COMPUTER VISION, GRAPHICS, AND IMAGE PROCESSING **23**, 42–66 (1983)

Over the past several years, the University of Maryland and Virginia Polytechnic Institute and State University have been collaborating on the development of a transportable image processing software system, written in RATFOR (RATional FORtran). This paper documents the "kernel" of the system, which interfaces to the given operating system and provides standardized operating system services to the image processing programs. The programs themselves will be documented in a subsequent paper.

Editor

An Operating System Interface for Transportable Image Processing Software

SCOTT KRUSEMARK AND R. M. HARALICK

Spatial Data Analysis Lab, Virginia Polytechnic Institute and State University, Blacksburg, Virginia 24061

Received March 17, 1982

The key to portability is the use of a kernel of routines that interface to the operating system of an individual machine. The kernel provides the sophisticated but standard operating system services required by image processing software. It makes the operating system of each computer appear identical and, when carefully designed, does not pose a difficult implementation problem. Above this interface, all applications programs can be machine independent, written in a structured language, such as RATFOR, without sacrificing power or ease of use on any machine. The details of such an interface design are given. This interface, called the kernel, has been implemented on the IBM 370/VM and the VAX 11/780.

1. INTRODUCTION

1.1 General Comments

In any large portable, structured software package, the applications programming code must be isolated from any code performing local operating system services. Defining the boundary between the operating system services and user code permits all operating system services to be supplied through a standard interface. It is the purpose of this paper to define such an interface for image processing software.

The operating system service routines (hereafter referred to as OS routines) are defined as a standard machine independent interface that the user can access. These routines bring the host operating system out to a standard interface. The OS routines are defined simply enough that they are straightforward to implement. They perform many services that are usually set up by the FORTRAN compiler, but tend to be handled differently by different compilers. Many of the OS routines are concerned with input/output. It is assumed that the image processing software is written in RATFOR.

1.2 Internal Structure

The OS routines are actually made up of two internal levels of code. The outer layer is used mostly for error checking and the manipulation of the file descriptors. This level is mostly machine independent. There will be occasions when these

routines will have to be modified from installation to installation but it is hoped that these will be rare and minor. This layer of code is what the user sees; it is defined as the installation interface.

The inner layer is the code that is truly machine-dependent. These routines are called with as many arguments as could possibly be needed. The particular implementation then uses only those that are needed. For example, the IBM 370 does not use a logical unit number for file access, but it does use the file name. So on the IBM, the logical unit number is ignored and only the file name is used. The PDP-15, on the other hand, uses a logical unit number once the file is open. So on the PDP-15, the open uses the logical unit number and filename and the read/write uses only the logical unit number. This level of code may be highly variable in function and is invisible to the user. The discussion here about the OS routines refers only to the outer layer or installation interface part of the OS routines.

1.3 Errors and Status Return

Each OS routine is an INTEGER FUNCTION subprogram whose returned values is an integer event variable (IEV) indicating success or failure. The value for success is zero. This indicates that everything went as expected. A negative value stands for an index to an error condition. If the value is negative, then the calling routine will want to exit, passing the error status all the way back to the command string interpreter level which can then communicate an error message to the user. Positive values do not represent errors. They are used to indicate nonerror status conditions.

The OS function calls are abnormal in the sense that they may redefine or change the values for some of the arguments. In other words, they allow information to be passed back through the argument list. An example is OSRDS which passes the record length back to the calling program.

1.4 File Descriptor

All the input/output related OS routines are passed a file descriptor (FD). The file descriptor is an array containing, in a system dependent format, all the information that is needed for communication between I/O routines. For example, the open and read routines need to communicate such information as the file name and logical unit number between themselves. The file descriptor holds this information.

Information held in the file descriptor must not be directly accessed by the user because its content and format may change from one installation to another. Access routines (OSINFD, OSGTNM, OSGLUN, etc.) are provided for retrieving the information from the file descriptor. In typical use, OSINFD is passed an uninitialized file descriptor and a user supplied free format file name (or device description). The OSINFD parses the file name and puts it into the file descriptor and initializes the file descriptor to a known state. The file descriptor is then passed to one of the file open routines (OSOPNR or OSOPNS) which then actually opens the file. Then the file is read, using OSRDS or OSRDR, one record at a time. Finally the file is closed, using the OSCLOS routine. The file descriptor, as it is passed in argument lists, gets updated with the current status of the file. The free format file name does not have to be kept separately since the routine OSGTNM can be used to retrieve the file name from the file descriptor whenever necessary.

Since the file descriptor is installation dependent, the local file name format can be used for image files or any other file one wishes the system to handle with no trouble. The file name conventions are set up to look like the ones the system uses. This reduces learning time for a new system.

The OS file passwords are allowed and are only designed to protect a user from unintentionally destroying something he might want to save. The passwords have a place in the file descriptor (put there by OSPINF) and are eventually stored in the file itself. The amount of safety added by the password facility is very limited since standard access routines such as editors and user written code can get at them very easily.

2. THE OS ROUTINES

In this section, software specifications for the OS routines are given. To make the description easier to follow, symbolic names will be used for certain numeric values. These symbolic names are given in two forms. The first is an upper case string which is, in all cases (except where noted), a variable in the argument list of a routine or the routine name itself. The other use of symbolic names is for token replacement definitions. These all begin with a period (.) to distinguish them. They are mnemonics for the programmer and are used here for the same reason. An example is the use of .OLD instead of 1 to indicate that an old file is to be opened. The RATFOR preprocessor translates .OLD into a 1 so that the output is FORTRAN compatible. (This is a use of the token DEFINE statement in RATFOR.) This token DEFINE can also be used to implement a CHARACTER statement in user code even if the particular FORTRAN does not have it.

The sections are organized as follows. The file descriptor manipulation routines are first described. Then the random and sequential I/O are described. Next, miscellaneous I/O involving parameter transfer and the renaming of a file are covered. Then, general features such as IEV manipulation and character manipulation are covered, and finally, program control is described. Each routine has a letter or several letters after the call arguments. These indicate what category of transportability the routines are in.

1. SNT—Nontransportable, but simple changes are required for the code of one machine to work on another machine.
2. TV—Transportable versions exist.
3. NT—Non-transportable and not easy to implement (usually).
4. NT-E—Non-transportable, but are extensions and thus are not really necessary.
5. MSWV—Transportable versions exist that work on MOST systems.
6. T—Fully transportable.

2.1 File Descriptor Manipulation

2.1.1 Initialize File Descriptor

IEV = OSINFD(FF, FD) (SNT)

This routine takes a free format one-dimensional array (FF) of the local file name or null string (defaults to device terminal) and formats it into the system defined format file descriptor (FD). The free format one-dimensional array is not fixed or limited in length but must be terminated with an .EOS character. The routine OSINFD parses the name and puts it into an internal standard format for faster processing by the other routines. It also initializes the other constants in the file descriptor. This routine only initializes the file descriptor with the defaults (except the file name). The other routines, OSPINF, OSCASE, etc., put the information needed to utilize the file descriptor into the file descriptor. By the defaults chosen, the need to call these other routines is minimal for sequential file I/O of variable length records, or of terminal or printer, etc., devices that have clearly defined characteristics. Since the terminal is the most common device, if the null string is passed to OSINFD (a string with the character .EOS alone), the resulting file descriptor is interpreted to be a terminal. For other devices, such as a printer, one should use OSPINF and indicate the correct device. Anything other than the null string is interpreted as a file name, although some systems may override this.

2.1.2 Get File Name from File Descriptor

IEV = OSGTNM(FD, FF) (T)

This routine performs the reverse of putting a file name into a file descriptor. The OSGTNM returns a free format file name/device into an array of length .FILENAMELENGTH, which can be used for direct output, etc. The characters in FF are the standard A1 format characters for both OSGTNM and OSINFD. Note that to be put back into OSINFD, an .EOS character must be added as the last character, which is reserved for this use. This routine and OSINFD should move an illegal file name (if such is possible on the particular machine) in such a way that it can be retrieved. An error value should be generated for OSGTNM, not by OSINFD where the name is illegal.

2.1.3 Get Information from File Descriptor

IEV = OSGINF(FD, OPT, VAL) (TV)

This routine gets information out of the file descriptor. The values returned by VAL are described with token replacements where applicable. The OPT argument is shown in parentheses at the end. The information available is also shown in Table 1. This available information is

1. Device in file descriptor. (.DEVICE)

2. TYPE file open: .OUTPUT, .NEW, .OLD, .NOTOPEN, and .INPUT are returned in VAL. (.FILETYPE)

3. Whether file has a .FIXED or .VARIABLE format. (.FIXVAR)

4. Number of records if file is a random file. (.NREC)

5. Length of each record if .FIXED, or the maximum record length if .VARIABLE format. (.LREC)

6. Mode of file: .INTMODE, .REALMODE, etc. (.MODE)

7. Logical unit number for a terminal/printer write. (.LUN)

8. Case indicator for the file which tells whether the file is forced .UPPER case or left .MIXED case. (.CASE)

TABLE 1
OSGINF Options and Returned Values

Meaning	Option	Possible values
Device in FD	.DEVICE	.DEVDISK, .DEVTERM, .DEVDISPLAY, .DEVTAPE, .DEVPRINT,...
Type file open	.FILETYPE	.OUTPUT, .INPUT, .NEW, .OLD, NOTOPEN
Fixed or variable	.FIXVAR	.FIXED, .VARIABLE
Number of records	.NREC	Number of records
Length of records	.LREC	Length of records
Mode of file	.MODE	.INTMODE, .REALMODE, .CHARMODE,...
Logical unit	.LUN	FORTRAN logical unit number
Force case	.CASE	.UPPER, .MIXED
Printer forms control	.FORMS	.OVERPRINT, .TOPPAGE, .NORMAL
Terminal Echo	.ECHO	.ECHO, .NOECHO
File unique number	.UNIQUE	A unique number for each file descriptor

9. Forms in which control values are available for normal, overprint, and top of page: .NORMAL, .OVERPRINT, and .TOPOFPAGE control constants. (.FORMS)

10. On echoable input, .ECHO and .NOECHO can be obtained. (.ECHO)

11. Because one file descriptor is hard to distinguish from another, by the use of the unique number obtainable from the file descriptor, two different file descriptors can be easily distinguished. This number is unique for each call to one of the open routines (OSOPNR or OSOPNS). (.UNIQUE)

The mnemonics and actual values are given in the Appendix.

The LUN is used for terminal/printer only. Such a logical unit number may be used in formated FORTRAN writes to terminal/printer.

Note that only output devices (i.e., not including files) are legal to write to with a FORTRAN WRITE statement.

The definition of the OS routines has been extended to allow FORTRAN sequential writes to output-only devices. (Terminals are considered to be two separate devices: one input and one output.) The FORTRAN writes must be used with care as they may interfere with some operating system memory allocation methods. The writes are allowed mostly as debugging tools and for outputting to the terminal.

2.2 Put Information into File Descriptor

IEV = OSPINF(FD, OPT, VAL) (NT)

This routine is used to set up or change values in the file descriptor. The values to be set (VAL) are described with token replacements where applicable. The OPT argument is the item to change and is shown in parentheses at the end of each of the following descriptions.

1. Device if the original string to OSINFD was the null string. If not set and was the null string, the device defaults to the terminal. (.DEVICE)

2. Number of records if file is to be a random file. (Must be a file name in FD.) (.NREC)

3. Mode of the file: .INTMODE, .REALMODE, etc. This defaults to .INTMODE for files and .CHARMODE for all devices. This should be set after .DEVICE if FF string to OSINFD is null. Good programming always sets this to a known value rather than relying on the default, however. (.MODE)

4. The password on a password-protected file must be set. It is a positive integer. To default is to not have a password. (.PASSWORD)

5. Logical record length. This must be set on random files before a .NEW open type in units of the .MODE of the file. For sequential files, this only has meaning for files that are of the .SYSMODE data type mode. (.LREC)

The interaction of the .MODE and .LREC and the actual record length (read and write) is as follows: the length of the record depends on the length and the mode. For example, if the mode is .INTMODE, then the length of each element is 1, so a length of 10 is 10 words long. If using the IBM 370/VM or VAX 11/780 in the mode double precision (.DBLPRECMODE), the length of each element is two words and the record of 10 takes 20 words. On the PDP-15 it takes 30 words, since each double precision element takes three words. Thus the actual length of a record in words is dependent on the MODE and LREC.

If the mode is .CHARMODE, then each element is in an A1 format in a CHARACTER variable. This is what happens to a single character when it is read in an A1 format read. The storage format is one character and blanks to fill to one word (either left or right justified). This should be the same format as a CHARACTER variable in a data statement

```
CHARACTER VAR
DATA VAR/'A'/
```

where CHARACTER is a RATFOR token that may, possibly, redefine it.

The other modes are .INTMODE (INTEGER), .REALMODE (REAL), .HINTMODE (HALF INTEGER), .DINTMODE (DOUBLE INTEGER), .DBLPRECMODE (DOUBLE PRECISION REAL), .PCHARACTER (PACKED CHARACTER), and .SYSMODE (SYSTEM FORMAT).

Of the two kinds of files, one is what could be called a KERNEL file. This is a file that the kernel implements and it has any information in it necessary for fast operation and ease of use. (A package of image processing programs that make use of the kernel will be described in a subsequent paper.) The other kind of file is one that the local system maintains. It includes files that come from the editor, the FORTRAN compiler, or whatever other files may exist. The two kinds of files are not usually interchangeable.

The local system files on many computer systems can be particularly difficult, so a mode called .SYSMODE is available. This mode allows files in local system format to be read (written) into (from) a program. Once in a program, they can be written onto (read in from) a kernel file. The modes available to a kernel file are all of the above (except .SYSMODE).

In a typical implementation the kernel file is simply a local system file with an extra record at the beginning. This extra record has information such as the password, whether the file is random or sequential, and, if random, the number of records and the record lengths. This information can be maintained easily by the OS routines so reliance on the operating system is minimal, thus improving transportability of the OS routines.

Terminals have a special need: two kinds of I/O. Logical I/O is the standard I/O where any special characters, such as character delete, are processed, and where carriage control is handled by the system. The reads and writes are line oriented. The second kind is physical I/O, which is usually character by character. A read is a character count read (read until one gets *n* characters). Character deletes and carriage returns are just another character. The kernel implements this by assigning .CHARMODE and .PCHARACTER to logical I/O and the other mode to physical I/O.

The primary difference between physical I/O and logical I/O is that physical I/O is data byte oriented (no parity checking or changing), and logical I/O is character line oriented (entire lines of data).

Physical output does not set parity on the byte to be transmitted. The byte is 8 bits (256 possible values). The routine OSWTS sends a stream of characters. There should be no automatic carriage return/line feed at any time. This operation is used to put out a line to a terminal.

Logical output, on the other hand, is line oriented. The last character in the call to OSWTS determines the kind of action. If it is a .ALTEOL character, then nothing is added and this character is not transmitted. The forms control (line feed, etc.) on the next I/O operation to this device is suppressed. If the last character is not .ALTEOL, then a carriage return/line feed is added following the characters in the buffer.

Note that what is described above is the FORTRAN convention for carriage control. The carriage return is the last character on a write. The carriage control character is sent on the next I/O operation, either a read or a write. For single spacing this is a line feed character. On a .ALTEOL, both the carriage return and the carriage control character (for the next I/O operation) are suppressed.

Physical input also does not change parity. Physical input can be echo suppressed or not (this is the only mode for which this is true). The OSRDS reads a stream of characters (bytes) from the input buffer of the system exactly as it was transmitted from the terminal. Any user type ahead can be purged with OSFLSH.

Logical input must be terminated by a carriage return sent from the terminal. Logical input is line oriented and a line on the terminal terminates with a carriage return. Case folding (lower case converted to upper case) may be accomplished by the OSCASE parameter CASE.

Note that logical I/O and physical I/O can be interspersed. These two types of I/O may be used in any order and the system must be able to keep the order correct. However, the process of switching from one mode to another purges the input buffer of any type-ahead characters. The routine OSPINF sets the mode. All output must be output before any read is satisfied so that (1) the echo is in correct place, and (2) the user sees the whole output line even if he has already responded to it with his answer in the correct place.

2.2.1 Generate Unique File Name

IEV = OSGFNM(FF1, FF2) (SNT)

This routine takes a six character alphabetic string and creates a unique (repeatable) file name from it. The name should be such that the user can more or less read the name back from the file name generated. This routine is a method for (1) generating temporary file names, and (2) getting system dependent names of fixed data files into the system. The array FF2 is .FILENAMELENGTH long and is acceptable to OSINFD. The array FF1 must be left justified if any character is a blank. A blank is assumed to be a terminator (i.e., end of string). Both FF1 and FF2 are A1 format character strings.

To generate the temporary file name, for example, this routine is first called with some string of characters, the name is made into a file descriptor (OSINFD), and then the existence of that file is checked (OSSTAT); if OSSTAT indicates that the file name is not there, then a temporary file name has been created. If OSSTAT indicates that it does exist, then a new string is passed to OSGFNM and the process starts over.

2.2.2 DEBUG: Display Contents of File Descriptor

CAL OSNAME(FD) (SNT)

This routine dumps out the file descriptor to the terminal. Since the file descriptor is system dependent and the data is usually not stored in character format, this routine prints it out in readable format. This routine outputs all the information in the file descriptor, including the system dependent portion. Note that user modification or access into the file descriptor is a violation of the concept behind the OS routines and is an illegal programming practice.

2.3 Random File I/O

2.3.1 Random File Open

IEV = OSOPNR(FD, TYPE) (NT)

This routine performs one of two types of random file open for a file with file descriptor FD. The TYPE argument can take the two values of .OLD and .NEW. If TYPE is

1. .OLD: OSOPNR goes out and checks if a file exists by the correct name; if it does, OSOPNR opens it.

2. .NEW: OSOPNR opens an output file for data to be written to. On the call to close, this file is closed and it replaces any file existing with the same name.

Number of records, fixed file format, and logical record length must have been previously specified for .NEW files (see OSPINF). For .OLD files, these values are returned (see OSGINF); however, the mode of the file must have been set before the call to OSOPNR so that the sizes can be correctly computed (see OSPINF).

2.3.2 Random File Read

IEV = OSRDR(FD, REC, BUF, COUNT, WAIT) (NT)

This routine does the random read. The record number to be read is REC which is checked to determine if it is positive and less than or equal to the NREC argument in OSPINF. The COUNT is the number of elements to be read into buffer BUF. If COUNT is longer than one record, it will read multiple records. (If any of these records are outside the file, an error results.) If COUNT Is less than a full record, only COUNT elements are returned. The rest are thrown away (see MODE in OSPINF for the length of each element). A check is made to determine if this file descriptor has been opened as a random file.

The parameter WAIT is used to enable the use of double buffering if the facility is available and if the programmer wants to go to the trouble. Normally, this parameter is set to a value of .WAIT but can be set to .NOWAIT for immediate return to user code after starting the read request (see OSWAIT for more).

2.3.3 Random File Write

IEV = OSWTR(FD, REC, BUF, COUNT, WAIT) (NT)

This routine does the random write. The record to be written is REC. The WAIT is the same as above. If COUNT is greater than one record in length (see OSPINF for record length), multiple records are written. If COUNT is not a multiple of the record length, the partial record contents are unpredictable.

2.3.4 Random File Multi-Buffering Wait

IEV = OSWAIT(FD, WAIT) (TV)

This routine checks to see if the input/output is completed for the file with file descriptor FD. If the argument WAIT is .WAIT, then the routine returns, after I/O is completed, from the file with file descriptor FD. If the argument WAIT is .NOWAIT, then the routine checks for I/O completed from the file for which file descriptor is FD. It returns immediately even if I/O is not completed. The IEV values can be .OK when I/O is finished, .NOTDONEIEV when I/O has not yet finished, or any other value which indicates the appropriate error.

This routine with the .NOWAIT option on the read/write can be used to advantage for overlapping I/O and computing. The programmer, however, is responsible for checking that the I/O has in fact been completed before attempting to use the data (read) or reuse the space (write).

2.3.5 Random File Growing

IEV = OSGROW(FD, NREC) (TV)

On some systems, random files are stored completely differently than sequential files in that the random files must be opened to a fixed size and never altered. This creates a problem if the user wants to make a file larger. The OSGROW provides a facility to get around this problem. On most systems, all that OSGROW does is increase the last record that can be written by bumping the value in the file descriptor. On some systems, however, the file is copied with the new larger size, so user algorithms that grow a file may result in excessive I/O time.

Note that OSGROW can only change the number of records. To be able to change anything else, such as logical record length, a full copy of the file must be made by the user. The OSGROW can only increase the number of records. If the user wants to shrink a file, it must be made by user copy.

2.3.6 File/Device Close (Random and Sequential)

IEV = OSCLOS(FD, TYPE) (NT)

All file operations and most other device operations need to be closed. There is one close routine to simplify the closing. The file descriptor contains all the needed information.

The arguments are simple: the file descriptor represents the file to close and TYPE indicates what to do with that file. There are only two options, keeping the file or getting rid of it, represented by the symbolic names .KEEP and .DELETE, respectively (possible values for the argument TYPE).

The observable result of opening an output file is that if a another file exists with the same name, it is destroyed at the time of file close. If two files, one .INPUT (.OLD) and the other .OUTPUT (.NEW), are opened at the same time, after both are closed .KEEP, only one file exists; that is, the .OUTPUT (.NEW) one. The order of the close must make no difference. If the .DELETE option is used, the file that is being closed is deleted. This means that a preexisting file will exist after a close .DELETE on an .OUTPUT or .NEW file.

2.4 Logical Sequential I/O

2.4.1 Sequential Open

IEV = OSOPNS(FD, TYPE) (NT)

This routine does the sequential I/O open which is much more complex than random I/O. The operating characteristics of random files are straightforward, but sequential I/O may include sequential files, terminals, printers, card readers, magnetic tape, and other sequential devices.

Random files are stored on the disk in a format that is maintained by the OS routines. The same is true of sequential files. However, to initially get these files in the system the files must have been in a "local operating system" format. This format is a pure data format because there is no extraneous information in them. These are accessible only as .SYSMODE mode files (see OSPINF).

The OSOPNS has two types of opens depending on the argument TYPE.

1. .INPUT—A file already exists that can be read from (input is to program).
2. .OUTPUT—Create a file composed of records to be written to this file.

Note that for sequential files a file opened as an input file cannot be written to and vise versa. If the reverse operation is wanted, the file must be closed and reopened.

All terminals, printers, etc., must also be opened and used correctly. Opening a printer for input is an error.

2.4.2 Logical Sequential Read

IEV = OSRDS(FD, BUF, LEN, BUFSIZ) (NT)

This routine does the sequential read operation. The data goes into the array BUF. The BUF has the length BUFSIZ. Data which takes more than is allocated in BUF produces an IEV error. The actual length of data is returned in LEN.

On a terminal which has no easy way of defining end of file, the END-OF-FILE is simulated by a control D character. It should be the first character in a line.

On random files, a fixed mode for the file is the only practical method, but sequential files use variable size record format so the mode can change with every record. The use of changing modes is allowed by the use of the routine OSPINF but care must be taken that the record sizes are within the limits.

Terminals represent a special case as far as operating systems are concerned. They have two modes: byte and character. Byte mode means 8-bit data, no parity check, and no interpretation of data. Terminals using character mode check parity and certain characters (usually control characters) which are intercepted by the operating system. These characters delete a line, delete a character, and abort a program, among other things.

Terminals using .BYTEMODE (or .INTMODE, .REALMODE, etc.) operate differently than .CHARMODE terminals. In this mode, the normal operating system functions are skipped and the characters in output buffers are output directly, while on input nothing happens to them. The user is responsible for all parity and line control operations such as carriage return/line feed. Input is read BUFSIZ characters, no matter what they are. The user is fully responsible for any side effects such as parity bits. This mode, however, is very useful for graphics operations. The mode is set by OSPINF. Those systems which do not allow this mode of operation should return an error from OSPINF when attempting to set binary modes on a terminal.

2.4.3 Logical Sequential Write

IEV = OSWTS(FD, BUF, LEN) (NT)

This routine does the sequential writes. It is defined for sequential files, terminal, printer, punch, and other output devices. The data is in BUF with length LEN.

This routine has two options for terminal character writes. One is where BUF(LEN) is .ALTEOL; this results in the suppression of the carriage return/line feed normally added. Trailing spaces are not suppressed. The other option is where BUF(LEN) is not .ALTEOL, in which case a carriage return/line feed is added. Carriage control is controlled by OSFORM which sets up forms control. Only terminals and printers are affected, though. Thus the first character in BUF is actual data rather than carriage control as is the practice in FORTRAN.

The modes for binary operations for the terminal act in a manner similar to the terminal binary reads. Parity is not checked or set, and special characters that are normally processed or suppressed are passed through unaltered. This is useful for graphics operations where the eight bits are used as an address on a screen and thus can take any value.

2.4.4 Upper / Lower Case Set/Clear

IEV = OSCASE(FD, CASE) (TV)

This routine changes the flag on case operations of the read or write. The argument CASE can have two values: .UPPER and .MIXED. A value of .UPPER forces all alphabetic characters to be upper case on the read. A value of .MIXED leaves the characters alone. Upper case is very useful for command line input. Mixed case is useful for descriptions that go into files, etc.

2.4.5 Set/Clear Echo/Noecho on Terminal

IEV = OSECHO(FD, ECHO) (NT–E)

This routine allows the user to enable or suppress terminal echo. The argument ECHO can have two values: .ECHO and .NOECHO. A value of .ECHO allows the user to see what was typed. A value of .NOECHO suppresses echo on the terminal. This operation is very useful for graphics terminal operation.

2.4.6 Set Printer Format Specifications

IEV = OSFORM(FD, CC) (TV)

This routine sets the carriage control character for the next write. The argument CC has values of .OVERPRINT, .TOPPAGE, and .NORMAL (for single space operation). This routine has no effect on any device except a printer, but can be used to control output that is also directable to the terminal or files.

2.5 Physical Sequential I/O

2.5.1 Set Parameters For Tape

IEV = OSTAPE(FD, OPER, CNT) (NT–E)

This routine controls the tape operations. The argument OPER is the operation to be performed; .REWIND, .WRITEENDOFFILE, .WRITEENDOFTAPE, .BACKSPACEFILE, etc., are the values. The argument CNT is for operations such as set logical record length where CNT is this number. Tape operations are defined to be such that a call to OSWTS or OSRDS writes or reads a physical block of the tape. The user is responsible for separating the logical records.

2.5.2 Flush Type-Ahead Input

IEV = OSFLSH(FD) (NT–E)

This routine is useful in those cases where a physical read from the terminal is done. In such situations, the information has a specific format so any characters the user might have typed ahead will cause this format to be upset. A call to OSFLSH clears the type-ahead buffers before the physical read is requested. This routine only has an effect on terminal input. A type-ahead buffer is where the user can type ahead characters that are not read until the read is issued. This type-ahead is usually only implemented on FULL-DUPLEX systems and not always then. For example, the IBM 370/VM, when used with ASCII terminals, generates an interrupt if any characters are typed before the read is issued. This is an example of a machine with no type-ahead buffer.

2.6 Miscellaneous I/O

2.6.1 Parameter Transfer

IEV = OSSEND(BUF, LEN) (TV)

IEV = OSRECV(BUF, LEN, BUFSIZ) (TV)

To send and receive data between overlays, two routines are used: OSSEND and OSRECV. The operation to send (OSSEND) may be implemented as many times as is necessary. The send operation is terminated by sending a record of zero length. The receive operation (OSRECV) is terminated upon return of .EOFIEV (end-of-file). The data is in the array BUF which is BUFSIZ INTEGER words long. The number of INTEGER words used is returned in LEN.

The send and receive operation is much like a push-down stack. The last thing sent is the first thing received, while the first thing sent is the last to be received. Thus, the send code for multiple sends will be in reverse order of the receive code.

Send and receive can be implemented using the random file I/O primitives. They are then fully transportable. At installations which have the capability for passing messages, this facility could be used to pass short message data. For large amounts of data, the random file primitives could be used.

2.6.2 Rename a File

IEV = OSRENM(FDI, FDO, TYPE) (TV)

This routine changes the name of a file. Both files must be closed on a rename. The first argument FDI is the file as it exists before renaming, and FDO is the file after renaming. If the argument TYPE is .DELETE and a previous file already exists by the name specified in FDO, then that previously existing file is deleted. If the argument is .KEEP, OSRENM generates an error and no names are changed. If there is no file by the name in FDO, then the value of the argument TYPE is of no consequence.

2.7 General Features

2.7.1 Integer Event Variable Manipulation

IEV = OSSIEV(IEV, MNERR, PMSG) (SNT)

IEV = OSGIEV(IEV) (T)

IEV = OSERRI(IEV) (T)

There is a save area where the integer event variable (IEV) is retained after being set by OSSIEV. The IEV set by the call to OSSIEV is always saved in the save area. The OSGIEV is used to retrieve the value. When an IEV is passed to OSSIEV it can have one of three ranges of values: positive, zero, or negative. Positive and zero values indicate success. Negative values are error conditions. The argument MNERR is used to indicate that this particular error is the main error of the routine. It has two values: .MAINERROR and .OK. Via OSERRI, the error message (PMSG) associated with the call to OSSIEV can be conditionally output to the terminal. The options for the value to be passed to OSERRI that affect the operation of OSSIEV are

1. .OK—output all messages and return IEV value.

2. .SUPPRESS—Suppress all messages but return IEV value.

3. .MAINERROR—Suppress all messages and return IEV as .OK value for all errors except main error to OSSIEV.

4. .SEEOK—Output messages and return IEV as .OK value.

5. *x*—Ignore IEV for value of *x* (where *x* is a negative number) but output all other messages. Return IEV value.

The use of .MAINERROR is as follows: if the .MAINERROR option of OSERRI is chosen, then all messages and IEV are suppressed in a call to OSSIEV except when argument MNERR is the .MAINERROR token. This combination allows creation of test routines that want to ignore certain conditions. Caution should be exercised in the use of this mode. Note that the IEV value is always retrievable from the save area by OSGIEV and is never suppressed. Inside the OS routines each routine has only one main error, usually the most common are, such as end of file on a read.

The IEV value returned by OSERRI is the previous setting of the value being set. This allows levels of code to anticipate an error.

One other feature of OSERRI is its ability to turn routine tracing on and off.

1. .TRACE—Turn tracing on using OSPUSH/OSPOP.

2. .NOTRACE—tracing off.

This routine and OSPUSH, OSPOP, and OSNAME assume that there is a standard terminal output to which error messages, etc., are output.

2.7.2 Routine Name Push and Pop

CALL OSPUSH(PSTRNG) (SNT)

CALL OSPOP (SNT)

This pair of routines can be used inside or outside the kernel. Its action is to show an entry into and exit from a kernel routine. The PSTRNG is a character routine name in quotes with a period terminator. This routine's output is turned on and off by calls to OSERRI. The routine OSPUSH should be the first executable statement in any OS routine and OSPOP should be the last executable statement before the return.

2.7.3 Get System Information

IEV = OSINFO(OPT, VAL) (SNT)

This routine returns information on the system. The following information is available:

1. The system name: six A1 characters. (.SYSNAM)

2. The smallest unit (in bits) in which I/O is measured. The IBM 370/CMS, for example, uses the BYTE as the smallest unit. Many FORTRANs, however, use the word or double word for file storage. (.NUMBITSIO)

3. The smallest number of units in which I/O is done. If the shortest record allowed, for example, is 10 INTEGERs and I/O is presented in INTEGER increments, then 10 is returned. (.NUMMINUNITSIO)

4. The optimum disk block size. (.OPTDISK)

5. The largest possible value that OSALOC will definitely accept. (.MEMMAX)

6. If the argument is a mode such as .INTMODE or .REALMODE, then the VALUE returned is the number of bits for that particular mode.

Only for the machine name is the VALUE an array; for the rest, it is an INTEGER.

2.7.4 Time Measurement

IEV = OSTIME(TIME, FUNC) (NT–E)

This routine measures CPU execution time or wall time. Most systems only keep a rough value of CPU execution time but it is usually fairly accurate for user purposes. To make the operation the same for all systems, the value of time is returned in seconds in a real variable. The value of FUNC determines what to do. The FUNC = 1 returns wall clock time, FUNC = 2 sets the timer to zero. The FUNC = 3 gets the value of CPU time since the last time the timer was set to zero. The timer should always be set to zero before use. Note that except for the data buffers, the value of TIME (in OSTIME and OSDLY) is the only real variable in the OS routines. Attempting to measure small time increments late in the day can result in loss of precision due to the floating point numbers.

2.7.5 Date

IEV = OSDATE(DATE) (NT–E)

This routine returns the current date from the system. The date is a six CHARACTER variable in the form MMDDYY where MM is the month, zero filled (01 to 12), DD is the day of the month, also zero filled (01 to 31), and YY are the last two digits of the year.

2.7.6 Time Delay

IEV = OSDLY(TIME) (NT–E)

This routine causes a delay of TIME seconds. It returns the CPU back to other time sharing users. This function differs from a loop checking wall clock time in that the program does not get charged for execution on those systems that charge for CPU time. (TIME is a REAL variable.)

2.7.7 Bit and Character Manipulation

IEV = OSGBTR(ARRAY, IDATA, SPOS, NBITS, FLAG) (MSWV)

IEV = OSPBTR(IDATA, ARRAY, SPOS, NBITS, FLAG) (MSWV)

IEV = OSUNPK(ARRAY, CDATA, SPOS, LEN, EOS) (MSWV)

IEV = OSPACK(CDATA, ARRAY, SPOS, LEN, EOS) (MSWV)

This set of four routines is for in-core data movement. The routines move bit-strings and characters into and out of arrays. The ARRAY is the place on which addressing is based. The NBITS is the number of bits, including any possible sign bit added for .SIGNED value of flag. The data in IDATA is an INTEGER variable. The routine OSGBTR gets a bit-string from the array and puts it into IDATA. The routine OSPBTR moves a bit-string from IDATA to ARRAY. The bit-string can cross all word boundaries in ARRAY. The maximum number of bits is the number of bits in an INTEGER. The argument FLAG has two values, .UNSIGNED or .SIGNED, indicating whether the sign is to be stored in the number so it can be retrieved. This is done so that different representations of integers (sign magnitude, twos complement, etc.) can be stored with only as many bits as necessary.

The routines OSUNPK and OSPACK get and put CDATA characters from and to ARRAY. The SPOS is the character position. The number of characters to move is LEN. The characters in CDATA are in A1 format. Both OSPACK and OSUNPK should be written such that the same array can be used for both input and output.

The origin of 1 has been chosen for compatibility because FORTRAN accesses arrays as origin 1. In other words, FORTRAN arrays begin at 1, so this routine indexes the same way.

2.7.8 CHARACTER Pack/Unpack for FORTRAN 77

IEV = OSUCHR(PSARRAY, CDATA, SPOS, LEN, EOS) (MSWV)

IEV = OSPCHR(CDATA, PSARRAY, SPOS, LEN, EOS) (MSWV)

The FORTRAN 77 revision of FORTRAN generates a problem due to the inherent conflict between FORTRAN 77 and FORTRAN 66. Array data types such as integers, bytes, etc., are treated differently than FORTRAN 77 CHARACTER**n* data types. (This is not to be confused with the RATFOR string called CHARACTER which is translated to become, in most cases, INTEGER.) These two data types are not interchangeable and the noninterchangeability causes problems for quoted strings in argument lists since they are by definition type FORTRAN CHARACTER.

The two routines OSPCHR and OSUCHR perform the same functions as OSPACK and OSUNPK. The PSARRAY argument is either a quoted string or a quoted string passed from further up. The routine OSPCHR should generally not be used unless the FORTRAN CHARACTER data type is absolutely needed. Any installation using FORTRAN 66 should keep in mind that the code must use the correct routine for quoted strings and array data types to maintain transportability. In FORTRAN 77 installations, this will become obvious because use of the wrong routine will usually generate a memory reference fault.

2.8 Program Control

2.8.1 Allocation and Check of Work Array

IEV = OSALOC(DIM) (TV)

This routine changes the size of the dynamic array WORK that was passed to the user via OSMAIN. This array is intended to be the main user work area. The user requests the number of integer words (DIM) needed and the routine OSALOC checks to see if this is within bounds. If it is, a value of .OK is returned; otherwise, a negative value is returned.

Depending on a site's implementation of OSALOC, this routine may either check this DIM against the size of a statically allocated array or change the bounds of the user memory space. Different operating systems allow different options. This routine should always generate a nonfatal error message using OSSIEV when the array is used but not allocated. The value from OSINFO for .MEMMAX can be used. It is the maximum memory available. This should be used sparingly as it allocates all that is available, and could be very costly.

2.8.2 Program Exchange

IEV = OSCHAN(PROGNM) (NT)

In many applications, there is too much code to fit in memory at one time so some form of overlay must be implemented. The piece of code that is brought in is called an overlay; other terms that can be used are segments, modules, chaining, and links. The routine OSCHAN performs this function. A call to OSCHAN causes execution to be passed to a new overlay. This is a one-way call. The program name (PROGNM) is an A1 format string of characters in the function call. The unpacked string must be six characters long with trailing spaces to pad up to six characters if necessary. Since this routine is never expected to return, any return is an error, indicating such as an overlay not found, regardless of the function value returned. The program name STOP is a special name that returns to the operating system level.

Since this is a one-way call, the user must set up any return needed. The user must also explicitly save any data areas because upon "return" a new copy of the code is overlayed into memory, and all previous data areas are destroyed.

All files and devices must be closed before the call to OSCHAN or unpredictable results may occur.

If a FORTRAN STOP statement is executed, it is interpreted as a return to the operating system (an abort exit). This use is discouraged. The OSCHAN should have at least two places that it searches in the following order:

1. A user's area where each user can test his own new overlays without endangering the system's overlays.

2. A system area which is available for access by all users and can be coded to always go to the same place.

The user overlays should override the system's overlays for ease of testing. More search places, while desirable, are not necessary. This routine must have a complement operation on the operating system that can generate an overlay. This operation should be available to all users to generate an overlay into their user area. To give an example, suppose the system consisted of commands MKCHK, LABEL, COPY, EXPAND, and STOP, and the user wanted to test a new version of LABEL. If there is only one search place, then all users of LABEL would have to wait while it was in the testing phase or the testing phase would have to be done at a time when no one would use LABEL. In any system that is used often, testing is very important and must occur in such a way as to not endanger or disable the current working version. Also, new commands, such as COMPRS, can be developed by several users independently and not generate any conflicts. Obviously only one of them can become the official copy if it proves itself worthy.

3. USER SUPPLIED CODE

3.1 User Entry Point

CALL OSMAIN(WORK) (SNT)

This routine is the user's code entry, i.e., user code is entered by a call to OSMAIN from an OS routine mainline. The argument WORK is the dynamic work array that the user will access. This array (described further under OSALOC) can be dynamically grown or shrunk depending on the needs of the program. This array can be subdivided any way the user needs. If the user ever does a return from OSMAIN, this is taken to be a return to system level.

The routine OSMAIN can be called twice. If the routine OSINTR is called and chooses the .RESTART option, then OSMAIN is called twice. The user should code for this occurrence and take appropriate abnormal action. One suggested method is to use a DATA statement to detect the restart by checking if a particular variable has been set. If it has, then restart has occurred and the routine (OSMAIN) should take appropriate action. If it has not been set, then this is the first time through and the normal action sequences should be executed. The variable should then be set (see OSINTR).

There also exists a routine that calls OSMAIN that sets up interrupt processing, the dynamic work array, and possibly some information on chaining. This routine is the "mainline" of the system and is reused for each overlay and may be written in assembly language.

3.2 User Initiated Interrupts

IEV = OSINTR(INTR) (TV)

This routine implements the function of regaining control after an interactive user interrupt. Thus if a function that takes a long time to execute is started with the wrong parameters, the user can interrupt processing. The OSINTR is called when an interrupt has been encountered. The argument indicates what kind of interrupt has occurred if more than one is possible. A return is always executed in OSINTR. The IEV value is interpreted to indicate what process is to occur:

1. If the IEV value is .OK then execution continues where it was before the interrupt. This is a continue-as-if-nothing-had-happened action; it is also called dismissing the interrupt.

2. If the IEV has the value .RESTART, then the overlay is restarted by the second call to OSMAIN.

3. If the value of IEV is .ABORT, then a return to the operating system is made.

All other values are treated as .ABORT. No operation that could upset any I/O, etc., should be executed. This includes the use of any FORTRAN functions such as ALOG or ** (exponentiation) because these call library routines that may have local storage associated with them. The recommended user routine is to set a flag in a labeled user common and then dismiss the interrupt (.OK).

4. EXAMPLE

```
#  exampl     example of os routine use
#
#
  include macal      # system include file for all tokens
#
  subroutine osmain( work )
#
  character fdi( .FDLENGTH ), fdo( .FDLENGTH )
  integer osinfd, osrecv, ossend, ossiev, osgiev
  integer ossize, osopnr, osaloc, osginf, osnrec
  integer osrdr, oswtr, osclos, oschan
#
  integer rec, nrec, lrec, mod
  character tname( 80 )
  integer work( .ARB )
#
  data rstrt/ .OK /      # initial value of rstrt
#
  if( rstrt~= .OK ) goto 9020      # restart is an error
  rstrt = .OK + 1      # thus~= .OK
  if( .OK~= oserri( .OK )) goto 9020      # set so all
                                          # messages are output
  if( .OK~= osrecv( tname, len, 80 ))
   goto 9000
  if ( len = = 80 ) goto 9010
  tname( len + 1 ) = .EOS
#
  if( .OK~= osinfd( tname, fdi )) goto 9000
#
#
  if( .OK~= osrecv( tname, len, 80 ))
   goto 9000
  if( len = = 80) goto 9010
  tname( len + 1 ) = .EOS
  if( .OK~= osinfd( tname, fdo)) goto 9000
#
#                                 set integer mode for
#                                 input file
#
  if( .OK~= ospinf( fdi, .MODE, .INTMODE ))
   goto 9000
  if( .OK~= osopnr( fdi, .OLD ))
   goto 9000
#
#                                 set up output file
#
  if( ok~= osginf(fdi, .NREC, nrec )) goto 9000
  if( ok˜ = osginf(fdi, .LREC, lrec )) goto 9000
  if( .OK˜ = ospinf( fdo, .MODE, .INTMODE )) goto 9000
  if( .OK˜ = ospinf( fdo, .LREC, lrec )) goto 9000
  if( .OK˜ = ospinf( fdo, .NREC, nrec )) goto 9000
#
  if( .OK˜ = osaloc( lrec )) goto 9000
#
  if( .OK˜ = osopnr( fdo, .NEW )) goto 9000
#
  do rec = 1, nrec
     $(
    if( .OK˜ = osrdr( fdi, rec, work, lrec, .WAIT ))
       goto 9000
#
    do pnt-1, lrec
       work( pnt ) = mod( work(pnt), 10 )
#
    if( .OK˜ = oswtr( fdo, rec, work, .WAIT ))
       goto 9000
    $)
#
#
  if( .OK˜ = osclos( fdi, .DELETE ))
    goto 9000
  if( .OK˜ = osclos( fdo, .KEEP ))
    goto 9000
#
#
  if( .OK˜ = ossend( fdo, .FDLENGTH ))
    goto 9000
  if( .OK˜ = ossend( fdi, .FDLENGTH ))
    goto 9000
  if( .OK˜ = ossend( 0, 0 ))       # close send file
    goto 9000
#
#
  iev = oschan( 'addit' )
#
#
  9000 continue
#
  iev = osgiev( iev )
#
  iev = ossiev( iev, 'error in exampl-see error table.' )
  iev = oschan( 'errorl' )       # try calling error overlay
  return                         # didn't make it so fatal stop
#
  9010 continue
```

```
#
  iev = ossiev( −2020, 'file name too large for eos.' )
  iev = oschan( 'command' )  # go get new command line
  return
#
  9020 continue
#
  iev = ossiev( −1030, 'restart attempted but not coded for.' )
  iev = oschan( 'command' )        # go get new command line
  return
#
  end
```

4.1 Explanation of Example

The above example takes two records of file names which could have been written by a call to OSSEND from the command interpreter. These are an input file and an output file, respectively. These file names are then converted into the internal format file descriptor.

The main operation creates an output file which has each value represent the corresponding value of the input file modulo 10. The files are then closed and the two file descriptors are passed to a overlay called "ADDIT." Note that the file descriptors, rather than the file names, are passed to make the next command easier.

Error conditions are taken care of, including the possibility that the user typed a user interrupt in the middle of processing. This assumes that the routine OSINTR consisted of setting the function value to .RESTART and returning. The routine OSERRI is called to force all output from OSSIEV to appear. Normal processing would do this in a cleaner way with complicated user routines but for the example this shows the operation of the routines. The array WORK is used to hold the data and it is checked for being as large as is needed by the call to OSALOC.

5. TRANSPORTABILITY REVISITED

The aim of the OS routines is to enhance transportability. From this point of view, the OS routines can be broken down into six categories, although any one routine could be rewritten to make it nontransportable in the interest of speed, space requirements, or a particularly difficult machine. The categories, although seemingly clear-cut, are not easily defined.

1. NT (nontransportable): the routines that are fully nontransportable are OSPINF, OSOPNS, OSRDR, OSWTR, OSCLOS, OSOPNS, OSRDS, OSWTS, and the most difficult of all OSCHAN.

2. NT–E (nontransportable–extensions): the routines that are extensions and are nontransportable are OSFLSH, OSECHO, OSTAPE, OSTIME, and OSDATE.

3. MSWV (most systems work versions): the routines that have inefficient transportable versions that work on most machines are the character and bit manipulation routines OSPBTR, OSGBTR, OSPACK, OSUNPK, OSUCHR, and OSPCHR.

4. SNT (simple nontransportable): the set of routines that are nearly transportable:
   (a) OSINFD—May have to partially parse a file name.
   (b) OSGFNM—Generating a file name is system dependent.
   (c) OSINFO—System dependent constants are easily "hard-wired" in.
   (d) OSNAME/OSSIEV/OSPUSH/OSPOP—Need to be able to write to "error" output and cannot use the OS routines because these are also called from within the OS routines.

5. T (transportable): the set of routines that are fully transportable are OSGTNM, OSGIEV, and OSERRI.

6. TV (transportable versions): the following routines that have transportable versions (or for which transportable versions can be written) but may be locally implemented in nontransportable code to make them work faster or use less memory are
   (a) OSGINF—Getting information is transportable if information is in a fixed place.
   (b) OSDLY—Call OSTIME in a loop (very poor on CPU time usage).
   (c) OSWAIT—Do not implement wait option.
   (d) OSGROW—Copy file.
   (e) OSRENM—Copy file, thus renaming it.
   (f) OSCASE—Implement case folding (simple nontransportable) in OSRDS and OSWTS. This routine simply sets the appropriate flag.
   (g) OSSEND/OSRECV—Fixed file name (from OSGFNM); uses random files.
   (h) OSALOC—Implement work array as fixed size; put size in a common and simply check it.
   (i) OSINTR—Do not implement interrupts.
   (j) OSFORM—The setup of forms control is transportable; it is the use in OSWTS that is nontransportable.

The method preferable for implementation of the OS routines requires taking a system similar to the one that the kernel is being implemented on and using it as a basis for modifications.

APPENDIX: OS ROUTINE TOKENS

The mnemonics in this appendix are separated into several different INCLUDE files. The constants described are shown under the INCLUDE file that they are in. The INCLUDE files are named ERRORCODES, INFOCODES, FILECODES, WORDSIZE, and GENCODES.

A.1 ERRORCODES

This INCLUDE file contains the major error codes returned by the OS routines.

1. .NOTFOUNDIEV—Whether a file is found (.OK) or not found.

2. .EOFIEV—End of file condition exists on input file during read.

3. .ILLEGALFDIEV—Either an error exists in a noninitialized file descriptor or the file descriptor has been written into in ways not programmed for.

4. .NODEVIEV—File descriptor built with OSINFD with a null string as input. This IEV returned from OSGINF.

5. .ILLEGALPARAM—This value is returned where a parameter value to a routine is wrong and that error is either obvious or very limited. The OSSIEV message output is particular about what is the problem.

A.2 INFOCODES

This has the tokens used directly and for OSGINF and OSINFO routine calls.

1. .SYSNAM—Return system name as a six character A1 format array (used in OSINFO).

2. .NUMBITSIO—The size of the basic I/O unit in bits.

3. .MINNUMUNITSFORIO—Minimum number of basic units that I/O is done in.

4. .OPTDISK—Optimum disk block size (used in OSGINF).

5. .MEMMAX—Memory maximum, largest value definitely ok in OSALOC (used in OSINFO).

6. .DEVICE—Type of device specified in the file descriptor name (used in OSGINF and OSPINF).

7. .FILETYPE—Type file is open such as .INPUT or .NOTOPEN, etc. (used in OSGINF).

8. .NREC—Number of records in file (random only) (used in OSGINF and OSPINF).

9. .LREC—Number of elements in each record. Actual length is fixed and maximum length in file is variable (used in OSGINF and OSPINF).

10. .MODE—Mode of file (see FILECODES) (used in OSGINF and OSPINF).

11. .CASE—The option to force upper case or not (used in OSGINF).

12. .FORM—The forms control information (used in OSGINF and OSFORM).

13. .ECHO—Whether input from echoable device is suppressed or not (used in OSGINF and OSECHO).

14. .UNIQUE—A unique number for each file (used in OSGINF).

15. .MIXED—Mixed case (used in OSGINF and OSCASE).

16. .UPPER—Force input/output upper case (used in OSGINF and OSCASE).

17. .OVERPRINT—Forms control-overprint next write (used in OSGINF and OSFORM).

18. .NORMAL—Forms control-single line space (used in OSGINF and OSFORM).

19. .TOPPAGE—Forms control-form feed (used in OSGINF and OSFORM).

20. .NOECHO—There is no echo on terminal input (used in OSGINF and OSECHO).

21. .WAIT—Wait for I/O to complete before return (used in OSWAIT, OSRDR, and OSWTR).

22. .NOWAIT—Return immediately after starting I/O request (used in OSWAIT, OSRDR, and OSWTR).

A.3 FILECODES

This INCLUDE file has those pieces of information relating to opening and maintaining a file descriptor.

1. .FDLENGTH—The length of the file descriptor array.

2. .FILENAMELENGTH—The length of the free format string returned from OSGTNM. The string is long enough to add .EOS to the last location and not destroy the last character of the file name.

3. .INPUT—Sequential file open as a read only file.

4. .OUTPUT—Sequential file opened as a write only output.

5. .NEW—Random file opened to be created.

6. .OLD—Random file opened and checked for existence.

7. .NOTOPEN—File is not open.

8. .KEEP—To close a file such that it exists as a permanent file.

9. .DELETE—To get rid of a file; to erase it.

10. .INTMODE—An integer in a INTEGER variable.

11. .REALMODE—Single precision floating point in a REAL variable.

12. .DINTMODE—Double integer mode (may be implemented as INTEGER) in a DINTEGER variable.

13. .HINTMODE—Half integer mode (may be implemented as INTEGER) in a HINTEGER variable.

14. .DBLPRCMODE—Double precision mode (REAL*8) in a DBLPRECISION variable.

15. .CHARMODE—Character mode A1 format characters in a CHARACTER variable.

16. .PCHARMODE—Character mode in packed format in a PCHARACTER variable (but not accessable individually; must use OSPACK, OSUNPK).

17. .LOGMODE—Logical or Boolean mode in a LOGICAL variable.

18. .SYSMODE—System format input file mode in a PCHARACTER variable.

19. .DEVERROR—Device or file name was in error when put into file descriptor.

20. .DEVTERM—Terminal.

21. .DEVPRINT—Printer.

22. .DEVDISK—File name on disk.

23. .DEVTAPE—Magnetic tape.

24. .DEVDISPLAY—Video display.

25. .DEVREADER—Card reader.

26. .DEVPUNCH—Card punch.

27. .EOL—End of line.

28. .ALTEOL—Alternate end of line. (Leave cursor at end of line.)

A.4 WORDSIZE

This INCLUDE file has the information about word size and character set.

1. .NUMBITPERWORD—Number of bits in an INTEGER word.

2. .NUMCHARPERINT—Number of characters in an integer variable.

3. .NUMPCHARPERCHAR—Number of characters in a CHARACTER variable when packed.

4. .LARGEINTEGER—Largest integer on a system for practical purposes.

5. .LARGEREAL—Largest real value on a system.

A.5 GENCODES

This INCLUDE file is sort of a catch-all. It defines many needed constants.

1. CHARACTER—Defines for each machine what a character variable is.

2. HINTEGER—Defines what a half integer is.

3. DINTEGER—Defines what a double integer is.

4. DBLPRC—Defines what a double precision variable is.

5. PCHARACTER—For allocating space only: packed character array definition.

6. PSCHARACTER—See OSUCHR and OSPCHR for explanation (similar to PCHARACTER).

7. .EOS—End of string.

8. .EOF—End of file.

9. .ARB—This is used for arbitrary sized arrays, but only when the actual size is not known.

10. .OK—All items are OK.

11. .WAIT—Wait for I/O to complete. (Put here because read/write is so common.)

12. .NOWAIT—Return immediately. User must call OSWAIT to check that I/O is finished.

REFERENCES

1. E. Guerrieri, *Software/O.S. Interface Kernel User Manual*, Preliminary Version, Technical Report, IPL, Rensselaer Polytechnic Institute, March 1981.
2. R. G. Hamlet and R. M. Haralick, Transportable "package" software, *Software Pract. Exper.* **10**, 1980, 1009–1027.
3. R. G. Hamlet and A. Rosenfeld, Transportable Image-Processing Software, *Proceedings, Nat. Comput. Conf.*, Vol. 48, 267–272, AFIPS Press, June 1979.
4. S. Krusemark and R. M. Haralick, Achieving portability in image processing software packages, *IEEE Computer Society Pattern Recognition and Image Processing Conference*, Las Vegas, Nevada, June 13–17, 1982.