Absoft Support Libraries

Aids to porting to/from UNIX, VAX/VMS

absoft
development tools and languages
2781 Bond Street
Rochester Hills, MI 48309
U.S.A.
Tel (248) 853-0095
Fax (248) 853-0108
support@absoft.com

CHAPTER 1

Introduction to Absoft Support Libraries

This manual describes the two support libraries that provide numerous helpful routines for use with Absoft Fortran 77. These two libraries increase compatibility, allowing for easier porting of code. The Unix library provides routines compatible with those provided by Sun Microsystems for the Sun FORTRAN compiler. The smaller VMS library has a few additional routines with calling conventions that match VAX FORTRAN. None of the routines in this manual are part of the ANSI FORTRAN 77 standard and should be used with caution if portability between platforms is a concern.

Source code to all library routines is supplied in the *example* directories or folders of the operating systems they are installed on.

ABOUT THIS MANUAL

This manual is a reference for using the routines provided in the Unix and VMS libraries.

Chapter 1 is a general introduction to the libraries. It explains the purpose and benefits of the libraries. The notational conventions of the manual are also explained.

Chapter 2 “Using the Support Libraries” discusses how to use the libraries, supplies helpful hints, and provides some examples on using the routines.

Chapter 3 “Support Libraries” lists all of the routines provided, gives a general description of their function, and states how they should be used.

NOTATIONAL CONVENTIONS

The following notation will be used in this manual.

`computer`	font will be used for system generated text (examples, file names, variable names, types, etc.). It should be entered exactly as shown. If input and output appear together, the input will be boldfaced.
-option	font indicates a compiler option.
`italicized`	terms may be replaced by anything which fits the definition. For example, a FORTRAN *`type`* could be `REAL`, `INTEGER`, etc. It is also used for Unix command names.
[optional]	terms enclosed in square brackets are optional.

CHAPTER 2

Using the Support Libraries

This chapter discusses how to use the libraries and general rules that should be followed to insure they are being used properly. The first section details compiler options that should be used when linking with the Unix and VMS libraries. The second section shows examples of compiling code that use these libraries.

NOTE: Some of the routines found in the Unix library may not be available on all operating systems (eg. `topen, tclose, tread`).

A `README` file may be included with these libraries. It contains information specific to Absoft Fortran 77 regarding routines implemented differently on various systems and additional libraries that must be linked to insure proper routine results.

COMPILER OPTIONS

The routine names in the libraries are provided in three spellings to avoid conflicts with other libraries; all uppercase, all uppercase with a trailing underscore, and all lowercase with a trailing underscore:

```
TIME
TIME_
time_
```

You can use any of these entry point names to access the functions in the libraries. Refer to your compiler User Guide to select appropriate compile time options to automatically achieve these spellings.

When porting code from another system, the **-s** option is recommended when compiling. This option causes all local variables to be stored statically, which is the default on many systems. Without the **-s** option, variables local to functions and subroutines will be stored dynamically.

Two additional options helpful when porting code, but not necessary when using these libraries, are **-N3** and **-N51**. The **-N3** option includes record length information for `SEQUENTIAL, UNFORMATTED` files. The **-N51** option causes the `RECL` specifier to be interpreted as the number of 32-bit words in a record.

ROUTINES RETURNING ERROR CODES

Some of the routines in the Unix library return error codes if the call is not successful. The `perror`, `gerror` and `ierrno` routines will assist in determining the meaning of these error codes. This makes it easier to resolve why the error code was returned.

LIBRARY NAMES

The names of the libraries and the directories they are installed in are consistent with the operating system they are implemented on. The following library names are used:

Library	Windows	Mac Classic	Max OS X	Linux
Unix	`unix.lib`	`unixlib.o`	`libU77.a`	`libU77.a`
VMS	`vms.lib`	`vmslib.o`	`libV77.a`	`libV77.a`

EXAMPLE USING THE UNIX LIBRARY

As an example, this small program calls the `sleep` function that is in the Unix library:

```
WRITE(*,*) "Sleeping for a second..."
CALL sleep(1)
WRITE(*,*) "Awake again!"
END
```

It can be compiled with the following command line:

```
f77 -N109 sleep.f unix.lib
```

EXAMPLE USING THE VMS LIBRARY

The VMS library has some `CHARACTER`-based time and date routines. This example calls the `date` subroutine:

```
CHARACTER*9 todays_date
CALL date(todays_date)
WRITE(*,*) "Today is ", todays_date
END
```

It can be compiled with the following command line:

```
f77 -N109 today.f vms.lib
```

CHAPTER 3

Support Libraries

This chapter lists the routines contained in the Unix and VMS libraries. A description of the routine and a small example are provided. References are also provided to indicate additional areas that will provide further information.

VMS LIBRARY ROUTINES

date `subroutine date(string)` (VMS compatible)
`character*9 string`

The `date` subroutine sets *`string`* to the current date in a format like "`26-Mar-91`".

Example:
```
character*9 the_date
call date(the_date)
```

---

idate `subroutine idate(month, day, year)` (VMS compatible)
`integer*4 month, day, year`

The `idate` subroutine sets the *`month`*, *`day`*, and *`year`* for the current date.

Example:
```
integer*4 month, day, year
call idate(month, day, year)
```

---

mvbits `subroutine mvbits(source, start1, len,` (VMS compatible)
`dest, start2)`
`integer*4 source, start1, len, dest, start2`

The `mvbits` subroutine is built into the Absoft FORTRAN 77 run time library and can be used without linking the VMS library with `-lV77`. It is documented here for completeness. This routine moves bits from *`source`* to *`dest`*. *`Len`* number of bits are moved starting from bit *`start1`* in source to *`start2`* in *`dest`*. The `mvbits` subroutine is compatible with MIL-STD-1753.

Example:
```
integer*4 source, middle16
call mvbits(source, 8, 16, middle16, 0)
```

---

ran `real*4 ran(seed)` (VMS compatible)
`integer*4 seed`

The `ran` function returns a random number between 0.0 inclusive and 1.0 exclusive. The argument *`seed`* must be a variable, array element, or RECORD element, and not a constant.

Example:
```
real*4 ran, result
integer*4 seed/760013/
result = ran(seed)
```

---

secnds `real*4 secnds(base)` (VMS compatible)
`real*4 base`

The `secnds` function returns the time, in seconds, since midnight minus the argument *`base`*.

Example:
```
real*4 secnds, diff, start
start = secnds(0)
.
.
.
diff = secnds(start)
```

---

time `subroutine time(string)` (VMS compatible)
`character*8 string`

The `time` subroutine sets *`string`* to the current time in a format like "`13:08:56`".

Example:
```
character*8 the_time
call time(the_time)
```

UNIX LIBRARY ROUTINES

abort `subroutine abort`

The `abort` subroutine closes all FORTRAN units and aborts execution causing a core dump. See also *abort*(3).

---

access
```
integer*4 function access(name, mode)
character*(*) name, mode
```

The `access` function determines if the specified file *`name`* can be accessed with the *`mode`* derived from one or more of the following:

- `r` read permission
- `w` write permission
- `x` execute permission

The return code is `0` if the file can be accessed in the specified modes. An error code is returned otherwise. See also *access*(2).

Example:
```
integer*4 access
if (access('test_file', 'rw') .eq. 0) ...
```

---

alarm
```
integer*4 function alarm(time, sbrtn)
integer*4 time
external sbrtn
```

The `alarm` function schedules to have the subroutine *`sbrtn`* called after *`time`* seconds. A *`time`* of `0` will turn off a pending alarm and the return value will be the time that was remaining. See also *alarm*(3) and the `signal` function.

Example:
```
integer*4 alarm, i
external alarm_sub
i = alarm(30, alarm_sub)
.
.
.
subroutine alarm_sub()
end
```

---

bic

```
subroutine bic(bitnum, word)
integer*4 bitnum, word
```

The `bic` subroutine clears the single bit *`bitnum`* in *`word`*. Using the intrinsic function `IBCLR()` is more efficient and more compatible than the `bic` subroutine.

Example:
```
integer*4 negative
call bic(31, negative)
```

---

bis

```
subroutine bis(bitnum, word)
integer*4 bitnum, word
```

The `bis` subroutine sets the single bit *`bitnum`* in *`word`*. See also the `setbit` function. Using the intrinsic function `IBSET()` is more efficient and more compatible than the `bis` subroutine.

Example:
```
integer*4 positive
call bis(31, positive)
```

---

bit

```
logical function bit(bitnum, word)
integer*4 bitnum, word
```

The `bit` function returns `.true.` if bit *`bitnum`* is set in *`word`* otherwise, it returns `.false.`. Using the intrinsic function `BTEST()` is more efficient and more compatible than the `bit` function.

Example:
```
integer*4 either
logical bit
if (bit(31, either)) ...
```

---

chdir

```
integer*4 function chdir(dirname)
character*(*) dirname
```

The `chdir` function changes the default directory to *`dirname`* when referencing files. The return code is `0` if the directory change was successful. An error code is returned otherwise. See also *chdir*(2), the `getcwd` function.

Example:
```
integer*4 chdir
if (chdir('/home') .eq. 0) ...
```

---

chmod

```
integer*4 function chmod(name, mode)
character*(*) name, mode
```

The `chmod` function changes the filesystem mode for the file *`name`*. The *`mode`* may be any string that is acceptable to the *chmod*(1) command. The return code is `0` if the directory change was successful. An error code is returned otherwise. See also *chmod*(1).

Example:

```
integer*4 chmod
if (chmod('test_file', 'oug+r') .eq. 0) ...
```

---

ctime

```
character*24 function ctime(stime)
integer*4 stime
```

The `ctime` function returns the date and time of the system time *`stime`* as a `CHARACTER*24` string in a format like "`Sun Sep 16 01:03:52 1973`". See also *ctime*(3) and the `time` function.

Example:

```
character*24 the_date, ctime
the_date = ctime(670000000)
write(*,*) "Written on: ", the_date
```

---

dflmax

```
real*8 function dflmax()
```

The `dflmax` function returns the maximum positive `real*8` number. See also the `dflmin` function.

Example:

```
real*8 max, dflmax
max = dflmax()
write(*,*) "Maximum REAL*8 is: ", max
```

---

dflmin

```
real*8 function dflmin()
```

The `dflmin` function returns the minimum positive `real*8` number. See also the `dflmax` function.

Example:

```
real*8 min, dflmin
min = dflmin()
write(*,*) "Minimum REAL*8 is: ", min
```

---

drand

```
real*8 function drand(flag)
integer*4 flag
```

The `drand` function returns a random `real*8` number between 0.0 and 1.0 according to *`flag`*. See also the `rand` function which returns `real*4` numbers.

`flag`	action
`0`	returns next random number in sequence
`1`	restart generator and return first number of sequence
other	seed generator with *`flag`* and return first number of new sequence

Example:

```
real*8 number, drand
number = drand(0)
write(*,*) "Random number is: ", number
```

---

dtime

```
real*4 function dtime(tarray)
real*4 tarray(2)
```

The `dtime` function returns the elapsed time, in seconds, since the previous call to `dtime` or since the start of execution on the first call. On return, the first element of *`tarray`* contains the elapsed user time and the second contains the elapsed system time. The return value is the sum of these two times. See also the `etime` function.

Example:

```
real*4 dtime
real*4 tarray(2), total
total = dtime(tarray)
```

---

etime

```
real*4 function etime(tarray)
real*4 tarray(2)
```

The `etime` function returns the elapsed time, in seconds, since the start of execution. On return, the first element of *`tarray`* contains the elapsed user time and the second contains the elapsed system time. The return value is the sum of these two times. See also the `dtime` function.

Example:

```
real*4 etime
real*4 tarray(2), total
total = etime(tarray)
```

---

exit

```
subroutine exit(status)
integer*4 status
```

The `exit` subroutine closes all FORTRAN units and exits the program. The *`status`* is returned to the parent process which may be the command shell. See also *exit*(2).

Example:
```
if (errors) then
  exit(1)
else
  exit(0)
end if
```

---

fdate

```
subroutine fdate(string)
```
(subroutine interface)

```
character*24 string
```

or

```
character*24 function fdate()
```
(function interface)

The `fdate` subroutine returns the current date and time in a `CHARACTER*24` string in a format like "`Sun Sep 16 01:03:52 1973`". This routine may be called as a function or subroutine. See also *ctime*(3).

Example:
```
character*24 the_date
call fdate(the_date)
write(*,*) "Today is: ", the_date
```

---

fgetc

```
integer*4 function fgetc(lunit, char)
integer*4 lunit
character char
```

The `fgetc` function returns in *`char`* the next character from the file associated with the FORTRAN unit *`lunit`*. Because normal FORTRAN I/O is bypassed, it is not recommended mixing standard FORTRAN I/O with this function. A return code of `0` indicates success, `-1` indicates that the end of the file has been reached, and positive values are error codes. See also *getc*(3) and the `getc` function.

Example:
```
integer*4 test, fgetc
character c
open(unit=1, file="test_file")
test = fgetc(1, c)
```

---

flmax `real*4 function flmax()`

The `flmax` function returns the maximum positive `real*4` number. See also the `inmax` and `flmin` functions.

Example:
```
real*4 max, flmax
max = flmax()
write(*,*) "Maximum REAL*4 is: ", max
```

---

flmin `real*4 function flmin()`

The `flmin` function returns the minimum positive `real*4` number. See also the `flmax` function.

Example:
```
real*4 min, flmin
min = flmin()
write(*,*) "Minimum REAL*4 is: ", min
```

---

flush
```
subroutine flush(lunit)
integer*4 lunit
```

The `flush` subroutine flushes the file buffers for the FORTRAN unit *`lunit`*.

Example:
```
call flush(1)
```

---

fork `integer*4 function fork()`

The `fork` function creates a child process which is an exact copy of the calling process. All FORTRAN units are flushed before the fork is made. The return code is negative if the call was not successful. See *fork*(2) for a complete description and see the `perror` function for error reporting.

Example:
```
integer*4 test, fork
test = fork()
```

---

fputc

```
integer*4 function fputc(lunit, char)
integer*4 lunit
character char
```

The `fputc` function writes the character *`char`* to the file associated with the FORTRAN unit *`lunit`*. Because normal FORTRAN I/O is bypassed, it is not recommended mixing standard FORTRAN I/O with this function. The return code is `0` if successful and an error code otherwise. See also *putc*(3) and the `putc` function.

Example:
```
integer*4 test, fputc
open(unit=1, file="test_file")
test = fputc(1, 'a')
```

---

free

```
subroutine free(pointer)
integer*4 pointer
```

The `free` subroutine frees a block of memory at *`pointer`* that was allocated by a previous call to the `malloc` function. See also the `malloc` function for an example.

---

fseek

```
integer*4 function fseek(lunit, offset, from)
integer*4 lunit, offset, from
```

The `fseek` function changes the current file position of the FORTRAN unit *`lunit`*. The offset is relative to the position specified by *`from`*:

- `0` beginning of the file
- `1` current file position
- `2` end of the file

The return code is `0` if the call was successful. It is not recommended mixing standard FORTRAN I/O with this function. See also *lseek*(2) and the `ftell` function.

Example:
```
integer*4 fseek
test = fseek(1, 1000, 0)
```

---

fstat

```
integer*4 function fstat(lunit, iarray)
integer*4 lunit
integer*4 iarray(13)
```

The `fstat` function returns statistics about the file associated with the FORTRAN unit *`lunit`*. The array *`iarray`* is filled with the following information:

`iarray` index	description
1	device on which the file resides
2	the serial number for the file (inode)
3	file mode
4	number of hard links to the file
5	user ID of file owner
6	group ID of file owner
7	device identifier (devices only)
8	size, in bytes, of file
9	last file access time
10	last file modify time
11	last file status change time
12	preferred block size for this file system
13	actual number of blocks allocated

The return code is `0` if successful and an error code otherwise. See also *stat*(2) and the `stat` and `lstat` functions.

Example:
```
integer*4 test, fstat
integer*4 array(13)
open(unit=1, file="test_file")
test = fstat(1, array)
write(*,*) "File size is: ", array(8)
```

---

ftell

```
integer*4 function ftell(lunit)
integer*4 lunit
```

The `ftell` function returns the current file position as an offset in bytes from the beginning of the file. The return code is `0` or positive if the call was successful. See also *lseek*(2) and the `fseek` function.

Example:
```
integer*4 ftell, position
position = ftell(1)
```

---

gerror

```
subroutine gerror(string)
character*(*) string
```
(subroutine interface)

or

```
character*(*) function gerror()
```
(function interface)

The `gerror` subroutine returns the most recently encountered system error message in *`string`*. This routine may be called as a function or subroutine. See also the `perror` and `ierrno` functions.

Example:
```
integer*4 test, chdir
character*100 the_error
test = chdir("/bad_directory")
if (test .ne. 0) then
  call gerror(the_error)
end if
```

---

getarg

```
subroutine getarg(k, arg)
integer*4 k
character*(*) arg
```

The `getarg` subroutine gets the *`k`*th argument from the command line and copies it into *`arg`*. For the following command line,

```
a.out first second third
```

the 0th argument is '`a.out`', the 1st is `'first'`, and so on. Use the `iargc` function to get the total number of arguments available.

Example:
```
character*100 string
call getarg(0, string)
write(*,*) "This executable is: ", string
```

---

getc

```
integer*4 function getc(char)
character char
```

The `getc` function returns in *`char`* the next character from the file associated with FORTRAN unit 5 which is usually standard input. Because normal FORTRAN I/O is bypassed, it is not recommended mixing standard FORTRAN I/O with this function. A return code of `0` indicates success, `-1` indicates that the end of the file has been reached, and positive values are error codes. See also *getc*(3) and the `fgetc` function.

Example:
```
integer*4 test, getc
character c
open(unit=5, file="test_file")
test = getc(c)
```

---

getcwd

```
integer*4 function getcwd(dirname)
character*(*) dirname
```

The `getcwd` function returns the current working directory pathname in *`dirname`*. A return code of `0` indicates success, otherwise an error occurred. See also *getwd*(3) and the `chdir` function.

Example:
```
integer*4 test, getcwd
character*100 path
test = getcwd(path)
```

---

getenv

```
subroutine getenv(ename, evalue)
character*(*) ename, evalue
```

The `getenv` subroutine returns in *`evalue`* the string associated with the environment variable *`ename`*. If an environment variable is not found, *`evalue`* is filled with blanks. See also *getenv*(3).

Example:
```
character*100 string
call getenv("TERM", string)
write(*,*) "Terminal type is: ", string
```

---

getfd

```
integer*4 function getfd(lunit)
integer*4 lunit
```

The `getfd` function returns the file descriptor associated with the FORTRAN unit *`lunit`*. If the unit is not connected, `-1` is returned. See also *open*(2).

Example:
```
integer*4 fd, getfd
fd = getfd(5)
```

---

getlog

```
subroutine getlog(name)
character*(*) name
```

The `getlog` subroutine returns in *`name`* the user's login name. See also *getlogin*(3).

Example:
```
character*100 my_name
call getlog(my_name)
write(*,*) "Currently logged in as: ", my_name
```

---

getgid `integer*4 function getgid()`

The `getgid` function returns the group ID number of the current process. See also *getgid*(2).

Example:
```
integer*4 getgid, my_gid
my_gid = getgid()
write(*,*) "My group ID is: ", my_gid
```

---

getpid `integer*4 function getpid()`

The `getpid` function returns the ID number of the current process. See also *getpid*(2).

Example:
```
integer*4 getpid, my_pid
my_pid = getpid()
write(*,*) "My process ID is: ", my_pid
```

---

getuid `integer*4 function getuid()`

The `getuid` function returns the user ID number of the current process. See also *getuid*(2).

Example:
```
integer*4 getuid, my_uid
my_uid = getuid()
write(*,*) "My user ID is: ", my_uid
```

---

gmtime
```
subroutine gmtime(stime, tarray)
integer*4 stime
integer*4 tarray(9)
```

The `gmtime` function returns information about the system time *`stime`* in the array *`tarray`* as follows. The GMT time zone is used.

`tarray` index	description
1	seconds
2	minutes
3	hours (GMT)
4	day of the month
5	month of the year
6	year (0 is 1900)
7	day of the week
8	day of the year
9	1 if DST is in effect

See also *ctime*(3), the `ltime` function and the `time` function.

Example:
```
integer tarray(9)
call gmtime(670000000, tarray)
write(*,*) "Year written is: ", 1900 + tarray(6)
```

---

hostnm
```
integer*4 function hostnm(name)
character*(*) name
```

The `hostnm` function sets the name of the host in *`name`*. The return code is `0` if successful. See also *gethostname*(2) and *uname*(2).

Example:
```
integer*4 test, hostnm
character*100 string
test = hostnm(string)
write(*,*) "The host name is: ", string
```

---

iargc
```
integer*4 function iargc()
```

The `iargc` function returns the number of arguments on the command line minus one. For the following command line,

```
a.out first second third
```

the value returned by `iargc` is 3. To get the arguments themselves, use the `getarg` function.

Example:
```
integer*4 iargc
write(*,*) "Number of arguments: ", iargc()
```

---

idate
```
subroutine idate(iarray)
integer*4 iarray(3)
```

The `idate` subroutine fills the array *`iarray`* with the following values:

`iarray` index	description	range
1	day	1-31
2	month	1-12
3	year	1900+

See also the `fdate` subroutine in this library and the `idate` subroutine in the VMS library which has different calling conventions that are compatible with VAX FORTRAN.

Example:
```
integer*4 iarray(3)
call idate(iarray)
```

---

ierrno `integer*4 function ierrno()`

The `ierrno` function returns the most recently encountered system error number. Do not use the return value to determine if an error had occurred. See also the `perror` and `gerror` functions.

Example:
```
integer*4 last_error, ierrno
last_error = ierrno()
```

---

inmax `integer*4 function inmax()`

The `inmax` function returns the maximum positive integer. See also the `flmax` and `flmin` functions.

Example:
```
integer*4 max, inmax
max = inmax()
write(*,*) "Maximum integer is: ", max
```

---

ioinit `logical function ioinit(`*`cctl`*`, `*`bzro`*`, `*`apnd`*`, `*`prefix`*`, `*`vrbose`*`)`
`logical `*`cctl`*`, `*`bzro`*`, `*`apnd`*`, `*`vrbose`
`character*(*) `*`prefix`

The `ioinit` function opens FORTRAN units with file names obtained from a set of environment variables composed of the characters *`prefix`* followed by a two-digit FORTRAN unit number. Some characteristics of how each file is opened are determined from the logical flags:

flag	meaning when `.true.`	meaning when `.false.`
cctl	ACTION='PRINT'	ACTION='BOTH'
bzro	BLANK='ZERO'	BLANK='NULL'
apnd	POSITION='APPEND'	POSITION='ASIS'

The *`vrbose`* flag, when `.true.`, causes the `ioinit` function to display its activity on standard error.

As an example, if the following environment variables are set-up,

```
setenv FILE01 data_file1
setenv FILE02 data_file2
```

the following call opens the files `data_file1` and `data_file2` on units 1 and 2, respectively.

```
call ioinit(.false., .false., .false., 'FILE', .false.)
```

The `ioinit` function only opens files, and the flags do not effect any other files opened with the FORTRAN OPEN statement. The return code is always `.true.`.

---

irand

```
integer*4 function irand(flag)
integer*4 flag
```

The `irand` function returns a random `integer*4` number between 0 and the largest integer according to *`flag`*.

`flag`	action
`0`	returns next random number of sequence
`1`	restart generator and return first number of sequence
other	seed generator with *`flag`* and return first number of new sequence

See also the `rand` function which returns `real*4` numbers.

Example:
```
integer*4 number, irand
number = irand(0)
write(*,*) "Random number is: ", number
```

---

isatty

```
logical*4 function isatty(lunit)
integer*4 lunit
```

The `isatty` function returns `.true.` if a terminal device is connected to the FORTRAN unit *`lunit`*. In Absoft FORTRAN 77, preconnected units are not assigned to a device until referenced. See also *ttynam*(3) and the `ttynam` function.

Example:
```
logical*4 isatty
if (isatty(1)) ...
```

---

itime

```
subroutine itime(iarray)
integer*4 iarray(3)
```

The `itime` subroutine fills the array *`iarray`* with the following values:

iarray index	description	range
1	hour	0-23
2	minute	0-59
3	second	0-59

See also the `ctime` subroutine in this library and the `time` subroutine in the VMS library.

Example:
```
integer*4 iarray(3)
call itime(iarray)
```

---

kill
```
integer*4 function kill(pid, signum)
integer*4 pid, signum
```

The `kill` function sends the signal *`signum`* to the process *`pid`*. The return code is `0` if successful and an error code otherwise. See also *kill*(2) and for a list of signals see *sigvec*(2).

Example:
```
integer*4 test, kill
test = kill(123, 9)
```

---

link
```
integer*4 function link(name1, name2)
character*(*) name1, name2
```

The `link` function creates a link of the file *`name1`* to the new file *`name2`*. The return code is `0` if successful and an error code otherwise. See also *link*(2) and the `symlnk` function.

Example:
```
integer*4 test, link
test = link("test_file", "new_file")
```

---

lnblnk
```
integer*4 function lnblnk(string)
character*(*) string
```

The `lnblnk` function returns the index of the last non-blank character in *`string`*.

Example:
```
integer*4 lnblnk, lastnb
lastnb = lnblnk('Hello world ')
```

---

long

```
integer*4 function long(int2)
integer*2 int2
```

The `long` function converts its `integer*2` argument *`int2`* into an `integer*4` value. To avoid conflict with the intrinsic function `long()` in Absoft FORTRAN 77, you must declare this function as external before use:

```
external long
```

Example:

```
integer*4 result, long
integer*2 i2
external long
result = long(i2)
```

---

lstat

```
integer*4 function lstat(name, iarray)
character*(*) name
integer*4 iarray(13)
```

The `lstat` function returns statistics about the file *`name`*. If name is a symbolic link, information is returned about the link. The array *`iarray`* is filled with the following information:

`iarray` index	description
1	device on which the file resides
2	the serial number for the file (inode)
3	file mode
4	number of hard links to the file
5	user ID of file owner
6	group ID of file owner
7	device identifier (devices only)
8	size, in bytes, of file
9	last file access time
10	last file modify time
11	last file status change time
12	preferred block size for this file system
13	actual number of blocks allocated

The return code is `0` if successful and an error code otherwise. See also *stat*(2) and the `stat` and `fstat` functions.

Example:

```
integer*4 test, lstat
integer*4 array(13)
test = lstat("test_file", array)
write(*,*) "File size is: ", array(8)
```

---

ltime

```
subroutine ltime(stime, tarray)
integer*4 stime
integer*4 tarray(9)
```

The `ltime` function returns information about the system time *`stime`* in the array *`tarray`* as follows. The local time zone is used.

tarray index	description
1	seconds
2	minutes
3	hours (local time zone)
4	day of the month
5	month of the year
6	year (0 is 1900)
7	day of the week
8	day of the year
9	1 if DST is in effect

See also *ctime*(3) and the `time` function.

Example:

```
integer tarray(9)
call ltime(670000000, tarray)
write(*,*) "Year written is: ", 1900 + tarray(6)
```

---

malloc

```
integer*4 function malloc(size)
integer*4 size
```

The `malloc` function allocates a block of memory containing *`size`* bytes. Zero is returned if the allocation could not be made. This function is most useful when it is declared as a pointer as in the example below. See also the `free` function.

Example:

```
STRUCTURE /str/
  integer*4 first_element
  integer*4 second_element
END STRUCTURE
RECORD /str/ my_struct
POINTER (pmy_struct, my_struct)
INTEGER malloc_result
POINTER (malloc, malloc_result)
pmy_struct = malloc(1000)
 .
 .
 .
call free(pmy_struct)
```

---

perror

```
subroutine perror(string)
character*(*) string
```

The `perror` subroutine writes the most recently encountered system error message to FORTRAN unit 0 (standard error). The message is preceded by *`string`*. See also the `gerror` and `ierrno` functions.

Example:
```
integer*4 test, chdir
test = chdir("/bad_directory")
if (test .ne. 0) then
  call perror("MyProgram")
end if
```

---

putc

```
integer*4 function putc(char)
character char
```

The `putc` function writes the character *`char`* to the file associated with FORTRAN unit 6 which is usually standard output. Because normal FORTRAN I/O is bypassed, it is not recommended mixing standard FORTRAN I/O with this function. The return code is `0` if successful and an error code otherwise. See also *putc*(3) and the `fputc` function.

Example:
```
integer*4 test, putc
open(unit=6, file="test_file")
test = putc('a')
```

---

qsort

```
subroutine qsort(array, len, size, compare)
integer*4 len, size
external compare
```

The `qsort` subroutine sorts the first *`len`* elements of *`array`* by using the comparison routine *`compare`* defined below. See also *qsort*(3).

The byte size of each element is determined from the *`size`* argument:

Array type	Value for *`size`* argument
`integer*2`	`2`
`integer*4`	`4`
`real*4`	`4`
`real*8`	`8`
`double precision`	`8`
`complex*8`	`8`
`complex*16`	`16`
`double complex`	`16`
`character`	length of character element

The user supplied *compare* routine must return an `integer*2` value as shown in this example which compares two `real*8` numbers:

```
integer*2 function real8_compare(first, second)
real*8 first, second

real8_compare = 1                          ! first > second
if (first .eq. second) real8_compare = 0   ! first = second
if (first .lt. second) real8_compare = -1  ! first < second
end
```

Example:
```
real*8 a(10)
external real8_compare
call qsort(a, 10, 8, real8_compare)
```

---

rand

```
real*4 function rand(flag)
integer*4 flag
```

The `rand` function returns a random `real*4` number between 0.0 and 1.0 according to *flag*:

flag	action
0	returns next random number of sequence
1	restart generator and return first number of sequence
other	seed generator with *flag* and return first number of new sequence

See also the `irand` function which returns `integer*4` numbers and the `drand` function which returns `real*8` numbers.

Example:
```
real*4 number, rand
number = rand(0)
write(*,*) "Random number is: ", number
```

---

rename

```
integer*4 function rename(from, to)
character*(*) from, to
```

The `rename` function changes the file name of the file *from* to *to*. If the file *to* exists, it will first be removed. The return code is `0` if successful and an error code otherwise. See also *rename*(2).

Example:
```
integer*4 test, rename
test = rename("test_file", "new_file")
```

---

rindex

```
integer*4 function rindex(string, substr)
character*(*) string, substr
```

The `rindex` function is similar to the intrinsic function `index`, but it returns the index of the last occurrence of *`substr`* in *`string`*. Zero is returned if the string is not found.

Example:
```
integer*4 rindex, first, last
first = index('11ab1111ab1ab', 'ab')
last = rindex('11ab1111ab1ab', 'ab')
```

---

setbit

```
subroutine setbit(bitnum, word, state)
integer*4 bitnum, word, state
```

The `setbit` subroutine sets the single bit *`bitnum`* in *`word`* only if *`state`* is non-zero. Otherwise, the bit is cleared. See also the `bic`, `bis`, and `bit` functions.

Example:
```
integer*4 either, flag
call setbit(31, either, flag)
```

---

short

```
integer*2 function short(int4)
integer*4 int4
```

The `short` function converts its `integer*4` argument *`int4`* into an `integer*2` value.

Example:
```
integer*2 result, short
integer*4 i4
result = short(i4)
```

---

signal

```
integer*4 function signal(signum, proc, flag)
integer*4 signum, flag
external proc
```

The `signal` function sets up a signal handling routine *`proc`* that is called when a signal *`signum`* is received. When *`flag`* is `-1`, the signal handler is set-up. When *`flag`* is `0` or positive, *`proc`* is ignored and the value of *`flag`* is the signal definition for the system. Specifically, when *`flag`* is `0`, the default action is taken for *`signum`* signals. When *`flag`* is `1`, the signal is ignored. A return code greater than `1` is the address of the previous handler for *`signum`*. This may be used to restore a previous signal handler. A negative return code is the negative error code. See also *signal*(3) and the `kill` function.

Example:
```
integer*4 test, signal
external handler
test = signal(14, handler, -1)
```

---

sleep

```
subroutine sleep(time)
integer*4 time
```

The `sleep` subroutine suspends execution for about *`time`* seconds. See also *sleep*(3).

Example:
```
call sleep(4)
```

---

stat

```
integer*4 function stat(name, iarray)
character*(*) name
integer*4 iarray(13)
```

The `stat` function returns statistics about the file *`name`*. The array *`iarray`* is filled with the following information:

`iarray` index	description
1	device on which the file resides
2	the serial number for the file (inode)
3	file mode
4	number of hard links to the file
5	user ID of file owner
6	group ID of file owner
7	device identifier (devices only)
8	size, in bytes, of file
9	last file access time
10	last file modify time
11	last file status change time
12	preferred block size for this file system
13	actual number of blocks allocated

The return code is `0` if successful and an error code otherwise. See also *stat*(2) and the `lstat` and `fstat` functions.

Example:
```
integer*4 test, stat
integer*4 array(13)
test = stat("test_file", array)
write(*,*) "File size is: ", array(8)
```

---

symlnk `integer*4 function symlnk(name1, name2)`
`integer*4 name1, name2`

The `symlnk` function creates a symbolic link of the file *`name1`* to the new file *`name2`*. The return code is `0` if successful and an error code otherwise. See also *symlink*(2) and the `link` function.

Example:
```
integer*4 test, symlnk
test = symlnk("test_file", "new_file")
```

---

system `integer*4 function system(string)`
`character*(*) string`

The `system` function executes the command line *`string`* in a shell. The return code is the exit status of the shell.

Example:
```
integer*4 test, system
test = system("ls -l")
```

---

tclose `integer*4 function tclose(tlu)`
`integer*4 tlu`

The `tclose` function closes the tape device associated with the *`tlu`*. The return code is `0` if the call was successful. See also *close*(2), *mtio*(4), and the `topen` function.

Example:
```
integer test, tclose
test = tclose(0)
```

---

time `integer function time()`

The `time` function returns the seconds since 00:00:00 GMT January 1, 1970, measured in seconds. See also *time*(3), the `ctime` function, the `gmtime` function and the `ltime` function.

Example:
```
integer now, time
now = time()
```

---

topen

```
integer*4 function topen(tlu, devname, islabeled)
integer*4 tlu
character*(*) devname
logical*4 islabeled
```

The `topen` function associates a logical tape unit (*`tlu`*) with a device *`devname`*. The *`tlu`* may be 0 to 7 and is used in the other tape routines to reference the tape device. The flag *`islabeled`* should be set to `.true.` if the tape has a label. The return code is `0` if the call was successful. See also *open*(2) and *mtio*(4).

Example:

```
integer test, topen
test = topen(0, "/dev/rst0", .false.)
```

---

tread

```
integer*4 function tread(tlu, buffer)
integer*4 tlu
character*(*) buffer
```

The `tread` function reads a block of data into *`buffer`* from the tape device associated with the *`tlu`*. The return code is `0` if the call was successful. See also *read*(2), *mtio*(4), and the `topen` function.

Example:

```
integer test, tread
character*1024 buffer
test = tread(0, buffer)
```

---

trewin

```
integer*4 function trewin(tlu)
integer*4 tlu
```

The `trewin` function rewinds the tape device associated with the *`tlu`*. The return code is `0` if the call was successful. See also *ioctl*(2) and *mtio*(4).

Example:

```
integer test, trewin
test = trewin(0)
```

---

tskipf

```
integer*4 function tskipf(tlu, nfiles, nrecords)
integer*4 tlu, nfiles, nrecords
```

The `tskipf` function skips over *`nfiles`* end-of-file marks and then skips over *`nrecords`* blocks of the tape device associated with the *`tlu`*. The return code is `0` if the call was successful. See also *ioctl*(2) and *mtio*(4).

Example:

```
integer test, tskip
test = tskip(0, 0, 1)
```

---

tstate

```
integer*4 function tstate(tlu, fileno, recno, errf, eoff,
                          eotf, tcsr)
integer*4 tlu, fileno, recno
logical*4 errf, eoff, eotf
integer*4 tcsr
```

The `tstate` function returns information about the tape device associated with the *`tlu`*:

`fileno` current file number

`recno` current record number

`errf` flag if an error had previously occurred

`eoff` flag if at the end-of-file. When `.true.`, the tread function will not work. This flag may be set to `.false.` by calling `tskipf(tlu, 1, 0)`.

`eotf` flag if at end-of-tape (not reliable)

`tcsr` contents of the tape control status register

The return code is `0` if the call was successful. See also *ioctl*(2) and *mtio*(4).

Example:
```
integer*4 test, tstate
integer*4 fileno, recno, tcsr
logical*4 errf, eoff, eotf
test = tstate(0, fileno, recno, errf, eoff, eotf, tcsr)
```

---

ttynam

```
character*(*) function ttynam(lunit)
integer*4 lunit
```

The `ttynam` function returns the name of the terminal device connected to the FORTRAN unit *`lunit`*. If *`lunit`* is not connected to a terminal device, blanks are returned. In Absoft FORTRAN 77, preconnected units are not assigned to a device until referenced. See also *ttynam*(3) and the `isatty` function.

Example:
```
character*100 name
name = ttynam(1)
```

---

twrite

```
integer*4 function twrite(tlu, buffer)
integer*4 tlu
character*(*) buffer
```

The `twrite` function writes a block of data from *`buffer`* to the tape device associated with the *`tlu`*. The return code is `0` if the call was successful. See also *write*(2), *mtio*(4), and the `topen` function.

Example:

```
integer test, twrite
character*1024 buffer
test = twrite(0, buffer)
```

---

unlink

```
integer*4 function unlink(name)
character*(*) name
```

The `unlink` function removes the file *`name`*. The return code is `0` if successful and an error code otherwise. See also *unlink*(2).

Example:

```
integer*4 test, unlink
test = unlink("test_file")
```

---

wait

```
integer*4 function wait(status)
character*(*) status
```

The `wait` function suspends execution until a signal is received or a child process terminates. A positive return code is the process ID of a child and *`status`* is the termination status. Otherwise, a negative return code is a negative error code. See also *wait*(2) and the `signal` function.

Example:

```
integer*4 test, wait, status
test = unlink(status)
```

U

V

W