

# **f90\_unix\_env: Unix Environment Functions Module**

July 25, 2022

## **1 Name**

`f90_unix_env` — Module of Unix environment functions

## **2 Usage**

### **USE F90\_UNIX\_ENV**

This module contains part of a Fortran API to functions detailed in ISO/IEC 9945-1:1990 Portable Operating System Interface (POSIX) - Part 1: System Application Program Interface (API) [C Language].

The functions in this module are from Section 4: Process Environment, plus `gethostname` from 4.3BSD.

Error handling is described in `F90_UNIX_ERRNO`. Note that for procedures with an optional `ERRNO` argument, if an error occurs and `ERRNO` is not present, the program will be terminated.

All the procedures in this module are generic; some are also specific (this may change in a future release).

## **3 Synopsis**

### **Parameters**

`CLOCK_TICK_KIND`, `LONG_KIND`, `SC_STDIN_UNIT`, `SC_STDOUT_UNIT`, `SC_STDERR_UNIT`, `SC_ARG_MAX`,  
`SC_CHILD_MAX`, `SC_CLK_TCK`, `SC_NGROUPS_MAX`, `SC_SAVED_IDS`, `SC_VERSION`, `TIME_KIND`.

### **Derived Types**

`TMS`, `UTSNAME`.

### **Generic Procedures**

`CLK_TCK`, `CTERMID`, `GETARG`, `GETEGID`, `GETENV`, `GETEUID`, `GETGID`, `GETGROUPS`, `GETLOGIN`, `GETPGRP`,  
`GETPID`, `GETPPID`, `GETUID`, `IARGC`, `ISATTY`, `SETGID`, `SETPGID`, `SETSID`, `SETUID`, `SYSCONF`, `TIME`,  
`TIMES`, `TTYNAME`, `UNAME`.

Note that many of these procedures are also specific; this may change in a future release.

## **4 Parameter Description**

`INTEGER(int32),PARAMETER :: clock_tick_kind`

The integer kind used for clock ticks (see `TIMES`).

`INTEGER(int32),PARAMETER :: id_kind`

Integer kind for representing IDs; that is, users (uids), groups (gids), process groups (pgids) and processes (pids).

```
INTEGER(int32),PARAMETER :: long_kind
```

The integer kind corresponding to the C type ‘long’. This is not actually used in this module, but is suitable for use when declaring a variable to receive a value from `SYSCONF`.

```
INTEGER(int32),PARAMETER :: sc_stdin_unit, sc_stdout_unit, sc_stderr_unit,  
    sc_arg_max, sc_child_max, sc_clk_tck, sc_job_control, sc_open_max,  
    sc_ngroups_max, sc_saved_ids, sc_stream_max, sc_tzname_max, sc_version
```

Values used as arguments to `SYSCONF`. The following table describes the returned information from `SYSCONF`, this is not the value of the `SC_*` constant.

#### `SC_STDIN_UNIT`

The logical unit number for standard input (`READ *,...`); this is the same value as `INPUT_UNIT` in the standard intrinsic module `ISO_FORTRAN_ENV`.

#### `SC_STDOUT_UNIT`

The logical unit number for standard output (`PRINT, WRITE(*,...)`); this is the same value as `OUTPUT_UNIT` in the standard intrinsic module `ISO_FORTRAN_ENV`.

#### `SC_STDERR_UNIT`

The logical unit number on which errors are reported; this is the same value as `ERROR_UNIT` in the standard intrinsic module `ISO_FORTRAN_ENV`.

#### `SC_ARG_MAX`

Maximum length of arguments for the `EXEC` functions, in bytes, including environment data.

#### `SC_CHILD_MAX`

Maximum number of simultaneous processes for a single user.

#### `SC_CLK_TCK`

Number of clock ticks per second. (This is the same value returned by the `CLK_TCK` function).

#### `SC_JOB_CONTROL`

Value available only if job control is supported by the operating system.

#### `SC_NGROUPS_MAX`

Maximum number of simultaneous supplementary group IDs per process.

#### `SC_OPEN_MAX`

Maximum number of files open simultaneously by a single process.

#### `SC_SAVED_IDS`

Value available only if each process has a saved set-uid and set-gid.

#### `SC_STREAM_MAX`

Maximum number of logical units that can be simultaneously open. Not always available.

#### `SC_TZNAME_MAX`

Maximum number of characters for the name of a time zone.

#### `SC_VERSION`

Posix version number. This will be 199009 if the underlying operating system’s C interface conforms to ISO/IEC 9945-1:1990.

```
INTEGER(int32),PARAMETER :: time_kind
```

The integer kind used for holding date/time values (see `TIME`).

## 5 Derived-Type Descriptions

```
TYPE tms
    INTEGER(clock_tick_kind) utime, stime, ctime, cstime
END TYPE
```

Derived type holding CPU usage time in clock ticks. **UTIME** and **STIME** contain CPU time information for a process, **CUTIME** and **CSTIME** contain CPU time information for its terminated child processes. In each case this is divided into user time (**UTIME**, **CUTIME**) and system time (**STIME**, **CSTIME**).

```
TYPE utsname
    CHARACTER(...) sysname, nodename, release, version, machine
END TYPE
```

Derived type holding data returned by **UNAME**. Note that the character length of each component is fixed, but may be different on different systems. The values in these components are blank-padded (if short) or truncated (if long). For further information see ISO/IEC 9945-1:1990.

## 6 Procedure Descriptions

```
PURE INTEGER(KIND=clock_tick_kind) FUNCTION clk_tck()
```

Returns the number of clock ticks in one second of CPU time (see **TIMES**).

```
PURE SUBROUTINE ctermid(s,lens)
    CHARACTER(*),OPTIONAL,INTENT(OUT) :: s
    INTEGER(int32),OPTIONAL,INTENT(OUT) :: lens
```

If present, **S** is set to the filename of the controlling terminal. If present, **LENS** is set to the length of the filename of the controlling terminal. If **S** is longer than the filename of the controlling terminal it is padded with blanks; if **S** is shorter it is truncated — it is the user's responsibility to check the value of **LENS** to detect such truncation.

If the filename of the controlling terminal cannot be determined for any reason **LENS** (if present) will be set to zero and **S** (if present) will be blank.

```
SUBROUTINE getarg(k,arg,lenarg,errno)
    INTEGER(*),INTENT(IN) :: k
    CHARACTER(*),OPTIONAL,INTENT(OUT) :: arg
    INTEGER(int32),OPTIONAL,INTENT(OUT) :: lenarg
    INTEGER(error_kind),OPTIONAL,INTENT(OUT) :: errno
```

Accesses command-line argument number **K**, where argument zero is the program name. If **ARG** is present, it receives the argument text (blank-padded or truncated as appropriate if the length of the argument differs from that of **ARG**). If **LENARG** is present, it receives the length of the argument. If **ERRNO** is present, it receives the error status.

Note that if **K** is less than zero or greater than the number of arguments (returned by **IARGC**) error **EINVAL** (see **F90\_UNIX\_ERRNO**) is raised.

This procedure is very similar to the standard intrinsic procedure **GET\_COMMAND\_ARGUMENT**.

```
PURE INTEGER(id_kind) FUNCTION getegid()
```

Returns the effective group number of the calling process.

```
SUBROUTINE getenv(name,value,lenvalue,errno)
  CHARACTER(*),INTENT(IN) :: name
  CHARACTER(*),OPTIONAL,INTENT(OUT) :: value
  INTEGER(int32),OPTIONAL,INTENT(OUT) :: lenvalue
  INTEGER(error_kind),OPTIONAL,INTENT(OUT) :: errno
```

Accesses the environment variable named by NAME. If VALUE is present, it receives the text value of the variable (blank-padded or truncated as appropriate if the length of the value differs from that of VALUE). If LENVALUE is present, it receives the length of the value. If ERRNO is present, it receives the error status.

If there is no such variable, error EINVAL (see F90\_UNIX\_ERRNO) is raised. Other possible errors include ENOMEM.

```
PURE INTEGER(id_kind) FUNCTION geteuid()
```

Returns the effective user number of the calling process.

```
PURE INTEGER(id_kind) FUNCTION getgid()
```

Returns the group number of the calling process.

```
SUBROUTINE getgroups(grouplist,ngroups,errno)
  INTEGER(id_kind),OPTIONAL,INTENT(OUT) :: grouplist(:)
  INTEGER(int32),OPTIONAL,INTENT(OUT) :: ngroups
  INTEGER(error_kind),OPTIONAL,INTENT(OUT) :: errno
```

Retrieves supplementary group number information for the calling process. If GROUPLIST is present, it is filled with the supplementary group numbers. If NGROUPS is present, it receives the number of supplementary group numbers. If ERRNO is present, it receives the error status.

If GROUPLIST is present but too small to contain the complete list of supplementary group numbers, error EINVAL (see F90\_UNIX\_ERRNO) is raised. The maximum number of supplementary group numbers can be found using SYSCONF (enquiry SC\_NGROUPS\_MAX); alternatively, 'CALL GETGROUPS(NGROUPS=N)' will reliably return the actual number in use.

```
PURE SUBROUTINE gethostname(name,lenname)
  CHARACTER(*),OPTIONAL,INTENT(OUT) :: name
  INTEGER(int32),OPTIONAL,INTENT(OUT) :: lenname
```

This provides the functionality of 4.3BSD's *gethostname*. If NAME is present it receives the text of the standard host name for the current processor, blank-padded or truncated if appropriate. If LENNAME is present it receives the length of the host name. If no host name is available LENNAME will be zero.

```
PURE SUBROUTINE getlogin(s,lens)
  CHARACTER(*),OPTIONAL,INTENT(OUT) :: s
  INTEGER(int32),OPTIONAL,INTENT(OUT) :: lens
```

Accesses the user name (login name) associated with the calling process. If **S** is present, it receives the text of the name (blank-padded or truncated as appropriate if the length of the login name differs from that of **S**). If **LENS** is present, it receives the length of the login name.

```
PURE INTEGER(id_kind) FUNCTION getpgrp()
```

Returns the process group number of the calling process.

```
PURE INTEGER(id_kind) FUNCTION getpid()
```

Returns the process number of the calling process.

```
PURE INTEGER(id_kind) FUNCTION getppid()
```

Returns the process number of the parent of the calling process.

```
PURE INTEGER(id_kind) FUNCTION getuid()
```

Returns the user number of the calling process.

```
PURE INTEGER(int32) FUNCTION iargc()
```

Returns the number of command-line arguments; this is the same value as the intrinsic function **COMMAND\_ARGUMENT\_COUNT**, except that it returns -1 if even the program name is unavailable (the intrinsic function erroneously returns the same value, 0, whether the program name is available or not).

```
SUBROUTINE isatty(lunit,answer,errno)
  INTEGER(*),INTENT(IN) :: lunit
  LOGICAL(*),INTENT(OUT) :: answer
  INTEGER(error_kind),OPTIONAL,INTENT(OUT) :: errno
```

**ANSWER** receives the value **.TRUE.** if and only if the logical unit identified by **LUNIT** is connected to a terminal.

If **LUNIT** is not a valid unit number or is not connected to any file, error **EBADF** (see **F90\_UNIX\_ERRNO**) is raised.

```
SUBROUTINE setgid(gid,errno)
  INTEGER(*),INTENT(IN) :: gid
  INTEGER(error_kind),OPTIONAL,INTENT(OUT) :: errno
```

Sets the group number of the calling process to **GID**. For full details refer to section 4.2.2 of ISO/IEC 9945-1:1990.

If **GID** is not a valid group number, error **EINVAL** (see **F90\_UNIX\_ERRNO**) is raised. If the process is not allowed to set the group number to **GID**, error **EPERM** is raised.

```
SUBROUTINE setpgid(pid,pgid,errno)
  INTEGER(*),INTENT(IN) :: pid, pgid
  INTEGER(error_kind),OPTIONAL,INTENT(OUT) :: errno
```

Sets the process group number of process PID (or, if PID is zero, the calling process) to PGID. For full details refer to section 4.3.3 of ISO/IEC 9945-1:1990.

Possible errors include EACCES, EINVAL, ENOSYS, EPERM, ESRCH (see F90\_UNIX\_ERRNO).

```
SUBROUTINE setsid(sid,errno)
  INTEGER(id_kind),OPTIONAL,INTENT(OUT) :: sid
  INTEGER(error_kind),OPTIONAL,INTENT(OUT) :: errno
```

Creates a session and sets the process group number of the calling process. For full details refer to section 4.3.2 of ISO/IEC 9945-1:1990. If SID is present it receives the new session id (equal to the process id); if an error occurs it receives -1.

Possible errors include EPERM (see F90\_UNIX\_ERRNO).

```
SUBROUTINE setuid(uid,errno)
  INTEGER(*),INTENT(IN) :: uid
  INTEGER(error_kind),OPTIONAL,INTENT(OUT) :: errno
```

Sets the user number of the calling process to UID. For full details refer to section 4.2.2 of ISO/IEC 9945-1:1990.

If UID is not a valid group number, error EINVAL (see F90\_UNIX\_ERRNO) is raised. If the process is not allowed to set the user number to UID, error EPERM is raised.

```
SUBROUTINE sysconf(name,val,errno)
  INTEGER(*),INTENT(IN) :: name
  INTEGER(*),INTENT(OUT) :: val
  INTEGER(error_kind),OPTIONAL,INTENT(OUT) :: errno
```

Returns the value of a system configuration variable. The variables are named by integer PARAMETERS defined in this module, and are described in the **Parameters** section.

If NAME is not a valid configuration variable name, error EINVAL (see F90\_UNIX\_ERRNO) is raised. If VAL is too small an integer kind to contain the result, error ERANGE is raised; kind LONG\_KIND is guaranteed to be big enough for all values.

```
SUBROUTINE time(itime,errno)
  INTEGER(time_kind),INTENT(OUT) :: itime
  INTEGER(error_kind),OPTIONAL,INTENT(OUT) :: errno
```

ITIME receives the operating system date/time in seconds since the Epoch.

```
INTEGER(KIND=clock_tick_kind) FUNCTION times(buffer)
  TYPE(tms),INTENT(OUT) :: buffer
```

This function returns the elapsed real time in clock ticks since an arbitrary point in the past, or -1 if the function is unavailable. BUFFER is filled in with CPU time information for the calling process and any terminated child processes.

If this function returns zero the values in BUFFER will still be correct but the elapsed-time timer was not available.

```

SUBROUTINE ttyname(lunit,s,lens,errno)
  INTEGER(*),INTENT(IN) :: lunit
  CHARACTER(*),OPTIONAL,INTENT(OUT) :: s
  INTEGER(int32),OPTIONAL,INTENT(OUT) :: lens
  INTEGER(error_kind),OPTIONAL,INTENT(OUT) :: errno

```

Accesses the name of the terminal connected to the logical unit identified by **LUNIT**. If **S** is present, it receives the text of the terminal name (blank-padded or truncated as appropriate, if the length of the terminal name differs from that of **S**). If **LENS** is present, it receives the length of the terminal name. If **ERRNO** is present, it receives the error status.

If **LUNIT** is not a valid logical unit number, or is not connected, error **EBADF** (see **F90.UNIX\_ERRNO**) is raised; otherwise, if the function is not available, **ENOSYS** is raised, or if **LUNIT** is not connected to a terminal, error **ENOTTY** is raised.

```

SUBROUTINE uname(name,errno)
  TYPE(UTSNAME),INTENT(OUT) :: name
  INTEGER(error_kind),OPTIONAL,INTENT(OUT) :: errno

```

Returns information about the operating system in **NAME**.

## 7 See Also

**f90\_unix\_errno(3)**, **intro(3)**, **nag\_modules(3)**, **nagfor(1)**.

## 8 Bugs

Please report any bugs found to ‘support@nag.co.uk’ or ‘support@nag.com’, along with any suggestions for improvements.