Absoft Support Libraries

Aids to porting to/from UNIX, VAX/VMS

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

absoft
development tools and languages
part of this publication may be reproduced or used in any form by any means, without the prior written permission of Absoft Corporation.

THE INFORMATION CONTAINED IN THIS PUBLICATION IS BELIEVED TO BE ACCURATE AND RELIABLE. HOWEVER, ABSOFT CORPORATION MAKES NO REPRESENTATION OF WARRANTIES WITH RESPECT TO THE PROGRAM MATERIAL DESCRIBED HEREIN AND SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. FURTHER, ABSOFT RESERVES THE RIGHT TO REVISE THE PROGRAM MATERIAL AND MAKE CHANGES THEREIN FROM TIME TO TIME WITHOUT OBLIGATION TO NOTIFY THE PURCHASER OF THE REVISION OR CHANGES. IN NO EVENT SHALL ABSOFT BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE PURCHASER'S USE OF THE PROGRAM MATERIAL.

U.S. GOVERNMENT RESTRICTED RIGHTS — The software and documentation are provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the Government is subject to restrictions set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at 252.227-7013. The contractor is Absoft Corporation, 2781 Bond Street, Rochester Hills, Michigan 48309.

ABSOFT CORPORATION AND ITS LICENSOR(S) MAKE NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, REGARDING THE SOFTWARE. ABSOFT AND ITS LICENSOR(S) DO NOT WARRANT, GUARANTEE OR MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RESULTS OF THE USE OF THE SOFTWARE IN TERMS OF ITS CORRECTNESS, ACCURACY, RELIABILITY, CURRENTNESS, OR OTHERWISE. THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE SOFTWARE IS ASSUMED BY YOU. THE EXCLUSION OF IMPLIED WARRANTIES IS NOT PERMITTED BY SOME STATES. THE ABOVE EXCLUSION MAY NOT APPLY TO YOU.

IN NO EVENT WILL ABSOFT, ITS DIRECTORS, OFFICERS, EMPLOYEES OR LICENSOR(S) BE LIABLE TO YOU FOR ANY CONSEQUENTIAL, INCIDENTAL OR INDIRECT DAMAGES (INCLUDING DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, AND THE LIKE) ARISING OUT OF THE USE OR INABILITY TO USE THE SOFTWARE EVEN IF ABSOFT HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. BECAUSE SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY FOR CONSEQUENTIAL OR INCIDENTAL DAMAGES, THE ABOVE LIMITATIONS MAY NOT APPLY TO YOU. Absoft and its licensor(s) liability to you for actual damages for any cause whatsoever, and regardless of the form of the action (whether in contract, tort, (including negligence), product liability or otherwise), will be limited to $50.

Absoft, the Absoft logo, Fx, and MacFortran are trademarks of Absoft Corporation
Apple, the Apple logo, and HyperCard are registered trademarks of Apple Computer, Inc.
CF90 is a trademark of Cray Research, Inc.
IBM, MVS, and RS/6000 are trademarks of IBM Corp.
Macintosh, NeXT, and NeXTSTEP, are trademarks of Apple Computer, Inc., used under license.
MetroWerks and CodeWarrior are trademarks of MetroWerks, Inc.
MS-DOS is a trademark of Microsoft Corp.
Pentium, Pentium Pro, and Pentium II are trademarks of Intel Corp.
PowerPC is a trademark of IBM Corp., used under license.
Sun and SPARC are trademarks of Sun Microsystems Computer Corp.
UNIX is a trademark of the Santa Cruz Operation, Inc.
VAX and VMS are trademarks of Digital Equipment Corp.
Windows NT, Windows 95, Windows 98, Windows 3.1, and Win32s are trademarks of Microsoft Corp.
All other brand or product names are trademarks of their respective holders.

Copyright © 1991-2002 Absoft Corporation and its licensor(s).
All Rights Reserved
Printed and manufactured in the United States of America.  2.0061802
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.
ROUNTEIS RETURNING ERROR CODES

Some of the routines in the Unix library return error codes if the call is not successful. The perror, gerror and ierror 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:

<table>
<thead>
<tr>
<th>Library</th>
<th>Windows</th>
<th>Mac Classic</th>
<th>Max OS X</th>
<th>Linux</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unix</td>
<td>unix.lib</td>
<td>unixlib.o</td>
<td>libU77.a</td>
<td>libU77.a</td>
</tr>
<tr>
<td>VMS</td>
<td>vms.lib</td>
<td>vmslib.o</td>
<td>libV77.a</td>
<td>libV77.a</td>
</tr>
</tbody>
</table>

EXAMPLE USING THE UNIX LIBRARY

As an example, this small program calls the \texttt{sleep} function that is in the Unix library:

\begin{verbatim}
WRITE(*,*) "Sleeping for a second..."
CALL sleep(1)
WRITE(*,*) "Awake again!"
END
\end{verbatim}

It can be compiled with the following command line:

\begin{verbatim}
f77 -N109 sleep.f unix.lib
\end{verbatim}

EXAMPLE USING THE VMS LIBRARY

The VMS library has some \texttt{CHARACTER}-based time and date routines. This example calls the \texttt{date} subroutine:

\begin{verbatim}
CHARACTER*9 todays_date
CALL date(todays_date)
WRITE(*,*) "Today is ", todays_date
END
\end{verbatim}

It can be compiled with the following command line:

\begin{verbatim}
f77 -N109 today.f vms.lib
\end{verbatim}
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**

```fortran
subroutine date(string)  
character*9 string
```

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

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

**idate**

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

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

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

**mvbits**

```fortran
subroutine mvbits(source, start1, len, 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:  
```fortran
integer*4 source, middle16  
call mvbits(source, 8, 16, middle16, 0)
```
6 Support Libraries

**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
8 Support Libraries

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) ...

Absoft Fortran 77 Compatibility Libraries
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) ...
```

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
```

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
```

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
10 Support Libraries

**drand**

```fortran
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.

<table>
<thead>
<tr>
<th>flag</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>returns next random number in sequence</td>
</tr>
<tr>
<td>1</td>
<td>restart generator and return first number of sequence</td>
</tr>
<tr>
<td>other</td>
<td>seed generator with <code>flag</code> and return first number of new sequence</td>
</tr>
</tbody>
</table>

**Example:**

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

**dtime**

```fortran
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:**

```fortran
real*4 dtime
total = dtime(tarray)
```

**etime**

```fortran
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:**

```fortran
real*4 etime
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)
12 Support Libraries

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**

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

---

**free**

```fortran
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**

```fortran
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:**
```fortran
integer*4 fseek
test = fseek(1, 1000, 0)
```
The `fstat` function returns statistics about the file associated with the FORTRAN unit `lunit`. The array `iarray` is filled with the following information:

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

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

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

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:**
```fortran
integer*4 ftell, position
position = ftell(1)
```
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:
```fortran
integer*4 test, chdir
character*100 the_error
test = chdir("/bad_directory")
if (test .ne. 0) then
call gerror(the_error)
end if
```

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

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

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

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.

<table>
<thead>
<tr>
<th><em>tarray</em> index</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>seconds</td>
</tr>
<tr>
<td>2</td>
<td>minutes</td>
</tr>
<tr>
<td>3</td>
<td>hours (GMT)</td>
</tr>
<tr>
<td>4</td>
<td>day of the month</td>
</tr>
<tr>
<td>5</td>
<td>month of the year</td>
</tr>
<tr>
<td>6</td>
<td>year (0 is 1900)</td>
</tr>
<tr>
<td>7</td>
<td>day of the week</td>
</tr>
<tr>
<td>8</td>
<td>day of the year</td>
</tr>
<tr>
<td>9</td>
<td>1 if DST is in effect</td>
</tr>
</tbody>
</table>
Support Libraries

See also \texttt{ctime(3)}, the \texttt{ltime} function and the \texttt{time} function.

Example: 
\begin{verbatim}
integer tarray(9)
call gmtime(670000000, tarray)
write(*,*) "Year written is: ", 1900 + tarray(6)
\end{verbatim}

---

\textbf{hostnm} \hspace{1cm} integer*4 function hostnm(name)
character*(*) name

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

Example: 
\begin{verbatim}
integer*4 test, hostnm
character*100 string
test = hostnm(string)
write(*,*) "The host name is: ", string
\end{verbatim}

---

\textbf{iargc} \hspace{1cm} integer*4 function iargc()

The \texttt{iargc} function returns the number of arguments on the command line minus one. For the following command line,

\begin{verbatim}
a.out first second third
\end{verbatim}

the value returned by \texttt{iargc} is 3. To get the arguments themselves, use the \texttt{getarg} function.

Example: 
\begin{verbatim}
integer*4 iargc
write(*,*) "Number of arguments: ", iargc()
\end{verbatim}

---

\textbf{idate} \hspace{1cm} subroutine idate(iarray)
integer*4 iarray(3)

The \texttt{idate} subroutine fills the array \texttt{iarray} with the following values:

\begin{verbatim}
  iarray index  description  range
  1       day         1-31
  2      month       1-12
  3       year  1900+
\end{verbatim}

See also the \texttt{fdate} subroutine in this library and the \texttt{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

dependent 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

dependent 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:

<table>
<thead>
<tr>
<th>flag</th>
<th>meaning when .true.</th>
<th>meaning when .false.</th>
</tr>
</thead>
<tbody>
<tr>
<td>cctl</td>
<td>ACTION='PRINT'</td>
<td>ACTION='BOTH'</td>
</tr>
<tr>
<td>bzro</td>
<td>BLANK='ZERO'</td>
<td>BLANK='NULL'</td>
</tr>
<tr>
<td>apnd</td>
<td>POSITION='APPEND'</td>
<td>POSITION='ASIS'</td>
</tr>
</tbody>
</table>

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,

```plaintext
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.

```fortran
  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**

```fortran
  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.

<table>
<thead>
<tr>
<th>flag</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>returns next random number of sequence</td>
</tr>
<tr>
<td>1</td>
<td>restart generator and return first number of sequence</td>
</tr>
<tr>
<td>other</td>
<td>seed generator with flag and return first number of new sequence</td>
</tr>
</tbody>
</table>

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

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

---

**isatty**

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

---

**itime**

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

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

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

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

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

**Example:**
```fortran
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:

```fortran
external long
```

**Example:**
```fortran
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:

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

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

**Example:**
```fortran
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.

<table>
<thead>
<tr>
<th>tarray index</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>seconds</td>
</tr>
<tr>
<td>2</td>
<td>minutes</td>
</tr>
<tr>
<td>3</td>
<td>hours (local time zone)</td>
</tr>
<tr>
<td>4</td>
<td>day of the month</td>
</tr>
<tr>
<td>5</td>
<td>month of the year</td>
</tr>
<tr>
<td>6</td>
<td>year (0 is 1900)</td>
</tr>
<tr>
<td>7</td>
<td>day of the week</td>
</tr>
<tr>
<td>8</td>
<td>day of the year</td>
</tr>
<tr>
<td>9</td>
<td>1 if DST is in effect</td>
</tr>
</tbody>
</table>

See also ctime(3) and the time function.

Example:

```fortran
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:

```fortran
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:

<table>
<thead>
<tr>
<th>Array type</th>
<th>Value for size argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>integer*2</td>
<td>2</td>
</tr>
<tr>
<td>integer*4</td>
<td>4</td>
</tr>
<tr>
<td>real*4</td>
<td>4</td>
</tr>
<tr>
<td>real*8</td>
<td>8</td>
</tr>
<tr>
<td>double precision</td>
<td>8</td>
</tr>
<tr>
<td>complex*8</td>
<td>8</td>
</tr>
<tr>
<td>complex*16</td>
<td>16</td>
</tr>
<tr>
<td>double complex</td>
<td>16</td>
</tr>
<tr>
<td>character</td>
<td>length of character element</td>
</tr>
</tbody>
</table>
The user supplied *compare* routine must return an *integer*2 value as shown in this example which compares two *real*8 numbers:

```fortran
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**

```fortran
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*:

<table>
<thead>
<tr>
<th><em>flag</em></th>
<th><em>action</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>returns next random number of sequence</td>
</tr>
<tr>
<td>1</td>
<td>restart generator and return first number of sequence</td>
</tr>
<tr>
<td>other</td>
<td>seed generator with <em>flag</em> and return first number of new sequence</td>
</tr>
</tbody>
</table>

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**

```fortran
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")
```
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('11ab111lablab', 'ab')  
last = rindex('11ab111lablab', 'ab')

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)

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)

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:

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

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)
Support Libraries

**symlink**

```fortran
integer*4 function symlink(name1, name2)
integer*4 name1, name2
```

The `symlink` 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:
```fortran
integer*4 test, symlink
test = symlink("test_file", "new_file")
```

**system**

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

**tclose**

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

**time**

```fortran
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:
```fortran
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:
```fortran
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:
```fortran
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:
```fortran
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:
```fortran
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:

```fortran
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:

```fortran
character*100 name
name = ttynam(1)
```
**twrite**

```fortran
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:
```fortran
integer test, twrite
class*1024 buffer
test = twrite(0, buffer)
```

**unlink**

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

**wait**

```fortran
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:
```fortran
integer*4 test, wait, status
test = unlink(status)
```
A
abort function ..............................................7
access function ..............................................7
alarm function ..............................................7
arguments, command line .............................15
arguments, number of ....................................18
B
bic subroutine ..............................................8
bis subroutine ..............................................8
bit function ..................................................8
blanks, finding last ......................................21
C
changing current directory .............................8
chdir function ..............................................8
chmod function ..............................................9
clearing a bit ..............................................8
command line arguments ...............................15
compiler options
- N109 option, case fold .....................................3
- N3 option, record length ...............................3
- N51 option, 32 bit RECL ..................................3
- s option, static storage ...................................3
compiling with the Unix library .........................4
compiling with the VMS library .........................4
conventions
notation in manual .........................................1
time function ..............................................9
D
date
as VMS integers .............................................5
as VMS string .................................................5
current in array ............................................18
date subroutine .............................................4
date VMS function .........................................5
deleting files ...............................................30
dflmax function ............................................9
dflmin function ............................................9
directory, changing current .........................8
directory, getting current ...............................16
drand function ............................................10
dtime function ............................................10
E
environment variables .................................16, 19
ePOCH ............................................................28
error code ....................................................3
error number, system .................................19
errors, getting text ......................................15
errors, printing text .....................................23
etime function ............................................10
examples
notation .......................................................1
executing system command ..........................27
exit subroutine ............................................11
F
fdate subroutine ...........................................11
fgetc function .............................................15
file descriptor, getting ...............................16
file permissions
determining with access ..............................7
setting with chmod ......................................9
file statistics ..............................................14, 22, 27
flmax function ............................................12
flmin function ............................................12
flush subroutine ..........................................12
fork function ..............................................12
fputc function .............................................13
free subroutine ...........................................13
fseek function .............................................13
fstat function .............................................14
ftell function .............................................14
G
gerror subroutine ..........................................3, 15
getarg subroutine .........................................3, 15
getc function .............................................15
getcwd function ..........................................16
getenv subroutine .........................................16
getfd function .............................................16
getgid function ...........................................17
getlog subroutine .........................................16
getpid function ...........................................17
gmtime subroutine .........................................17
gmtime subroutine .........................................17
group ID, getting ..........................................17
H
handler, setting signal ................................26
hostnm function ..........................................18

Absoft Fortran 77 Compatibility Libraries
Absoft Fortran 77 Compatibility Libraries

I
iargc function ........................................ 18
idate subroutine ................................... 18
idate VMS function ................................ 5
ierrno function ............................................. 3, 19
incompatible routines ....................................... 3
index function ........................................ 25
inmax function ........................................ 19
iopenit function ................................ 19
irand function ........................................ 20
isatty function ..................................... 20
itime subroutine ................................... 20
putc function .......................................... 24
qsort subroutine ..................................... 24
ran VMS function ..................................... 6
rand function ........................................ 25
random numbers
  double precision .................................. 10
  integer ............................................. 20
  single precision ................................ 25
VMS routine ........................................... 6
removing files ........................................ 30
rename function ..................................... 25
rindex function ..................................... 25
routine incompatibility .................................. 3

S
secnds VMS function ................................ 6
sending messages ..................................... 21
setbit subroutine ..................................... 26
settin a bit ............................................ 8
short function ........................................ 26
signal function ..................................... 26
signal, waiting for ................................ 31
sleep function ........................................ 4
sleep subroutine ....................................... 26
sorting array data ..................................... 24
stat function .......................................... 27
Sun FORTRAN ........................................... 1
support libraries ....................................... 1
symbolic linking ....................................... 27
symlnk function ..................................... 27
syntax
  notation in manual .................................. 1
  system function ..................................... 27

T
tape, opening ........................................ 28
tclose function ...................................... 28
terminal device ....................................... 20, 30
time
  as VMS string ..................................... 6
current time as array ................................ 20
current time as string .............................. 11
elapsed ................................................. 10
elapsed since midnight ............................ 6
GMT relative ......................................... 17
sleeping ................................................. 26
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>system time as array</td>
<td>22</td>
</tr>
<tr>
<td>system time as string</td>
<td>9</td>
</tr>
<tr>
<td>VMS function</td>
<td>6</td>
</tr>
<tr>
<td>time function</td>
<td>28</td>
</tr>
<tr>
<td>topen function</td>
<td>28</td>
</tr>
<tr>
<td>tread function</td>
<td>28</td>
</tr>
<tr>
<td>trewin function</td>
<td>29</td>
</tr>
<tr>
<td>tskipf function</td>
<td>29</td>
</tr>
<tr>
<td>tstate function</td>
<td>29</td>
</tr>
<tr>
<td>ttynam function</td>
<td>30</td>
</tr>
<tr>
<td>twrite function</td>
<td>30</td>
</tr>
<tr>
<td>Unix library</td>
<td>1</td>
</tr>
<tr>
<td>unlink function</td>
<td>30</td>
</tr>
<tr>
<td>user ID, getting</td>
<td>17</td>
</tr>
<tr>
<td>VAX FORTRAN</td>
<td>1</td>
</tr>
<tr>
<td>VMS library</td>
<td>1</td>
</tr>
<tr>
<td>wait function</td>
<td>31</td>
</tr>
<tr>
<td>waiting for an alarm</td>
<td>7</td>
</tr>
</tbody>
</table>