE [XREF] amp#put_partial ALIAS 'amxputp' (file_identifier:
    amt$file_identifier;
    working_storage_area: ^cell;
    working_storage_length: amt$working_storage_length;
    VAR byte_address: amt$file_byte_address;
    term_option: amt$term_option;
    VAR status: ost$status);

*callc amdglob
*callc osdstat
```

2.5.1.13 AMXREWD (AMP$REWIND)

```
{
{   The purpose of this request is to reposition to the beginning of}
{ information of the file. This request has meaning only for files opened}
{ with amc$record or amc$physical access_level.}
{   Any modified data residing in memory is flushed to the current volume.}
{ Any outstanding no_wait requests are completed before the repositioning}
{ is attempted.}
{   If the label_type is amc$labelled and the preceding operation was an}
{ output, a standard ANSI EOF label group is written. If the HDR1 label}
{ corresponding to this file is not on the current volume, the current}
{ volume is dismounted and the volume containing the beginning of the file}
{ is mounted. The file is positioned beyond the verified HDR label group.}
{   If the label_type is amc$unlabelled or amc$non_standard_labelled, the}
{ current volume is rewound.}
{   If a label_exit_procedure was specified, control will be passed to this}
{ procedure if labels matching the label_options attribute are encountered}
{ during this process.}
{}
{       AMP$REWIND (FILE_IDENTIFIER, WAIT, STATUS)
{
{ FILE_IDENTIFIER: (input) This parameter specifies the file access
{       identifier established when the file was opened.
{
{ WAIT: (input) This parameter specifies the action to be taken
{       if the rewind can not be completed without causing
{       a physical I/O operation to be submitted.
{         wait: Don't return control until the operation is complete.
{         nowait: Return control to the user even though the operation
{                 may not be complete.  The AMP$FETCH_ACCESS_INFORMATION
{                 request may be used to determine the last_op_status.
{
{ STATUS: (output) This parameter specifies the request status.
{


{ * amxrewd}
  PROCEDURE [XREF] amp#rewind ALIAS 'amxrewd' (file_identifier:
    amt$file_identifier;
    wait: ost$wait;
    VAR status: ost$status);

*callc amdglob
*callc osdwnw
*callc osdstat
```

2.5.1.14 <u>AMXSEEK (AMP$SEEK_DIRECT)</u>

```
{
{   The purpose of this request is to change the current_byte_address of}
{ a file opened with amc$record access_level and amc$byte_addressable}
{ file_organization. This request does nothing to change the physical}
{ position of the mass storage device assigned to the file. This request}
{ is provided to allow the location in the file to be established prior}
{ to issuing an amp#get_next or amp#get_partial. Note that the requests}
{ such as amp#get_direct and amp#get_partial_direct perform an implicit}
{ seek_direct followed by record movement.}
{
{       AMP$SEEK_DIRECT (FILE_IDENTIFIER, BYTE_ADDRESS, STATUS)
{
{ FILE_IDENTIFIER: (input) This parameter specifies the file access
{       identifier established when the file was opened.
{
{ BYTE_ADDRESS: (input) This parameter specifies the file byte address of
{       the record to be located.
{
{ STATUS: (output) This parameter specifies the request status.
{

{amxseek}
  PROCEDURE [XREF] amp#seek_direct ALIAS 'amxseek' (file_identifier:
    amt$file_identifier;
    byte_adress: amt$file_byte_address;
    VAR status: ost$status);

*callc amdglob
*callc osdstat
```

2.5.1.15 AMXSTOR (AMP$STORE)

```
{
{   The purpose of this request is to change the value of one or more
{ file attributes. This request may only be issued after the file has
{ been opened.
{   If this request is issued by a user with control permission on the
{ file, any stored attributes will be preserved with the file. If the
{ user does not have control permission, the supplied attributes will
{ be used to describe the task's access to the file but will be discarded
{ when the task terminates.
{
{       AMP$STORE (FILE_IDENTIFIER, FILE_ATTRIBUTES, STATUS)
{
{ FILE_IDENTIFIER: (input) This parameter specifies the file access
{       identifier established when the file was opened.
{
{ FILE_ATTRIBUTES: (input-output) This parameter specifies one or more
{       attributes which are to be preserved with the file.
{
{ STATUS: (output) This parameter specifies the request status.
{


{ * amxstor}
  PROCEDURE [XREF] amp#store ALIAS 'amxstor' (file_identifier:
    amt$file_identifier;
    VAR file_attributes: amt$store_attributes;
    VAR status: ost$status);

*callc amdglob
*callc amdstor
*callc osdstat
```

2.5.2 NESTED COMMON DECKS

These common decks defin the data types and constants used by NOS/VE. They
automatically included through nested calls in the procedure common decks.

2.5.2.1 AMDBLKH

```
{ * amdblkh}
  CONST
    amc$max_block_header = 128 {bytes} ,
    amc$min_block_header = 16 {bytes} ;

  TYPE
    amt$block_header_length = amc$min_block_header .. amc$max_block_header,
    amt$block_header_type = (amc$tapemark_block, amc$data_block),
    amt$block_status = (amc$no_error, amc$recovered_error,
      amc$unrecovered_error),
    amt$pack_block_header = record
      header_type: amt$block_header_type,
      block_length: amt$max_block_length,
      previous_length: amt$max_block_length,
      unused_bit_count: amt$unused_bit_count,
    recend,
    amt$unpack_block_header = record
      header_type: amt$block_header_type,
      block_length_as_read: amt$max_block_length,
      block_length_as_written: amt$max_block_length,
      previous_length: amt$max_block_length,
      unused_bit_count: amt$unused_bit_count,
      block_status: amt$block_status,
    recend,
    amt$user_info_length = 0 .. (amc$max_block_header - amc$min_block_header);

*callc amdglob
```

2.5.2.2 AMDFATT

```
{ * amdfatt}
  TYPE
    amt$file_attributes = array[ * ] of amt$file_attribute,
    amt$file_attribute = record
      source {output} : amt$attribute_source,
      case key {input} : amt$file_attribute_keys of
{ One of the following attributes is output by amp$get_file_attributes or}
{ amp$fetch; exactly which one is determined by the case selector. The caller}
{ of amp$file must input one of the following attributes designated by the}
{ case selector.}
      =amc$access_mode=
        access_mode: pft$usage_selections,
      =amc$block_header_length=
        block_header_length: amt$block_header_length,
      =amc$block_type=
        block_type: amt$block_type,
      =amc$character_conversion=
        character_conversion: boolean,
      =amc$clear_space=
        clear_space: ost$clear_file_space,
      =amc$error_exit_procedure=
        error_exit_procedure: amt$error_exit_procedure,
      =amc$error_options=
        error_options: amt$error_options,
      =amc$file_command_processing=
        file_command_processing: boolean,
      =amc$file_content=
        file_content: amt$file_content,
      =amc$file_disposition=
        file_disposition: amt$file_disposition,
      =amc$file_limit=
        file_limit: amt$file_limit,
      =amc$file_organization=
        file_organization: amt$file_organization,
      =amc$horizontal_print_density=
        horizontal_print_density: amt$horizontal_print_density,
      =amc$internal_code=
        internal_code: amt$internal_code,
      =amc$job_local_share_mode=
        job_local_share_mode: pft$share_selections,
      =amc$label_exit_procedure=
        label_exit_procedure: amt$label_exit_procedure,
      =amc$label_options=
        label_options: amt$label_options,
      =amc$label_type=
        label_type: amt$label_type,
      =amc$line_number=
        line_number: amt$line_number,
      =amc$max_block_length=
        max_block_length: amt$max_block_length,
      =amc$max_record_length=
        max_record_length: amt$max_record_length,
      =amc$min_block_length=
        min_block_length: amt$min_block_length,
      =amc$min_record_length=
        min_record_length: amt$min_record_length,
      =amc$open_position=
        open_position: amt$open_position,
      =amc$owncode_procedure=
        owncode_procedure: pmt$program_name,
      =amc$padding_character=
        padding_character: amt$padding_character,
      =amc$page_format=
        page_format: amt$page_format,
      =amc$page_length=
        page_length: amt$page_length,
      =amc$page_width=
        page_width: amt$page_width,
      =amc$preset_value=
        preset_value: amt$preset_value,
      =amc$record_type=
        record_type: amt$record_type,
      =amc$ring_attributes=
        ring_attributes: amt$ring_attributes,
      =amc$suppress_buffering=
        suppress_buffering: boolean,
      =amc$transfer_unit=
        transfer_unit: amt$transfer_unit,
      =amc$user_info=
        user_info: amt$user_info,
      =amc$vertical_print_density=
        vertical_print_density: amt$vertical_print_density,
      casend
    recend;

*callc amdfile
*callc osdcfs
*callc pfdatrb
*callc pmdname
```

2.5.2.3 <u>AMDFILE</u>

```
{ * amdfile}
  CONST
    amc$max_user_info = 32;

  TYPE
    amt$file_attribute_keys = (amc$access_mode, amc$block_header_length,
      amc$block_type, amc$character_conversion, amc$clear_space,
      amc$error_exit_procedure, amc$error_options, amc$file_command_processing,
      amc$file_content, amc$file_disposition, amc$file_limit,
      amc$file_organization, amc$horizontal_print_density, amc$internal_code,
      amc$job_local_share_mode, amc$label_exit_procedure, amc$label_options,
      amc$label_type, amc$line_number, amc$max_block_length,
      amc$max_record_length, amc$min_block_length, amc$min_record_length,
      amc$open_position, amc$owncode_procedure, amc$padding_character,
      amc$page_format, amc$page_length, amc$page_width, amc$preset_value,
      amc$record_type, amc$ring_attributes, amc$suppress_buffering,
      amc$transfer_unit, amc$user_info, amc$vertical_print_density),
    amt$attribute_source = (amc$undefined_attribute, amc$access_method_default,
      amc$file_command, amc$file_request, amc$store_request,
      amc$preserved_attribute),
    amt$block_type = (amc$system_specified, amc$user_specified),
    amt$error_exit_procedure = ^procedure (file_identifier:
      amt$file_identifier),
    amt$error_options = (amc$terminate_file, amc$drop_record,
      amc$accept_record, amc$terminate_and_display, amc$drop_and_display,
      amc$accept_and_display),
    amt$file_content = record
      class: amt$file_class,
      kind: amt$file_kind,
      nature: amt$file_nature,
    recend,
    amt$file_disposition = (amc$task_local, amc$job_global),
    amt$file_limit = 0 .. amc$file_byte_limit,
    amt$file_organization = (amc$sequential, amc$byte_addressable,
      amc$index_sequential),
    amt$horizontal_print_density = (amc$ten_cpi, amc$twelve_cpi,
      amc$thirteen_cpi, amc$fifteen_cpi),
    amt$internal_code = (amc$as6, amc$as8, amc$ascii, amc$dis3, amc$dis4,
      amc$ebcdic, amc$bcd),
    amt$label_exit_procedure = ^procedure (file_identifier:
      amt$file_identifier),
    amt$label_options = set of (amc$vol1, amc$uvl, amc$hdr1, amc$hdr2,
      amc$eov1, amc$eov2, amc$uhl, amc$eof1, amc$eof2, amc$utl),
    amt$label_type = (amc$labelled, amc$non_standard_labelled, amc$unlabelled),
    amt$line_number = record
      length: 1 .. 999,
      location: 1 .. 999,
    recend,
    amt$min_block_length = 1 .. amc$maximum_block,
    amt$min_record_length = 0 .. amc$maximum_record,
    amt$padding_character = char,
    amt$page_format = (amc$continuous_form, amc$non_continuous_form),
    amt$page_length = 1 .. 65536,
    amt$page_width = 1 .. 65536,
    amt$preset_value = integer,
    amt$record_type = (amc$variable, amc$undefined, amc$ansi_fixed,
      amc$ansi_spanned, amc$ansi_variable),
    amt$ring_attributes = record
      r1: ost$ring,
      r2: ost$ring,
      r3: ost$ring,
    recend,
    amt$transfer_unit = (amc$t, amc$t1, amc$t2, amc$t4, amc$t8, amc$t16,
      amc$t32, amc$t64, amc$t128, amc$t256, amc$t512),
    amt$user_info = string (amc$max_user_info),
    amt$vertical_print_density = (amc$six_lpi, amc$eight_lpi, amc$nine_lpi,
      amc$twelve_lpi),
    amt$file_class = (amc$unknown_class, amc$data, amc$library),
    amt$file_kind = (amc$unknown_kind, amc$apl, amc$basic, amc$cobol,
      amc$cybil, amc$fortran, amc$pascal, amc$pli, amc$scl, amc$sympl,
      amc$cpu_assembler),
    amt$file_nature = (amc$unknown_nature, amc$source, amc$object, amc$list);

*callc amdblkh
*callc amdglob
*callc amdopos
*callc osdaddr
```

2.5.2.4 AMDFNFO

```
{ * amdfnfo}
  TYPE
    amt$access_info_keys = (amc$access_level, amc$allocation_unit_length,
      amc$block_number, amc$current_byte_address, amc$cycle_number,
      amc$device_class, amc$eoi_byte_address, amc$error_status,
      amc$file_position, amc$global_file_name, amc$global_share_mode,
      amc$last_operation, amc$last_op_status, amc$local_file_name,
      amc$record_header_length, amc$record_length, amc$residual_skip_count,
      amc$storage_class, amc$transfer_count, amc$volume_number),
    amt$access_information = array[ * ] of amt$access_info,
    amt$access_info = record
      case key {input} : amt$access_info_keys of {output}
      =amc$access_level=
        access_level: amt$access_level,
      =amc$allocation_unit_length=
        allocation_unit_length: amt$allocation_unit_length,
      =amc$block_number=
        block_number: amt$block_number,
      =amc$current_byte_address=
        current_byte_address: amt$file_byte_address,
      =amc$cycle_number=
        cycle_number: pft$cycle,
      =amc$device_class=
        device_class: rmt$device_class,
      =amc$eoi_byte_address=
        eoi_byte_address: amt$file_byte_address,
      =amc$error_status=
        error_status: ost$status_condition,
      =amc$file_position=
        file_position: amt$file_position,
      =amc$global_file_name=
        global_file_name: amt$global_file_name,
      =amc$global_share_mode=
        global_share_mode: pft$share_selections,
      =amc$last_operation=
        last_operation: amt$last_operation,
      =amc$last_op_status=
        last_op_status: amt$last_op_status,
      =amc$local_file_name=
        local_file_name: amt$local_file_name,
      =amc$record_header_length=
        record_header_length: amt$record_header_length,
      =amc$record_length=
        record_length: amt$max_record_length,
      =amc$residual_skip_count=
        residual_skip_count: amt$residual_skip_count,
      =amc$storage_class=
        storage_class: rmt$storage_class,
      =amc$transfer_count=
        transfer_count: amt$transfer_count,
      =amc$volume_number=
        volume_number: amt$volume_number,
      casend
    recend;

*callc amdinfo
*callc amdlop
*callc amdname
*callc amdopen
*callc amdrcdh
*callc osdstat
*callc pfdatrb
*callc rmdclas
```

2.5.2.5 AMDGLOB

```
{ * amdglob}
  CONST
    amc$file_byte_limit = 2147483648, { 2**31 bytes }
    amc$maximum_block = 16777216, { 2**24 bytes }
    amc$max_buffer_length = 16777216, { 2**24 bytes }
    amc$max_files = 2000,
    amc$maximum_record = amc$file_byte_limit;

  TYPE
    amt$access_method_pointer = amt$owncode_pointer,
    amt$buffer_length = 1 .. amc$max_buffer_length,
    amt$file_byte_address = 0 .. amc$file_byte_limit - 1,
    amt$file_identifier = record
      ordinal: amt$file_id_ordinal,
      sequence: amt$file_id_sequence,
      primary_pointer: amt$owncode_pointer,
      secondary_pointer: amt$owncode_pointer,
    recend,
    amt$file_id_ordinal = 1 .. amc$max_files,
    amt$file_id_sequence = 0 .. 4095,
    amt$file_position = (amc$bov, amc$boi, amc$mid_record, amc$eor,
      amc$prior_to_eop {after skip_back} , amc$eop, amc$eoi , amc$eov),
    amt$max_block_length = 1 .. amc$maximum_block,
    amt$max_record_length = 0 .. amc$maximum_record,
    amt$owncode_pointer = ^procedure (file_identifier: amt$file_identifier;
      call_block: amt$call_block_seq;
      access_method_pointer: amt$owncode_pointer;
      VAR status: ost$status),
    amt$call_block_seq = SEQ (REP 320 of cell),
    amt$physical_transfer_count = 0 .. amc$max_buffer_length,
    amt$skip_option = (amc$skip_to_eor, amc$no_skip),
    amt$term_option = (amc$start, amc$continue, amc$terminate),
    amt$transfer_count = 0 .. amc$maximum_record,
    amt$unused_bit_count = 0 .. 7,
    amt$working_storage_length = 0 .. amc$maximum_record;

*callc osdstat
```

2.5.2.6 AMDINFO

```
{ * amdinfo}
  CONST
    amc$max_allocation_unit = 8388608, {2**23 bytes, 1,048,576 words}
    amc$max_vol_number = 64,
    amc$minimum_addressable_unit = 2048; {bytes}


  TYPE
    amt$allocation_unit_length = amc$minimum_addressable_unit ..
      amc$max_allocation_unit,
    amt$block_number = 1 .. amc$file_byte_limit,
    amt$global_file_name = integer,
    amt$last_op_status = (amc$active, amc$complete),
    amt$residual_skip_count = amt$skip_count,
    amt$volume_number = 1 .. amc$max_vol_number;

*callc amdskip
```

2.5.2.7 AMDLOP

```
{ * amdlop}
  TYPE
    amt$last_operation = (amc$check_buffer_req, amc$close_req,
      amc$close_volume_req, amc$delete_req, amc$delete_direct_req,
      amc$fetch_access_info_req, amc$fetch_req, amc$flush_req,
      amc$get_direct_req, amc$get_label_req, amc$get_next_req,
      amc$get_partial_req, amc$get_partial_direct_req,
      amc$get_segment_pointer_req, amc$lock_file_req, amc$open_req,
      amc$pack_block_req, amc$pack_record_req, amc$put_direct_req,
      amc$put_label_req, amc$put_next_req, amc$put_partial_req,
      amc$put_partial_direct_req, amc$read_req, amc$read_direct_req,
      amc$read_direct_skip_req, amc$read_skip_req, amc$replace_req,
      amc$replace_direct_req, amc$rewind_req, amc$rewind_volume_req,
      amc$seek_direct_req, amc$set_segment_eoi_req, amc$set_segment_pos_req,
      amc$skip_req, amc$store_req, amc$unlock_file_req,
      amc$unpack_block_req, amc$unpack_record_req, amc$write_req,
      amc$write_direct_req, amc$write_eop_req, amc$write_tapemark_req);
```

2.5.2.8 AMDNAME

```
{ * amdname}
  TYPE
    amt$file_name = string ( * ),
    amt$local_file_name = ost$name;

*callc osdname
```

2.5.2.9 AMDOPEN

```
{ * amdopen}
  TYPE
    amt$access_level = (amc$physical, amc$record, amc$segment);
```

2.5.2.10 AMDOPOS

```
{ * amdopos}
  TYPE
    amt$open_position = (amc$open_no_positioning, amc$open_at_boi,
      amc$open_at_eop, amc$open_at_eoi);
```

2.5.2.11 AMDOWNC

```
{ * amdownc}
  TYPE
    amt$call_block = record
      case operation: amt$last_operation of
      =amc$check_buffer_req=
        check: amt$check_buffer_req,
      =amc$delete_direct_req=
        deld: amt$delete_direct_req,
      =amc$fetch_access_info_req=
        fetch_access: amt$fetch_access_info_req,
      =amc$fetch_req=
        fetch: amt$fetch_req,
      =amc$get_direct_req=
        getd: amt$get_direct_req,
      =amc$get_label_req=
        getl: amt$get_label_req,
      =amc$get_next_req=
        getn: amt$get_next_req,
      =amc$get_partial_req=
        getp: amt$get_partial_req,
      =amc$get_partial_direct_req=
        getpd: amt$get_partial_direct_req,
      =amc$get_segment_pointer_req=
        getsegp: amt$get_segment_pointer_req,
      =amc$lock_file_req=
        lock: amt$lock_file_req,
      =amc$open_req=
        open: amt$open_req,
      =amc$pack_block_req=
        packb: amt$pack_block_req,
      =amc$pack_record_req=
        packr: amt$pack_record_req,
      =amc$put_direct_req=
        putd: amt$put_direct_req,
      =amc$put_label_req=
        putl: amt$put_label_req,
      =amc$put_next_req=
        putn: amt$put_next_req,
      =amc$put_partial_req=
        putp: amt$put_partial_req,
      =amc$put_partial_direct_req=
        putpd: amt$put_partial_direct_req,
      =amc$read_req=
        rsq: amt$read_req,
      =amc$read_direct_req=
        rba: amt$read_direct_req,
      =amc$read_direct_skip_req=
        rbaskp: amt$read_direct_skip_req,
      =amc$read_skip_req=
        rsqskp: amt$read_skip_req,
      =amc$replace_req=
        replace: amt$replace_req,
      =amc$replace_direct_req=
        repld: amt$replace_direct_req,
      =amc$rewind_req=
        rewind: amt$rewind_req,
      =amc$rewind_volume_req=
        rewvol: amt$rewind_volume_req,
      =amc$seek_direct_req=
        seekd: amt$seek_direct_req,
      =amc$set_segment_eoi_req=
        segeoi: amt$set_segment_eoi_req,
      =amc$set_segment_pos_req=
        segpos: amt$set_segment_pos_req,
      =amc$skip_req=
        skp: amt$skip_req,
      =amc$store_req=
        store: amt$store_req,
      =amc$unpack_block_req=
        unpackb: amt$unpack_block_req,
      =amc$unpack_record_req=
        unpackr: amt$unpack_record_req,
      =amc$write_req=
        wsq: amt$write_req,
      =amc$write_direct_req=
        wba: amt$write_direct_req,
      casend,
    recend,
    amt$check_buffer_req = record
      buffer_area: ^cell,
      request_complete: ^boolean,
      byte_address: ^amt$file_byte_address,
      transfer_count: ^amt$physical_transfer_count,
      wait: ost$wait,
    recend,
    amt$delete_direct_req = record
      byte_address: amt$file_byte_address,
    recend,
    amt$fetch_access_info_req = record
      information: ^amt$access_information,
    recend,
    amt$fetch_req = record
      file_attributes: ^amt$file_attributes,
    recend,
    amt$get_direct_req = record
      working_storage_area: ^cell,
      working_storage_length: amt$working_storage_length,
      record_length: ^amt$max_record_length,
      transfer_count: ^amt$transfer_count,
      byte_address: amt$file_byte_address,
      file_position: ^amt$file_position,
    recend,
    amt$get_label_req = record
      label_area: ^cell,
      label_area_length: amt$label_area_length,
    recend,
    amt$get_next_req = record
      working_storage_area: ^cell,
      working_storage_length: amt$working_storage_length,
      record_length: ^amt$max_record_length,
      transfer_count: ^amt$transfer_count,
      byte_address: ^amt$file_byte_address,
      file_position: ^amt$file_position,
    recend,
    amt$get_partial_req = record
      working_storage_area: ^cell,
      working_storage_length: amt$working_storage_length,
      record_length: ^amt$max_record_length,
      transfer_count: ^amt$transfer_count,
      byte_address: ^amt$file_byte_address,
      file_position: ^amt$file_position,
      skip_option: amt$skip_option,
    recend,
    amt$get_partial_direct_req = record
      working_storage_area: ^cell,
      working_storage_length: amt$working_storage_length,
      record_length: ^amt$max_record_length,
      transfer_count: ^amt$transfer_count,
      byte_address: amt$file_byte_address,
      file_position: ^amt$file_position,
      skip_option: amt$skip_option,
    recend,
    amt$get_segment_pointer_req = record
      kind: amt$pointer_kind,
      pointer: ^amt$segment_pointer,
    recend,
    amt$lock_file_req = record
      status: ^amt$file_lock,
    recend,
    amt$open_req = record
      access_level: amt$access_level,
      file_identifier: ^amt$file_identifier,
      local_file_name: amt$file_name,
    recend,
    amt$pack_block_req = record
      buffer_area: ^cell,
      buffer_length: amt$buffer_length,
      header: amt$pack_block_header,
      user_information: ^cell,
      user_info_length: amt$user_info_length,
    recend,
    amt$pack_record_req = record
      buffer_area: ^cell,
      buffer_length: amt$buffer_length,
      header: amt$record_header,
    recend,
    amt$put_direct_req = record
      working_storage_area: ^cell,
      working_storage_length: amt$working_storage_length,
      byte_address: amt$file_byte_address,
    recend,
    amt$put_label_req = record
      label_area: ^cell,
      label_area_length: amt$label_area_length,
    recend,
    amt$put_next_req = record
      working_storage_area: ^cell,
      working_storage_length: amt$working_storage_length,
      byte_address: ^amt$file_byte_address,
    recend,
    amt$put_partial_req = record
      working_storage_area: ^cell,
      working_storage_length: amt$working_storage_length,
      byte_address: ^amt$file_byte_address,
      term_option: amt$term_option,
    recend,
    amt$put_partial_direct_req = record
      working_storage_area: ^cell,
      working_storage_length: amt$working_storage_length,
      byte_address: amt$file_byte_address,
      term_option: amt$term_option,
    recend,
    amt$read_req = record
      buffer_area: ^cell,
      buffer_length: amt$buffer_length,
      byte_address: ^amt$file_byte_address,
      transfer_count: ^amt$physical_transfer_count,
      wait: ost$wait,
    recend,
    amt$read_direct_req = record
      buffer_area: ^cell,
      buffer_length: amt$buffer_length,
      byte_address: amt$file_byte_address,
      transfer_count: ^amt$physical_transfer_count,
      wait: ost$wait,
    recend,
    amt$read_direct_skip_req = record
      buffer_area: ^cell,
      buffer_length: amt$buffer_length,
      byte_address: amt$file_byte_address,
      transfer_count: ^amt$physical_transfer_count,
      wait: ost$wait,
    recend,
    amt$read_skip_req = record
      buffer_area: ^cell,
      buffer_length: amt$buffer_length,
      byte_address: ^amt$file_byte_address,
      transfer_count: ^amt$physical_transfer_count,
      wait: ost$wait,
    recend,
    amt$replace_req = record
      working_storage_area: ^cell,
      working_storage_length: amt$working_storage_length,
    recend,
    amt$replace_direct_req = record
      working_storage_area: ^cell,
      working_storage_length: amt$working_storage_length,
      byte_address: amt$file_byte_address,
    recend,
    amt$rewind_req = record
      wait: ost$wait,
    recend,
    amt$rewind_volume_req = record
      wait: ost$wait,
    recend,
    amt$seek_direct_req = record
      byte_address: amt$file_byte_address,
    recend,
    amt$set_segment_eoi_req = record
      pointer: amt$segment_pointer,
    recend,
    amt$set_segment_pos_req = record
      pointer: amt$segment_pointer,
    recend,
    amt$skip_req = record
      direction: amt$skip_direction,
      units: amt$skip_units,
      count: amt$skip_count,
      file_position: ^amt$file_position,
    recend,
    amt$store_req = record
      file_attributes: amt$store_attributes,
    recend,
    amt$unpack_block_req = record
      buffer_area: ^cell,
      buffer_length: amt$buffer_length,
      header: ^amt$unpack_block_header,
      user_information: ^cell,
      user_info_length: ^amt$user_info_length,
    recend,
    amt$unpack_record_req = record
      buffer_area: ^cell,
      buffer_length: amt$buffer_length,
      pointer_kind: amt$pointer_kind,
      header: ^amt$record_header,
      data_address: ^amt$segment_pointer,
    recend,
    amt$write_req = record
      buffer_area: ^cell,
      buffer_length: amt$buffer_length,
      byte_address: ^amt$file_byte_address,
      wait: ost$wait,
    recend,
    amt$write_direct_req = record
      buffer_area: ^cell,
      buffer_length: amt$buffer_length,
      byte_address: amt$file_byte_address,
      wait: ost$wait,
    recend;

*callc amdblkh
*callc amdfatt
*callc amdfnfo
*callc amdlabl
*callc amdlock
*callc amdlop
*callc amdname
*callc amdopen
*callc amdrcdh
*callc amdsgpt
*callc amdskip
*callc amdstor
*callc osdwnw
```

2.5.2.12 AMDRCDH

```
{ * amdrcdh}
  CONST
    amc$max_record_header = 16;

  TYPE
    amt$record_header = record
      header_type: amt$record_header_type,
      length: amt$max_record_length,
      previous_length: amt$max_record_length,
      unused_bit_count: amt$unused_bit_count,
      user_information: cell,
    recend,
    amt$record_header_length = 0 .. amc$max_record_header,
    amt$record_header_type = (amc$full_record, amc$start_record,
      amc$continued_record, amc$end_record, amc$partition, amc$deleted_record);
```

2.5.2.13 AMDSKIP

```
{ * amdskip}
  TYPE
    amt$skip_count = 1 .. amc$file_byte_limit,
    amt$skip_direction = (amc$forward, amc$backward),
    amt$skip_units = (amc$skip_record, amc$skip_block, amc$skip_partition,
      amc$skip_tape_mark);
```

2.5.2.14 AMDSTOR

```
{ * amdstor}
  TYPE
    amt$store_attributes = array[ * ] of amt$store_attribute,
    amt$store_attribute = record
      case key: amt$file_attribute_keys of
      =amc$character_conversion=
        character_conversion: boolean,
      =amc$error_exit_procedure=
        error_exit_procedure: amt$error_exit_procedure,
      =amc$error_options=
        error_options: amt$error_options,
      =amc$file_content=
        file_content: amt$file_content,
      =amc$file_limit=
        file_limit: amt$file_limit,
      =amc$horizontal_print_density=
        horizontal_print_density: amt$horizontal_print_density,
      =amc$label_exit_procedure=
        label_exit_procedure: amt$label_exit_procedure,
      =amc$label_options=
        label_options: amt$label_options,
      =amc$line_number=
        line_number: amt$line_number,
      =amc$padding_character=
        padding_character: amt$padding_character,
      =amc$page_format=
        page_format: amt$page_format,
      =amc$page_length=
        page_length: amt$page_length,
      =amc$page_width=
        page_width: amt$page_width,
      =amc$suppress_buffering=
        suppress_buffering: boolean,
      =amc$transfer_unit=
        transfer_unit: amt$transfer_unit,
      =amc$user_info=
        user_info: amt$user_info,
      =amc$vertical_print_density=
        vertical_print_density: amt$vertical_print_density,
      casend,
    recend;

*callc amdfile
```

2.5.2.15 OSDADDR

```
{ * osdaddr}

{ NOS/180 address constants. }


CONST


  { Ring names. }

  osc$min_ring = 1, { Lowest ring number (most privledged). }
  osc$max_ring = 15, { Highest ring number (least privledged). }
  osc$invalid_ring = 0,
  osc$os_ring_1 = 1, { Reserved for Operating System. }
  osc$tmtr_ring = 2, { Task Monitor. }
  osc$tsrv_ring = 3, { Task services. }
  osc$sj_ring_1 = 4, { Reserved for system job. }
  osc$sj_ring_2 = 5,
  osc$sj_ring_3 = 6,
  osc$application_ring_1 = 7, { Reserved for application subsystems.}
  osc$application_ring_2 = 8,
  osc$application_ring_3 = 9,
  osc$application_ring_4 = 10,
  osc$user_ring = 11, { Standard user task. }
  osc$user_ring_1 = 12, { Reserved for user...O.S. requests available.}
  osc$user_ring_2 = 13,
  osc$user_ring_3 = 14, { Reserved for user...O.S. requests not available. }
  osc$user_ring_4 = 15;


{ Virtual address space dimensions. }

CONST
  osc$maximum_segment = 0fff(16),
  osc$maximum_offset = 7ffffffe(16), {!!! CYBIL BUG !!!}
  osc$max_segment_length = osc$maximum_offset + 1;


{ Global-local key lock definition. }

TYPE
  ost$gl_key = packed record
    global: boolean, { True if value is global key. }
    local: boolean, { True if value is local key. }
    key_lock: ost$key_lock_value, { Key or lock value. }
  recend,

  ost$key_lock_value = 0 .. 3f(16),


  { CYBER 180 forty eight bit PVA definition. }

  ost$ring = osc$invalid_ring .. osc$max_ring, { Ring number. }
  ost$valid_ring = osc$min_ring .. osc$max_ring, { Valid Ring Number. }
  ost$segment = 0 .. osc$maximum_segment, { Segment number. }
  ost$segment_offset = -osc$maximum_offset .. osc$maximum_offset,
    { Address offset. }

  ost$segment_length = 0 .. osc$max_segment_length,

  ost$relative_pointer = - 7fffffff(16) .. 7fffffff(16),

  ost$pva = packed record
```

```
    ring: ost$ring,
    seg: ost$segment,
    offset: ost$segment_offset,
  recend;
```

2.5.2.16 <u>OSDCFS</u>

```
{ * osdcfs}
{ Secure memory/file parameter }

TYPE
  ost$clear_file_space = boolean;
```

2.5.2.17 <u>OSDNAME</u>

```
{ OSDNAME   Type definitions for standard OS names. }

  CONST
    osc$name_size = 31;

  TYPE
    ost$name_size = 1 .. osc$name_size;

  TYPE
    ost$name = STRING (osc$name_size);
```

2.5.2.18 <u>OSDSTAT</u>

```
CONST
  osc$max_condition = 999999,
  osc$status_parameter_delimiter = '"';

TYPE
  ost$status_condition = 0 .. osc$max_condition;

TYPE
  ost$status = record
    case normal: boolean of
    =false=
      identifier: string (2),
      condition: ost$status_condition,
      text: ost$string,
    casend,
  recend;
```

```
*callc osdstr
```

2.5.2.19 <u>OSDSTR</u>

```
*callc zoststr

{ OSDSTR    Type definitions for the 'text' portion of the status record }

  TYPE
    ost$string = record
      size: ost$string_length,
      value: string (osc$max_string_length),
    recend;
```

2.5.2.20 <u>OSDWNW</u>

```
{ * osdwnw}
{ Asynchronous request parameter: used by all NOS/180 requests that }
{ can be performed asynchronously to indicate whether the caller }
{ wishes to execute the request synchronously or asynchronously. }

TYPE
  ost$wait = (osc$wait, osc$nowait);
```

2.5.2.21 <u>PFDATRB</u>

```
{ * pfdatrb}
  CONST
    pfc$maximum_cycle_number = 999;


  TYPE
    pft$cycle = 1 .. pfc$maximum_cycle_number,

    pft$permit_options = (pfc$read, pfc$shorten, pfc$append, pfc$modify,
      pfc$execute, pfc$add_cycle, pfc$control),

    pft$usage_options = pfc$read .. pfc$execute,
    pft$usage_selections = set of pft$usage_options,

    pft$share_options = pfc$read .. pfc$execute,
    pft$share_selections = set of pft$share_options;
```

2.5.2.22 PMDNAME

```
{ * pmdname}
TYPE
  pmt$program_name = ost$name;

*callc osdname
```

2.5.2.23 RMDCLAS

```
{ * rmdclas}
  TYPE
    rmt$device_class = (rmc$mass_storage_device, rmc$magnetic_tape_device,
      rmc$terminal_device),
    rmt$storage_class = (rmc$temporary_device, rmc$permanent_device,
      rmc$queue_device);
```

2.6 IDIOSYNCRACIES OF THE SIMULATED NOS/VE I/O SYSTEM

2.6.1 NOS/170 FILE STRUCTURE "MARKS"

Neither SIMULATED NOS/VE I/O nor the NOS/170 INTERFACE create or support the detection of the NOS/170 file structuring marks "end_of_record" or "end_of_file (logical)" with the amc$variable record type. Neither SIMULATED NOS/VE I/O nor the NOS/170 INTERFACE generate the "end_of_partition" described in the NOS/VE ERS, nor does it detect such a structure for any record type.

2.6.1.1 SIMULATED NOS/VE I/O

The structuring marks, "end_of_record" and "end_of_file" are detected with the "interchange" file format having a record type of AMC$UNDEFINED and an "end_of_partition" status is returned whenever either type of mark is detected.

2.6.1.2 NOS/170 INTERFACE I/O

The NOS/170 INTERFACE is the same as SIMULATED NOS/VE I/O for the "amc$variable" record_type files in that it does not support the detection or creation of any NOS/170 structure marks. If such a mark is encountered the interface will abort the program.

2.6.2 NOS/170 FILES

The only capacity provided by the SIMULATED NOS/VE I/O system is that of reading, writing, positioning, opening, and closing files. The SIMULATED NOS/VE I/O system does <u>not</u> provide any of the following functions:

Permanent File Manager Functions
  GET, ATTACH, DEFINE, etc.
Local File Manager functions.
Dispose Processor (DSP) functions.

In general, the user must perform all file preparation functions prior to using the SIMULATED NOS/VE I/O system.

2.7 EXAMPLES OF FILE USAGE

2.7.1 NOS/VE DEFAULT FILE TYPE

```
 MODULE  amp_example_deck ;

   ?? set (list := off) ??
 *callc amxclse
 *callc amxfile
 *callc amxfnfo
 *callc amxftch
 *callc amxgetd
 *callc amxgetn
 *callc amxgetp
 *callc amxgfat
 *callc amxopen
 *callc amxputd
 *callc amxputn
 *callc amxputp
 *callc amxrewd
 *callc amxstor
   ?? set (list := on) ??

 PROGRAM amp_example;
   CONST
     in_file_name = 'source',
     out_file_name = 'fmt180',
     comm_file_name = 'output',
     max_rec_size = 1000;

   TYPE
     file = amt$file_identifier,
     wsa_record = string(*),
     wsa = ^wsa_record;

   VAR
     in_fid : file,
     out_fid : file,
     comm_fid : file,
     file_attributes : ^amt$file_attributes,
     wsa_ptr : wsa,
     wsa_length : [static] integer := max_rec_size,
     wsa_out_len : 1 .. max_rec_size,
     rec_length : amt$max_record_length,
     tr_count : amt$transfer_count,
     skip : [static] amt$skip_option := amc$no_skip,
     no_such_file_msg : [STATIC] STRING(21) := 'SOURCE not local_file',
     bum_gfa_msg : [STATIC] STRING(23) := 'bad GET_FILE_ATTRIBUTES',
     id_msg : [STATIC] STRING(60) :=
         ' testing GET_PARTIAL, PUT_PARTIAL, PUT_NEXT.  1000 cha~ rec.',
     start_program_msg : [STATIC] STRING(20) := 'program start AMPEX0',
     end_program_msg : [STATIC] STRING(18) := 'end program AMPEX]',
     term : amt$term_option,
     file_pos : amt$file_position,
     cba : amt$file_byte_address,
     stat : ost$status,
     local_file: BOOLEAN,
     existing_file: BOOLEAN,
     contains_data: BOOLEAN;

   /program#/
   begin
     stat.normal := TRUE;
 {  Allocate space for record area.  }
     ALLOCATE wsa_ptr : [ max_rec_size ] ;
     IF wsa_ptr = nil THEN
       EXIT  /program#/;
     IFEND;
 {  Allocate space for file_attributes record.  }
     ALLOCATE file_attributes : [ 1 .. 5 ] ;
     IF file_attributes = nil THEN
       EXIT /program#/;
     IFEND;
     /file#/
     begin
       file_attributes^[1].key := amc$access_mode;
       file_attributes^[2].key := amc$block_type;
       file_attributes^[3].key := amc$file_organization;
       file_attributes^[4].key := amc$open_position;
       file_attributes^[5].key := amc$record_type;
 {  Initialize  file_attributes record for communications file. }
       amp$get_file_attributes ( comm_file_name, file_attributes^,
           local_file, existing_file, contains_data, stat );
       file_attributes^[1].access_mode :=
           $pft$usage_selections [ pfc$append ];
       file_attributes^[4].open_position := amc$open_at_eoi;
       amp$file ( comm_file_name, file_attributes^, stat );
       amp$open ( comm_file_name, amc$record, comm_fid, stat );
       amp$put_next (comm_fid, #LOC(start_program_msg),
           #SIZE(start_program_msg), cba, stat);
       amp$put_next (comm_fid, #LOC(id_msg), #SIZE(id_msg), cba, stat);
 {  Initialize file_attributes record for input file.  }
       amp$get_file_attributes ( in_file_name, file_attributes^,
           local_file, existing_file, contains_data, stat );
```

```
      IF NOT local_file THEN
        amp$put_next (comm_fid, #LOC(no_such_file_msg),
            #SIZE(no_such_file_msg), cba, stat);
        EXIT /file#/
      IFEND;
      file_attributes^[1].access_mode := $pft$usage_selections[ pfc$read ];
      amp$file ( in_file_name, file_attributes^, stat );
      amp$open ( in_file_name, amc$record, in_fid, stat) ;
 {  Initialize user file description record for output file.  }
      file_attributes^[1].access_mode :=
          $pft$usage_selections[ pfc$shorten, pfc$append, pfc$modify ];
      amp$file ( out_file_name, file_attributes^, stat );
      amp$open (out_file_name, amc$record, out_fid, stat) ;
      term := amc$start;

 {  This is the main data copying loop.  }
      /main_loop/
      WHILE TRUE DO
        amp$get_partial ( in_fid, #LOC(wsa_ptr^), wsa_length,
          rec_length, tr_count, cba, file_pos, skip, stat);
        wsa_out_len := tr_count;
        CASE file_pos OF
        =amc$eoi=
          amp$put_next (comm_fid, #LOC(end_program_msg),
              #SIZE(end_program_msg), cba, stat);
          exit /main_loop/;
        =amc$eor=
          IF (term = amc$start) THEN
            amp$put_next (out_fid, #LOC(wsa_ptr^), wsa_out_len,
                          cba, stat)
          ELSE
            term := amc$terminate;
            amp$put_partial (out_fid, #LOC(wsa_ptr^), wsa_out_len,
                             cba, term, stat );
          IFEND;
          term := amc$start;
        =amc$mid_record=
          amp$put_partial (out_fid, #LOC(wsa_ptr^), wsa_out_len, cba,
                           term, stat );
          term := amc$continue;
 { NOTE: no other statuses can logically occur. }
        ELSE
          CYCLE /main_loop/;
        CASEND ;
      WHILEND /main_loop/ ;
 { Close input and output files. }
      amp$close (in_fid, stat) ;
      amp$close (out_fid, stat) ;
```

```
{ Just for kicks, try to open the ms-files again. }
      amp$get_file_attributes ( in_file_name, file_attributes^,
          local_file, existing_file, contains_data, stat );
      amp$open (in_file_name, amc$record, in_fid, stat);

      amp$get_file_attributes ( out_file_name, file_attributes^,
          local_file, existing_file, contains_data, stat );
      IF NOT existing_file OR NOT local_file THEN
        amp$put_next ( comm_fid, #LOC(bum_gfa_msg),
            #SIZE(bum_gfa_msg), cba, stat );
      IFEND;
      amp$open (out_file_name, amc$record, out_fid, stat);

      amp$close (in_fid, stat);
      amp$close (out_fid, stat);
      amp$close (comm_fid, stat);
    END /file#/;
{ Free the allocated storage areas. }
    free wsa_ptr ;
    free file_attributes;
  END /program#/;
PROCEND amp_example ;
MODEND amp_example_deck;
```

2.7.2 NOS/170 INTERCHANGE FILE TYPE

```
MODULE amp_example_deck ;

  ?? set (list := off) ??
*callc amxclse
*callc amxftch
*callc amxfnfo
*callc amxfile
*callc amxgetd
*callc amxgetn
*callc amxgetp
*callc amxgfat
*callc amxopen
*callc amxputd
*callc amxputn
*callc amxputp
*callc amxrewd
*callc amxseek
*callc amxstor
  ?? set (list := on) ??

VAR
```

```
   display_flag : [XDCL] BOOLEAN := FALSE,
   display_code_character: STRING(64) :=
       ':ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789+-*/()$= ,.#[]%"_!&''?<>@\^;',
   special_display_codes: STRING(64) :=
       ' abcdefghijklmnopqrstuvwxyz{|}~                                  ',
   output_line: STRING(64) :=
       '          w   oo oo oo oo oo oo oo oo oo oo    a a a a a a a a a a',
   title_line: STRING(64) :=
       '1    WORD   OCTAL DATA                         DISPLAY CODE CHARS.';

 CONST
   octal_data = 13,
   ascii_data = 46,
   address_data = 9;

 PROGRAM amp_example ;
   CONST
     in_file_name = 'source',
     out_file_name = 'icfile',
     ve_file_name = 'fmt180',
     comm_file_name = 'output',
     max_rec_size = 64;

   TYPE
     file = amt$file_identifier,
     wsa_word = RECORD
       two_bit_field: PACKED ARRAY [ -1 .. 30 ] OF 0 .. 3,
     RECEND,
     wsa_record = RECORD
       word : PACKED ARRAY[*] OF wsa_word,
     RECEND,
     wsa = ^wsa_record;

   VAR
     i,j,k: INTEGER,
     in_fid : file,
     out_fid : file,
     ve_fid : file,
     comm_fid : file,
     file_attributes: ^amt$file_attributes,
     dec_char : INTEGER,
     wsa_ptr : wsa,
     wsa_out_len : INTEGER,
     rec_length : amt$max_record_length,
     tr_count : amt$transfer_count,
     skip : [static] amt$skip_option := amc$no_skip,
     no_such_file_msg : [STATIC] STRING(21) := 'SOURCE not local_file',
     start_program_msg : [STATIC] STRING(20) := 'program start AMPEXA',
     end_program_msg : [STATIC] STRING(18) := 'end program AMPEX1',
     id_msg: [STATIC] STRING(49) :=
         ' testing INTERCHANGE FILES for GET_NEXT, PUT_NEXT',
     escape76 : BOOLEAN,
     escape74 : BOOLEAN,
     zero_byte : BOOLEAN,
     file_pos : amt$file_position,
     cba : amt$file_byte_address,
     sba : amt$file_byte_address,
     stat : ost$status,
     local_file: BOOLEAN,
     existing_file: BOOLEAN,
     contains_data: BOOLEAN,
     word_address: INTEGER;

   /program#/
   begin
 {  Allocate space for record area.  }
     ALLOCATE wsa_ptr : [ 1 .. max_rec_size ] ;
     IF wsa_ptr = nil THEN
       EXIT  /program#/;
     IFEND;
 {  Allocate space for file_attributes record.  }
     ALLOCATE file_attributes: [ 1 .. 5 ] ;
     IF file_attributes = nil THEN
       EXIT /program#/;
     IFEND;
     /file#/
     begin
       file_attributes^[1].key := amc$access_mode;
       file_attributes^[2].key := amc$block_type;
       file_attributes^[3].key := amc$file_organization;
       file_attributes^[4].key := amc$open_position;
       file_attributes^[5].key := amc$record_type;
 {  Initialize file_attributes record for communications file. }
       amp#get_file_attributes ( comm_file_name, file_attributes^,
           local_file, existing_file, contains_data, stat );
       file_attributes^[1].access_mode :=
           $pft$usage_selections [ pfc$append ];
       file_attributes^[4].open_position := amc$open_at_eoi;
       amp#file ( comm_file_name, file_attributes^, stat );
       amp#open ( comm_file_name, amc$record, comm_fid, stat );
       amp#put_next (comm_fid, #LOC(start_program_msg),
           #SIZE(start_program_msg), cba, stat);
       amp#put_next (comm_fid, #LOC(id_msg), #SIZE(id_msg), cba, stat);
 {  Initialize file_attributes record for input interchange file.  }
       amp#get_file_attributes (in_file_name, file_attributes^,
           local_file, existing_file, contains_data, stat );
       IF (local_file <> TRUE) THEN
         amp#put_next (comm_fid, #LOC(no_such_file_msg),
             #SIZE(no_such_file_msg), cba, stat);
         EXIT /file#/
       IFEND;
       file_attributes^[1].access_mode :=
           $pft$usage_selections [ pfc$read ];
       file_attributes^[2].block_type := amc$user_specified;
       file_attributes^[3].file_organization := amc$sequential;
       file_attributes^[4].open_position := amc$open_at_boi;
       file_attributes^[5].record_type := amc$undefined;
       amp#file ( in_file_name, file_attributes^, stat );
       amp#open (in_file_name, amc$record, in_fid, stat) ;
 {  Initialize file_attributes record for output interchange file.  }
       file_attributes^[1].access_mode :=
           $pft$usage_selections [ pfc$shorten, pfc$append ];
       amp#file ( out_file_name, file_attributes^, stat );
       amp#open (out_file_name, amc$record,  out_fid, stat) ;
 { Initialize file_attributes record for NOS/VE format file. }
       file_attributes^[1].access_mode :=
           $pft$usage_selections [ pfc$shorten, pfc$modify ];
       file_attributes^[2].block_type := amc$system_specified;
       file_attributes^[3].file_organization := amc$sequential;
       file_attributes^[4].open_position := amc$open_at_boi;
       file_attributes^[5].record_type := amc$variable;
       amp#file ( ve_file_name, file_attributes^, stat );
       amp#open (ve_file_name, amc$record, ve_fid, stat);

       escape76 := FALSE;
       escape74 := FALSE;
       word_address := 0;

 {  This is the main data copying loop.  }
       /main_loop/
       WHILE TRUE DO
         amp#get_next ( in_fid, #LOC(wsa_ptr^), #SIZE(wsa_ptr^),
           rec_length, tr_count, sba, file_pos, stat);
         wsa_out_len := tr_count ;
         CASE file_pos OF
         =amc$eoi=
           amp#put_next (comm_fid, #LOC(end_program_msg),
               #SIZE(end_program_msg), cba, stat);
           exit /main_loop/;
         =amc$eor,amc$mid_record,amc$eop=
           amp#put_next (out_fid, #LOC(wsa_ptr^), wsa_out_len,
               cba, stat);

           IF display_flag THEN
 { Convert data from 6-bit display codes to ascii characters. }
            FOR i := 1 to tr_count DIV 8 DO
              zero_byte := FALSE;
              FOR j := 1 TO 10 DO
                k := 3 * j - 2;
                dec_char := 16 * wsa_ptr^.word[i].two_bit_field[k] +
                             4  * wsa_ptr^.word[i].two_bit_field[k+1] +
                                  wsa_ptr^.word[i].two_bit_field[k+2] ;
                k := ascii_data + 2 * ( j - 1);
                IF escape76 = FALSE THEN
                  IF escape74 = FALSE THEN
                    IF dec_char = 62 THEN
                      escape76 := TRUE;
                      output_line(k) := ' ';
                    ELSE
                      IF dec_char = 60 THEN
                        escape74 := TRUE;
                        output_line(k) := ' ';
                      ELSE
                        output_line(k) := display_code_character(dec_char+1);
                      IFEND;
                    IFEND;
                  ELSE
                    IF dec_char = 1 THEN
                      output_line(k) := '@';
                    ELSE
                      IF dec_char = 2 THEN
                        output_line(k) := '^';
                      ELSE
                        IF dec_char = 4 THEN
                          output_line(k) := ':';
                        ELSE
                          output_line(k) := ' ';
                        IFEND;
                      IFEND;
                    IFEND;
                    escape74 := FALSE;
                  IFEND;
                ELSE
                  output_line(k) := special_display_codes(dec_char+1);
                  escape76 := FALSE;
                IFEND;
   { Generate octal data output. }
                k := octal_data + 3 * ( j - 1 );
                output_line(k) := CHR ( dec_char DIV 8 + 30(16) ) ;
                output_line(k+1) := CHR ( dec_char MOD 8 + 30(16) ) ;
              FOREND;
   { Generate the word_address of the file. }
```

```
               dec_char := sba DIV 8 + i - 1;
               k := address_data ;
               REPEAT
                 output_line(k) := CHR ( dec_char MOD 10 + 30(16) ) ;
                 dec_char := dec_char DIV 10 ;
                 k := k - 1 ;
               UNTIL dec_char = 0 ;
   { Check for zero_byte_terminated line. }
               k := ascii_data + 18;
               /zero_byte_check/
               REPEAT
                 IF output_line(k) = ':' THEN
                   output_line(k) := ' ';
                   k := k - 2 ;
                 ELSE
                   EXIT /zero_byte_check/;
                 IFEND;
               UNTIL k < ascii_data;
               IF (( word_address + i - 1 ) MOD 50 ) = 0 THEN
                 IF display_flag THEN
                   amp#put_next (ve_fid, #LOC(title_line), #SIZE(title_line),
                       cba, stat);
                   amp#put_next (comm_fid, #LOC(title_line), #SIZE(title_line),
                       cba, stat);
                 IFEND;
               IFEND;
               IF display_flag THEN
                 amp#put_next (ve_fid, #LOC(output_line), #SIZE(output_line),
                     cba, stat );
                 amp#put_next ( comm_fid, #LOC(output_line), #SIZE(output_line),
                     cba, stat );
               IFEND;
             FOREND;
           IFEND;
           word_address := word_address + tr_count DIV 8;
 { NOTE: no other statuses can logically occur. }
         ELSE
           CYCLE /main_loop/;
         CASEND ;
       WHILEND /main_loop/ ;
 { Close input and output files. }
       amp#close (in_fid, stat) ;
       amp#close (out_fid, stat) ;
       amp#close (ve_fid, stat);
       amp#close (comm_fid, stat);
     END /file#/;
 { Free the allocated storage areas. }
     free wsa_ptr ;
     free file_attributes;
   END /program#/;
 PROCEND amp_example ;
 MODEND amp_example_deck;
```

2.8 <u>INTERACTIVE FILES</u>

2.8.1 SIMULATED NOS/VE I/O

SIMULATED NOS/VE I/O detects any file connected to a terminal when the file is opened and all connected files are supported. The user must insure that the terminal connection does not change when the SIMULATOR is running.

Terminal input and output are provided through the Input/Output Control interface (IOC) of SES. All data input or output will appear on the 'SESLOG' file too.

When the user program requests terminal input from SIMULATED NOS/VE I/O, the preface "SIMULATED INPUT:" appears on the OUTPUT file just prior to the data input request from the INPUT file. This preface serves to differentiate input being sent as data to the program being simulated from input that is processed by the SIMULATOR as a command.

2.8.2 NOS/170 INTERFACE I/O

The NOS/170 INTERFACE detects any file connected to a terminal when the file is opened. Data transmission with all such files is supported. The only restriction is that terminal connection must not change during the course of execution.

2.9 SIMULATED NOS/VE <-> NOS/170 FILE CONVERSIONS

These procedures generate or convert only SIMULATED NOS/VE files written sequentially.

2.9.1 CONVERTING A NOS/170 FILE TO SIMULATED NOS/VE FORMAT

A procedure is available to convert NOS/170 source files (containing character data) to SIMULATED NOS/VE format. The procedure is callable under SES via the command:

```
SES.TO180,I=ILFN,O=OLFN,CS612 or CS64
```

ILFN is the input (NOS/170) logical file name and OLFN is the output (SIMULATED NOS/VE format) logical file name. The optional parameters CS612 and CS64 indicate the character set of the input file. If this parameter is absent, CS612 is assumed. CS612 indicates the NOS/170 6/12 ASCII character set. CS64 indicates NOS/170 display code or 6-bit ASCII character set. Note: The input file must be in "Z" type NOS/170 format.
This procedure can convert only character data. It is not possible to transport binary information from a NOS/170 format file to a SIMULATED NOS/VE format file.

2.9.2 CONVERTING A SIMULATED NOS/VE FILE TO NOS/170 FORMAT

A procedure is available to convert a SIMULATED NOS/VE file (containing character data) to NOS/170 format. The procedure is callable under SES via the command:

```
SES.FROM180,I=ILFN,O=OLFN,CS612 or CS64
```

ILFN is the input (SIMULATED NOS/VE) logical file name and OLFN is the output (NOS/170 format) logical file name. The optional parameters CS612 and CS64 indicate the character set of the output file. If this parameter is absent CS612 is assumed. CS612 indicates the NOS/170 6/12 ASCII character set. CS64 indicates NOS/170 display code. Note: The output file will be in NOS/170 "Z" format.
This procedure cannot convert binary data from SIMULATED NOS/VE files to NOS/170 files.

2.9.3 INSPECTING A SIMULATED NOS/VE FORMAT FILE

A procedure is available to inspect the contents of a SIMULATED NOS/VE format file. The procedure prints a hexadecimal and an the ascii character corresponding to each 8-bit byte of data on the file. The procedure additionally diagnoses any errors which prevent the file from being converted by the FROM180 procedure. The procedure is callable under SES via the command:

```
SES.DUMP180 I=ILFN O=OLFN CS612 or CS64 FW=first LW=last
```

ILFN is the input, SIMULATED NOS/VE format file and OLFN is the output file. If OLFN is omitted then "OLFN=OUTPUT" is assumed. CS612 and CS64 are keywords to indicate the character set of the output file. If no keyword is given then CS612 is assumed. CS612 indicates the NOS 6/12 ASCII character set and CS64 indicates the 64 character display code character set. FW indicates the first NOS/170 word to dump and LW indicates the last NOS/170 word to dump.

The output format is suitable for printing and is as follows:

```
1 WORD    BYTE       HEXADECIMAL DATA          ASCII DATA
 ADDRESS ADDRESS   F  0  1  2  3  4  5  6    0 1 2 3 4 5 6
    0        0     0 00 00 00 00 00 00 00
    1        7     0 07 41 4D 44 55 46 44      A M D U F D
    2        E     1 49 00 00 00 07 00 00    I
    3       15     0 00 06 43 4F 4D 4D 4F        C O M M O
    4       1C     1 4E 00 00 00 06 00 00    N
    5       23     0 00 43 7B 20 54 68 69        {   T h i
    6       2A     0 73 20 63 6F 6D 6D 6F    s   c o m m o
    7       31     0 6E 20 64 65 63 6B 20    n   d e c k
```

Note: The "F" column of hexadecimal data is 4-bits of fill and a non-zero digit here indicates the next record header on the file starts "f" bytes from the leftmost byte of the word.

2.10 NOS/170 INTERFACE <-> NOS/170 FILE CONVERSIONS

These procedures generate or convert only NOS/170 INTERFACE files written sequentially.

2.10.1 CONVERTING A NOS/170 FILE TO NOS/170 INTERFACE FORMAT

A procedure is available to convert NOS/170 source files in "Z" type format (containing only character data) to NOS/170 INTERFACE format. The procedure is callable under SES via the command:

```
SES.TO170,I=ILFN,O=OLFN,CS612 OR CS64
```

ILFN is the input (NOS/170) local file name and OLFN is the output (NOS/170 INTERFACE format) logal file name. The optional parameters CS612 and CS64 indicate the character set of the input file. If this parameter is absent CS612 is assumed. CS612 indicates the NOS/170 6/12 ASCII character set. CS64 indicates the NOS/170 display code or 6-bit ASCII character set.
This program converts only character data. It is not possible to record binary information on a zero byte terminated file.

2.10.2 CONVERTING A NOS/170 INTERFACE FILE TO NOS/170 FORMAT

A procedure is available to convert a NOS/170 INTERFACE format file (containing only character data) to a NOS/170 "Z" type file. The procedure is callable under SES via the command:

```
SES.FROM170,I=ILFN,O=OLFN,CS612 or CS64
```

ILFN is the input (NOS/170 INTERFACE format) local file name and OLFN is the output (NOS/170 format) local file name. The optional keywords CS612 and CS64 indicate the character set of the output file. If this parameter is absent CS612 is assumed. CS612 indicates the NOS/170 6/12 ASCII character set. CS64 indicates NOS/170 display code. The output file will be a "zero byte terminated" format.
This program cannot convert binary data.

2.10.3 INSPECTING A NOS/170 INTERFACE FILE

A procedure is available to dump the contents of a NOS/170 INTERFACE format file. The procedure produces both a hexadecimal dump and an ASCII character dump of each word of the file. This procedure also diagnoses any errors which might prevent conversion of the file by the FROM170 procedure. This procedure is callable under SES via the command:

```
      SES.DUMP170 I=ILFN O=OLFN CS612 or CS64 FW=first LW=last
```

ILFN is the input, NOS/170 INTERFACE format file and OLFN is the output file. If OLFN is omitted "OLFN=OUTPUT" is assumed. CS612 and CS64 indicate the character set of the output file. CS612 indicates the NOS/170 6/12 ASCII character set and CS64 indicates NOS/170 display code. FW is the first NOS/170 word of the file to dump and LW is the last word of the file to dump.

The output of DUMP170 is printable and is in the following format:

```
1 WORD     HEXADECIMAL DATA        ASCII DATA
  ADDRESS 0   1   2   3   4       0 1 2 3 4

        0  000 000 000 000 002  *
        1  041 04D 044 055 046    A M D U F
        2  044 049 020 020 020    D I
        3  800 000 080 000 002  *
        4  043 04F 04D 04D 04F    C O M M O
        5  04E 020 020 020 020    N
        6  800 000 080 000 00E  *
        7  07B 020 054 068 069    {   T h i
        8  073 020 063 06F 06D    s   c o m
        9  06D 06F 06E 020 064    m o n   d
```

Note: A word with ascii data all blanks and a "*" just to the left of the ascii data area indicates a word containing a record header.

2.11 ERROR CODES

The following list includes only those error codes returned by the SIMULATED NOS/VE I/O system. A complete list of all error codes generated by the SIMULATOR can be found in the ERS for the SIMULATOR. Within each message, the character string "+P" (plus, letter P) is replaced by a parameter that is generated by the SIMULATOR.

NUMBER	MESSAGE / SIGNIFICANCE
6350	"ILLEGAL AMP$FILE REQUEST, +P, LFN=+P" The caller has attempted an AMP$FILE function when the file is open. The file must be closed before the AMP$FILE call is accepted.
6355	"+"+P", MEMORY ERROR, STATUS=+P" An error was detected attempting to reference virtual memory while processing a NOS/VE I/O request. The most likely cause of the error is an invalid parameter passed with the I/O call by the program.
6360	"+CANT ALLOCATE SPACE FOR +P" There is insufficient space to run the program. Consult those who maintain the SIMULATOR and have the SIMULATOR rebuilt with a larger stack/heap or reduce the amount of space used by the program.
6365	"+PARAMETER ERROR FOR "+P", LFN=+P" The value of the named parameter is out of range or the parameter specifies an ordinal not supported by the SIMULATED NOS/VE I/O system.
6370	"+ATTEMPTED SECOND OPEN OF LFN=+P" An open was requested of a file which was previously opened and has not been closed. The program may contain an error or one of the load memory commands (such as Load_virtual_Environment) has been used more than once to reload a program for execution. For the latter case the Close_Logical_File command may be used to logically close the file.
6375	"+DIRECT ACCESS ATTEMPTED OF SEQUENTIAL FILE, LFN=+P" A "get_direct" or "put_direct" was attempted on a file opened with a file_organization of "amc$sequential".
6380	"+GET BEYOND END-OF-INFORMATION, LFN=+P" A sequential get has been attempted after end of information has been returned for the file or the byte address specified for a get_direct is beyond

the last record on the file.

6385 "+NO RECORD AT BYTE-ADDRESS OF FILE, LFN=+P"
This error can occur for any get request or a put_direct with a byte_address that is before end of information. This error occurs only for files with record type amc$variable. The data on the file at the current byte address cannot be recognized as a valid record header. The current byte address does not point to a record. The program making the request must be in error.

6390 "+ILLEGAL AMP$GET_ REQUEST, LFN=+P" !
The caller has made an AMPGET_DIRECT, AMPGET_NExT, or AMP$GET_PARTIAL call when the access_mode of the file does not include the set elements PFC$READ or PFC$MODIFY (i.e. the file cannot be read). !

6395 "+ILLEGAL AMP$PUT_ REQUEST, LFN=+P" !
The caller has made an AMPPUT_DIRECT, AMPPUT_NEXT, or AMP$PUT_PARTIAL call when the access_mode of the file does not include the set elements PFC$SHORTEN, PFC$APPEND, or PFC$MODIFY (i.e. the file can not be written). !

2.12 USING THE CYBER 180 SIMULATOR

Programs compiled with the CYBIL (CI) compiler may be prepared for SIMULATION via the GENCPF procedure. The output file from GENCPF is a CheckPoint File. When the SIMULATOR is called into execution, the program may be loaded into the SIMULATOR by specifying the file as the ReStart file to the SIM180 procedure or after the SIMULATOR has been called the file may be loaded by the restart (RS) command.

In the event that an error is found in the program, the SIMULATOR stops running, however all files previously opened via AMP$OPEN procedure calls in the program are still logically open. In addition, the P-register is advanced to the next instruction beyond the AMP$ procedure call. A run command entered at this point will resume execution of the program at this point.

It also possible to reload the program again and restart execution from the beginning of the program after an error stop has been made. For this case the SIMULATOR does not automatically perform the logical close of any files opened in the previous run. The user is responsible for maintaining the logical file status correlation between the SIMULATOR and the program. The Close_Logical_Files (CLF) command may be used at any time to logically close (to the SIMULATOR) any file opened by the program.

When a program built via "SES.GENCPF" runs to completion on the SIMULATOR the message "NORMAL termination" is output to the display. This message is built into the program by GENCPF. Following this message is a SIMULATOR message of the form "** MSG 6218: CPU HALT AT ...", after which the SIMULATOR accepts further commands. To log off the SIMULATOR use the "END" command.

2.13 SAMPLE TERMINAL SESSION FOR SIMULATED NOS/VE I/O

```
/get,ampex
/ses.gencomp m=ampex b=base cybicmn
*    GENERATING COMPILE FILE  COMPILE
*    END GENCOMP    COMPILE <- BASE
/ses.cybil ci
*    COMPILING COMPILE
*    END CYBIL   COMPILE -> LISTING, LGO
/define,xampex
/ses.gencpf ofl=lgo cybilib cpf=xampex
*.EXECUTING THE VE LINKER.
*.EXECUTING THE CPF GENERATOR
*    END GENCPF
/ses.print f=(linkmap,listing) h=ampex
*    END PRINT    LINKMAP..LISTING
/attach,data=zsmplsd
/ses.to180 data source
*    END TO180    DATA -> SOURCE
/ses.sim180 xampex
  ** MSG 6052: CYBER 180 SIMULATOR VER 6.0 (REV. Q) LEV 110
? r
  program start AMPEX
  end program AMPEX
  NORMAL termination
  ** MSG 6218: CPU HALT AT P=0000 B000 0000 0018, STATE=JOB
? end
*    END SIM180
/ses.from180 fmt180 list
*    END FROM180    FMT180 -> LIST
```

2.14 NOS/170 INTERFACE DIFFERENCES

Differences between SIMULATED NOS/VE I/O and the NOS/170 INTERFACE have been minimized, but cannot be completely eliminated. These differences arise from the hardware differences between the two machines and from the intentionally designed differences between the two operating systems. The differences are:

1. Working_storage_length parameter is always given in CELLs and should be generated via the #SIZE function of CYBIL. Difficulties may arise when communicating with an interactive file because this device type requires a length in characters. NOS/VE has no problem here because CELL size equals character size. NOS/170 packs 5 characters per CELL (word). Unless the user's messages are multiples of 5 characters there is a possibility that 1 to 4 garbage characters may be transmitted to the terminal. The NOS/170 INTERFACE will truncate all "non_graphic" characters in the last CELL of data. The NOS/170 INTERFACE also blank fills any remaining characters in the last CELL of any input data. Normally all CYBIL variables start on a CELL boundary, but an exception is strings that are part of a PACKED RECORD. If the message does not start on a CELL boundary 1 to 4 garbage characters may precede the message transmitted to the terminal. Input from a terminal to a variable not starting on a CELL boundary may result in the loss of the first 1 to 4 characters of data.

2. Since both NOS/VE and SIMULATED NOS/VE I/O simulate the same procedures, both would ordinarily have the same entry_point names. In order to avoid this problem the SIMULATED NOS/VE I/O procedures have entry_point names beginning with "AMP#", whereas NOS/VE will keep the entry_point names beginning with "AMP$". The "AMP#" entry_point names will come from the copy of the common decks for NOS/VE kept on the CYBICMN library. Should one want the NOS/VE common decks instead, they may be obtained by specifying and alternate base with these common decks to SES.GENCOMP.

2.15 SAMPLE SESSION FOR THE NOS/170 INTERFACE

```
/get,source
/ses.to170 source
*    END TO170    SOURCE
/ses.gencomp m=ampex b=amp cybccmn
*    GENERATING COMPILE FILE  COMPILE
*    END GENCOMP    COMPILE <- AMP
/ses.cybil debug
*    COMPILING  COMPILE
*    END CYBIL    COMPILE -> LISTING, LGO
/ses.link170 cybclib debug
*    END LINK170    LGOB,ZZZZZDT
/debug,on
$DEBUG,ON.
/lgob
 CYBIL INTERACTIVE DEBUG
? sb 1114;go
program start AMPEX
 *B #1, AT  LINE=1114
? dv tr_count
 TR_COUNT = 2
? cb b=1;go
end program AMPEX
 *T #17, END IN  MODULE SW=CMFR OFFSET 0(8)
? quit
 DEBUG TERMINATED
/debug,off
$DEBUG,OFF.
/ses.from170 fmt180 file
*    END FROM170    FMT180 -> FILE
```

3.0 SIMULATED NOS/VE PROGRAM MANAGEMENT INTERFACES

3.1 PMXGTIM (PMP$GET_TIME)

```
{
{   The purpose of this request is to obtain the current time of day
{  in a user selected format.
{
{        PMP$GET_TIME (FORMAT, TIME, STATUS)
{
{  FORMAT: (input) This parameter specifies the format in which the time
{        will be returned.  Valid specifications are:
{          pmc$ampm_time :  HH:MM AM or PM
{            example:  1:15 PM
{          pmc$hms_time : HH:MM:SS
{            example: 13:15:21
{          pmc$millisecond_time : HH:MM:SS:MMM
{            example: 13:15:21:453
{
{  TIME: (output) This parameter specifies the current time.
{
{  STATUS: (output) This parameter specifies the request status.
{

  PROCEDURE [XREF] pmp$get_time (format: ost$time_formats;
    VAR time: ost$time;
    VAR status: ost$status);
```

3.2 PMXGDAT (PMP$GET_DATE)

```
{
{   The purpose of this request is to obtain the current date in}
{  a user selected format.}
{
{        PMP$GET_DATE (FORMAT, DATE, STATUS)
{
{  FORMAT: (input) This parameter specifies the format in which the date
{        will be returned.  Valid specifications are:
{          pmc$month_date : month DD, YYYY
{             example: November 13, 1978
{           pmc$mdy_date : MM/DD/YY
{             example: 11/13/78
{           pmc$iso_date : YYYY-MM-DD
{             example: 1978-13-11
{           pmc$ordinal_date : YYYYDDD
{             example: 1978317
{
{  DATE: (output) This parameter specifies the current date.
{
{  STATUS: (output) This parameter specifies the request status.
{

  PROCEDURE [XREF] pmp$get_date (format: ost$date_formats;
    VAR date: ost$date;
    VAR status: ost$status);
```

3.3 PMXLOG (PMP$LOG)

This interface differs from the description in the NOS/VE ERS, in that TEXT issued as a message to the CYBER 170 dayfile, bracketed by the messages "*** MESSAGE ***" and "*** HSS MESSAGE ENDS ***". The parameter LOG is not used, and included solely for the sake of compatibility with NOS/VE.

```
{
{   The purpose of this request is to place a message onto the dayfile
{
{       PMP$LOG ( TEXT, LOG, STATUS )
{
{ TEXT: (input) This parameter specifies the text of the message.
{
{ LOG: (input) This parameter specifies which log(s) is to receive
{       the message.
{
{ STATUS: (output) This parameter specifies the request status.
{

PROCEDURE [XREF] pmp$log (text: pmt$log_msg_text;
  log: pmt$logset;
  VAR status: ost$status);
```

3.4 PMXGMSC (PMP$GET_MICROSECOND_CLOCK)

This routine functions differently depending on whether CPU_TIMING is turne or off in the CYBER 180 SIMULATOR. If CPU_TIMING is turned on, then the vari MICROSECOND_CLOCK returns an accurate value of the time since the star

execution. If CPU_TIMING is turned off, then MICROSECOND_CLOCK contains a coun
the instructions executed since the beginning of execution.

```
{
{    The purpose of this request is to obtain the current value of the
{  microsecond clock.  Successive requests are guaranteed to return
{  different values.
{
{        PMP$GET_MICROSECOND_CLOCK (MICROSECOND_CLOCK, STATUS)
{
{  MICROSECOND_CLOCK: (output) This parameter specifies the current value
{        of the microsecond clock.
{
{  STATUS: (output) This parameter specifies the request status.
{

PROCEDURE [XREF] pmp$get_microsecond_clock (VAR microsecond_clock:
  ost$microsecond_clock;
  VAR status: ost$status);

TYPE
  ost$microsecond_clock = 0 .. 0ffffffffffff(16);
```

3.5 DATA TYPES

3.5.1 PMDLOGP

```
{ System logging interface  type declarations. }

TYPE
  pmt$log_msg_text = string ( * ),

  pmt$log_msg_origin = (pmc$msg_origin_command, pmc$msg_origin_system,
    pmc$msg_origin_procedure, pmc$msg_origin_program),

  pmt$logs = (pmc$job_statistic_log, pmc$account_log, pmc$engineering_log,
    pmc$statistic_log, pmc$system_log, pmc$job_log),

  pmt$ascii_logs = pmc$system_log .. pmc$job_log,

  pmt$binary_logs = pmc$job_statistic_log .. pmc$statistic_log,

  pmt$global_logs = pmc$account_log .. pmc$system_log,
  pmt$logset = set OF pmt$logs,

  pmt$ascii_logset = set OF pmt$ascii_logs,

  pmt$binary_logset = set OF pmt$binary_logs,

  pmt$global_logset = set OF pmt$global_logs;
```

3.5.2 OSDTIME

```
{ Time request return value. }

TYPE
  ost$time = record
    CASE time_format: ost$time_formats OF
    =osc$ampm_time=
      ampm: string (8), { HH:MM: AM or PM }
    =osc$hms_time=
      hms: string (8), { HH:MM:SS }
    =osc$millisecond_time=
      millisecond: ost$time_ms,
    CASEND,
  recend,

  ost$time_ms = string (12), { HH:MM:SS.MMM }

  ost$time_formats = (osc$default_time, osc$ampm_time, osc$hms_time,
    osc$millisecond_time);
```

3.5.3 OSDDATE

```
{ Date request return value. }

TYPE
  ost$date = record
    CASE date_format: ost$date_formats OF
    =osc$month_date=
      month: string (18), { month DD, YYYY }
    =osc$mdy_date=
      mdy: string (8), { MM/DD/YY }
    =osc$iso_date=
      iso: string (10), { YYYY-MM-DD }
    =osc$ordinal_date=
      ordinal: string (7), { YYYYDDD }
    CASEND,
  recend,

  ost$date_formats = (osc$default_date, osc$month_date, osc$mdy_date,
    osc$iso_date, osc$ordinal_date);
```

3.5.4 OSDSTAT

```
CONST
  osc$max_condition = 999999,
  osc$status_parameter_delimiter = '"';

TYPE
  ost$status_condition = 0 .. osc$max_condition;

TYPE
  ost$status = record
    case normal: boolean of
    =false=
      identifier: string (2),
      condition: ost$status_condition,
      text: ost$string,
    casend,
  recend;

*callc osdstr
```

Table of Contents