

**dos**

**COLLABORATORS**

|               |                       |                |                  |
|---------------|-----------------------|----------------|------------------|
|               | <i>TITLE :</i><br>dos |                |                  |
| <i>ACTION</i> | <i>NAME</i>           | <i>DATE</i>    | <i>SIGNATURE</i> |
| WRITTEN BY    |                       | March 28, 2025 |                  |

**REVISION HISTORY**

| NUMBER | DATE | DESCRIPTION | NAME |
|--------|------|-------------|------|
|        |      |             |      |

# Contents

|          |                           |          |
|----------|---------------------------|----------|
| <b>1</b> | <b>dos</b>                | <b>1</b> |
| 1.1      | dos.doc                   | 1        |
| 1.2      | dos.library/Close         | 1        |
| 1.3      | dos.library/CreateDir     | 2        |
| 1.4      | dos.library/CreateProc    | 2        |
| 1.5      | dos.library/CurrentDir    | 3        |
| 1.6      | dos.library/DateStamp     | 3        |
| 1.7      | dos.library/Delay         | 4        |
| 1.8      | dos.library/DeleteFile    | 4        |
| 1.9      | dos.library/DeviceProc    | 5        |
| 1.10     | dos.library/DupLock       | 5        |
| 1.11     | dos.library/Examine       | 6        |
| 1.12     | dos.library/Execute       | 6        |
| 1.13     | dos.library/Exit          | 7        |
| 1.14     | dos.library/ExNext        | 8        |
| 1.15     | dos.library/Info          | 9        |
| 1.16     | dos.library/IoErr         | 10       |
| 1.17     | dos.library/Input         | 10       |
| 1.18     | dos.library/IsInteractive | 10       |
| 1.19     | dos.library/LoadSeg       | 11       |
| 1.20     | dos.library/Lock          | 12       |
| 1.21     | dos.library/Open          | 12       |
| 1.22     | dos.library/Output        | 13       |
| 1.23     | dos.library/ParentDir     | 13       |
| 1.24     | dos.library/Read          | 14       |
| 1.25     | dos.library/Rename        | 14       |
| 1.26     | dos.library/Seek          | 15       |
| 1.27     | dos.library/SetComment    | 16       |
| 1.28     | dos.library/SetProtection | 16       |
| 1.29     | dos.library/UnLoadSeg     | 17       |
| 1.30     | dos.library/UnLock        | 17       |
| 1.31     | dos.library/WaitForChar   | 18       |
| 1.32     | dos.library/Write         | 18       |

---

# Chapter 1

## dos

### 1.1 dos.doc

|               |                  |                  |
|---------------|------------------|------------------|
| Close ()      | Exit ()          | Read ()          |
| CreateDir ()  | ExNext ()        | Rename ()        |
| CreateProc () | Info ()          | Seek ()          |
| CurrentDir () | Input ()         | SetComment ()    |
| DateStamp ()  | IoErr ()         | SetProtection () |
| Delay ()      | IsInteractive () | UnLoadSeg ()     |
| DeleteFile () | LoadSeg ()       | UnLock ()        |
| DeviceProc () | Lock ()          | WaitForChar ()   |
| DupLock ()    | Open ()          | Write ()         |
| Examine ()    | Output ()        |                  |
| Execute ()    | ParentDir ()     |                  |

### 1.2 dos.library/Close

#### NAME

Close -- Close an open file

#### SYNOPSIS

```
Close( file )  
    D1
```

```
struct FileHandle *file;
```

#### FUNCTION

The file specified by the file handle is closed. You must close all files you explicitly opened, but you must not close inherited file handles that are passed to you (each filehandle must be closed once and ONLY once).

#### INPUTS

file - BCPL pointer to a file handle

#### SEE ALSO

Open

---

## 1.3 dos.library/CreateDir

### NAME

CreateDir -- Create a new directory

### SYNOPSIS

```
lock = CreateDir( name )
D0          D1
```

```
struct FileLock *lock;
char *name;
```

### FUNCTION

CreateDir creates a new directory with the specified name. An error is returned if it fails. Directories can only be created on devices which support them, e.g. disks. A return of zero means that AmigaDOS has found an error you should then call IoErr() to find out more; otherwise, CreateDir returns an exclusive lock on the new directory.

### INPUTS

name - pointer to a null-terminated string

### OUTPUTS

lock - BCPL pointer to a lock

## 1.4 dos.library/CreateProc

### NAME

CreateProc -- Create a new process

### SYNOPSIS

```
process = CreateProc( name, pri, segment, stackSize )
D0          D1      D2      D3      D4
```

```
struct Process *process;
char *name;
LONG pri, stackSize;
BPTR *segment;
```

### FUNCTION

CreateProc creates a new AmigaDOS process of name 'name'. AmigaDOS processes are a superset of exec tasks.

A segment list, as returned by LoadSeg(), is passed as 'seglist'. This represents a section of code which is to be run as a new process. The code is entered at the first hunk in the segment list, which should contain suitable initialization code or a jump to such. A process control structure is allocated from memory and initialized. If you wish to fake a segment list (that will never have DOS UnLoadSeg() called on it), use this code:

```
ds.l      0      ;Align to longword
DC.L     16     ;Segment "length" (faked)
```

```
DC.L    0    ;Pointer to next segment
...start of code...
```

The size of the root stack upon activation is passed as 'stackSize'. 'pri' specifies the required priority of the new process. The result will be the process identifier of the new process, or zero if the routine failed. The argument 'name' specifies the new process name. A zero return code indicates error.

#### INPUTS

```
name - pointer to a null-terminated string
pri - signed integer
segment - BCPL pointer to a segment
stackSize - integer (must be a multiple of 4 bytes)
```

#### OUTPUTS

```
process - process identifier
```

## 1.5 dos.library/CurrentDir

#### NAME

```
CurrentDir -- Make a directory associated with a lock the working
              directory
```

#### SYNOPSIS

```
oldLock = CurrentDir( lock )
D0          D1

struct FileLock *oldlock, *lock;
```

#### FUNCTION

CurrentDir() causes a directory associated with a lock to be made the current directory. The old current directory lock is returned.

A value of zero is a valid result here, this 0 lock represents the root of file system that you booted from (which is, in effect, the parent of all other file system roots.)

#### INPUTS

```
lock - BCPL pointer to a lock
```

#### OUTPUTS

```
oldLock - BCPL pointer to a lock
```

#### SEE ALSO

```
Lock
```

## 1.6 dos.library/DateStamp

#### NAME

```
DateStamp -- Obtain the date and time in internal format
```

## SYNOPSIS

```
DateStamp( v );  
          D1
```

```
LONG *v;
```

## FUNCTION

DateStamp() takes a vector of three longwords that is set to the current time. The first element in the vector is a count of the number of days. The second element is the number of minutes elapsed in the day. The third is the number of ticks elapsed in the current minute. A tick happens 50 times a second. DateStamp ensures that the day and minute are consistent. All three elements are zero if the date is unset. DateStamp() currently only returns even multiples of 50 ticks. Therefore the time you get is always an even number of ticks.

## INPUTS

v - pointer to an array of three longwords

## OUTPUTS

The array is filled as described.

## 1.7 dos.library/Delay

## NAME

Delay -- Delay a process for a specified time

## SYNOPSIS

```
Delay( ticks )  
      D1
```

```
LONG ticks;
```

## FUNCTION

The argument 'ticks' specifies how many ticks (50 per second) to wait before returning control.

## BUGS

Due to a bug in the timer.device in V1.2/V1.3, specifying a timeout of zero for Delay() can cause the unreliable timer & floppy disk operation.

## INPUTS

ticks - integer

## 1.8 dos.library/DeleteFile

## NAME

DeleteFile -- Delete a file or directory

---

## SYNOPSIS

```
success = DeleteFile( name )  
D0                      D1
```

```
BOOL success;  
char *name;
```

## FUNCTION

This attempts to delete the file or directory specified by 'name'. An error is returned if the deletion fails. Note that all the files within a directory must be deleted before the directory itself can be deleted.

## INPUTS

name - pointer to a null-terminated string

## OUTPUTS

success - boolean

## SEE ALSO

IoErr

## 1.9 dos.library/DeviceProc

## NAME

DeviceProc -- Return the process I.D. of specific I/O handler

## SYNOPSIS

```
process = DeviceProc( name )  
D0                      D1
```

## FUNCTION

DeviceProc() returns the process identifier of the process which handles the device associated with the specified name. If no process handler can be found then the result is zero. If the name refers to a file on a mounted device then a pointer to a directory lock is returned in IoErr().

## 1.10 dos.library/DupLock

## NAME

DupLock -- Duplicate a lock

## SYNOPSIS

```
lock = DupLock( lock )  
D0                      D1
```

```
struct FileLock *newlock, *lock;
```

## FUNCTION

DupLock() is passed a shared filing system lock. This is the ONLY way to obtain a duplicate of a lock... simply copying is not

allowed.

Another lock to the same object is then returned. It is not possible to create a copy of a write lock.

A zero return indicates failure.

#### INPUTS

lock - BCPL pointer to a lock

#### OUTPUTS

newLock - BCPL pointer to a lock

#### SEE ALSO

Lock()

## 1.11 dos.library/Examine

#### NAME

Examine -- Examine a directory or file associated with a lock

#### SYNOPSIS

```
success = Examine( lock, infoBlock )
D0          D1    D2
```

```
BOOL success;
struct FileLock *lock;
struct FileInfoBlock *infoBlock
```

#### FUNCTION

Examine() fills in information in the FileInfoBlock concerning the file or directory associated with the lock. This information includes the name, size, creation date and whether it is a file or directory. FileInfoBlock must be longword aligned. Examine() gives a return code of zero if it fails.

You may make a local copy of the FileInfoBlock, as long as it is never passed back to the operating system.

#### INPUTS

lock - BCPL pointer to a lock  
infoBlock - pointer to a FileInfoBlock (must be longword aligned)

#### OUTPUTS

success - boolean

## 1.12 dos.library/Execute

#### NAME

Execute -- Execute a CLI command

#### SYNOPSIS

---

```
success = Execute( commandString, input, output )
D0          D1          D2          D3
```

```
BOOL success
char *commandString;
struct FileHandle *input, *output;
```

#### FUNCTION

This function attempts to execute the string `commandString` as though it were a CLI command and arguments. The string can contain any valid input that you could type directly in a CLI, including input and output redirection using `<` and `>`.

The input file handle will normally be zero, and in this case `Execute()` will perform whatever was requested in the `commandString` and then return. If the input file handle is nonzero then after the (possibly null) `commandString` is performed subsequent input is read from the specified input file handle until end of that file is reached.

In most cases the output file handle must be provided, and is used by the CLI commands as their output stream unless output redirection was specified. If the output file handle is set to zero then the current window, normally specified as `*`, is used. Note that programs running under the Workbench do not normally have a current window.

`Execute()` may also be used to create a new interactive CLI process just like those created with the `NEWCLI` function. In order to do this you would call `Execute()` with an empty `commandString`, and pass a file handle relating to a new window as the input file handle. The output file handle would be set to zero. The CLI will read commands from the new window, and will use the same window for output. This new CLI window can only be terminated by using the `ENDCLI` command.

For this command to work the program `RUN` must be present in `C:.`

#### INPUTS

```
commandString - pointer to a null-terminated string
input - BCPL pointer to a file handle
output - BCPL pointer to a file handle
```

#### OUTPUTS

```
success - BOOLEAN indicating whether Execute was successful
          in finding and starting the specified program
```

## 1.13 dos.library/Exit

#### NAME

```
Exit -- Exit from a program
```

#### SYNOPSIS

```
Exit( returnCode )
D1
```

```
LONG returnCode;
```

#### FUNCTION

Exit() is currently for use with programs written as if they were BCPL programs. This function is not normally useful for other purposes.

In general, therefore, please DO NOT CALL THIS FUNCTION!

In order to exit, C programs should use the C language exit() function (note the lower case letter "e"). Assembly programs should place a return code in D0, and execute an RTS instruction.

#### IMPLEMENTATION

The action of Exit() depends on whether the program which called it is running as a command under a CLI or not. If the program is running under the CLI the command finishes and control reverts to the CLI. In this case, returnCode is interpreted as the return code from the program.

If the program is running as a distinct process, Exit() deletes the process and release the space associated with the stack, segment list and process structure.

#### INPUTS

returnCode - integer

## 1.14 dos.library/ExNext

#### NAME

ExNext -- Examine the next entry in a directory

#### SYNOPSIS

```
success = ExNext( lock, infoBlock )
D0          D1    D2
```

```
BOOL success;
struct FileLock *lock;
struct FileInfoBlock *infoBlock;
```

#### FUNCTION

This routine is passed a directory lock and a FileInfoBlock that have been initialized by a previous call to Examine(), or updated by a previous call to ExNext. ExNext gives a return code of zero on failure. The most common cause of failure is reaching the end of the list of files in the owning directory. In this case, IoErr will return ERROR\_NO\_MORE\_ENTRIES and a good exit is appropriate.

So, follow these steps to examine a directory:

- 1) Pass a Lock and a FileInfoBlock to Examine(). The Lock must be on the directory you wish to examine.
- 2) Pass ExNext the same Lock and FileInfoBlock.
- 3) Do something with the information returned in the FileInfoBlock. Note that the type field is positive for directories, negative for files.

- 4) Keep calling ExNext until it returns FALSE. Check IoErr() to ensure that the reason for failure was ERROR\_NO\_MORE\_ENTRIES.

Note: if you wish to recursively scan the file tree and you find another directory while ExNext'ing you must Lock that directory and Examine() it using a new FileInfoBlock. Use of the same FileInfoBlock to enter a directory would lose important state information such that it will be impossible to continue scanning the parent directory. While it is permissible to UnLock and Lock the parent directory between ExNext calls, this is not recommended. Important state information is associated with the parent lock so if it is freed between ExNext calls this information has to be rebuilt on each new ExNext call and will significantly slow down directory scanning.

It is NOT legal to Examine() a file, and then to ExNext from that FileInfoBlock. You may make a local copy of the FileInfoBlock, as long as it is never passed back to the operating system.

#### INPUTS

lock - BCPL pointer to a lock originally used for the Examine() call  
infoBlock - pointer to a FileInfoBlock used on the previous Examine() or ExNext() call.

#### OUTPUTS

success - boolean

#### SPECIAL NOTE

The FileInfoBlock must be longword aligned.

## 1.15 dos.library/Info

#### NAME

Info -- Returns information about the disk

#### SYNOPSIS

```
success = Info( lock, parameterBlock )  
D0          D1    D2
```

```
struct FileLock *lock;  
struct InfoData *parameterBlock
```

#### FUNCTION

Info() can be used to find information about any disk in use. 'lock' refers to the disk, or any file on the disk. The parameter block is returned with information about the size of the disk, number of free blocks and any soft errors.

#### INPUTS

lock - BCPL pointer to a lock  
parameterBlock - pointer to an InfoData structure  
(longword aligned)

#### OUTPUTS

success - boolean

---

**SPECIAL NOTE:**

Note that InfoData structure must be longword aligned.

## 1.16 dos.library/loErr

**NAME**

IoErr -- Return extra information from the system

**SYNOPSIS**

```
error = IoErr()  
DO
```

```
LONG error;
```

**FUNCTION**

I/O routines return zero to indicate an error. When this happens, this routine may be called to determine more information. It is also used in some routines to pass back a secondary result.

**OUTPUTS**

error - integer

**SEE ALSO**

Open, Read, ExNext

## 1.17 dos.library/Input

**NAME**

Input -- Identify the program's initial input file handle

**SYNOPSIS**

```
file = Input()  
DO
```

```
struct FileHandle *file;
```

**FUNCTION**

Input() is used to identify the initial input stream allocated when the program was initiated.

**OUTPUTS**

file - BCPL pointer to a file handle

**SEE ALSO**

Output()

## 1.18 dos.library/IsInteractive

---

## NAME

IsInteractive -- Discover whether a file is a virtual terminal

## SYNOPSIS

```
status = IsInteractive( file )
```

```
D0                                D1
```

```
BOOL status;  
struct FileHandle *file;
```

## FUNCTION

The return value 'status' indicates whether the file associated with the file handle 'file' is connected to a virtual terminal.

## INPUTS

file - BCPL pointer to a file handle

## OUTPUTS

status - boolean

## 1.19 dos.library/LoadSeg

## NAME

LoadSeg -- Load a load module into memory

## SYNOPSIS

```
segment = LoadSeg( name )
```

```
D0                                D1
```

```
BPTR segment;  
char *name;
```

## FUNCTION

The file 'fileName' should be a load module produced by the linker. LoadSeg scatter loads the CODE, DATA and BSS segments into memory, chaining together the segments with BPTR's on their first words. The end of the chain is indicated by a zero.

In the event of an error any blocks loaded will be unloaded and a FALSE (zero) result returned.

If the module is correctly loaded then the output will be a pointer at the beginning of the list of blocks. Loaded code is unloaded via a call to UnLoadSeg().

## INPUTS

name - pointer to a null-terminated string

## OUTPUTS

segment - BCPL pointer to a segment

---

## 1.20 dos.library/Lock

### NAME

Lock -- Lock a directory or file

### SYNOPSIS

```
lock = Lock( name, accessMode )
D0          D1      D2
```

```
struct FileLock *lock;
char *name;
LONG accessMode;
```

### FUNCTION

A filing system lock on the file or directory 'name' is returned if possible.

If the accessMode is ACCESS\_READ, the lock is a shared read lock; if the accessMode is ACCESS\_WRITE then it is an exclusive write lock.

If Lock() fails (that is, if it cannot obtain a filing system lock on the file or directory) it returns a zero. Note that the overhead for doing a Lock() is less than that for doing an Open(), so that, if you want to test to see if a file exists, you should use Lock(). Of course, once you've found that it exists, you must use Open() if you want to open it.

Tricky assumptions about the internal format of a lock are unwise.

### INPUTS

name - pointer to a null-terminated string  
accessMode - integer

### OUTPUTS

lock - BCPL pointer to a lock

## 1.21 dos.library/Open

### NAME

Open -- Open a file for input or output

### SYNOPSIS

```
file = Open( name, accessMode )
D0          D1      D2
```

```
struct FileHandle *file;
char *name;
LONG accessMode;
```

### FUNCTION

The named file is opened and a file handle returned. If the accessMode is MODE\_OLDFILE, an existing file is opened for reading or writing. If the value is MODE\_NEWFILE, a new file is created for

---

writing. `MODE_READWRITE` opens an old file with and exclusive lock. Open types are documented in the "libraries/dos.h" include file.

The 'name' can be a filename (optionally prefaced by a device name), a simple device such as `NIL:`, a window specification such as `CON:` or `RAW:` followed by window parameters, or `*`, representing the current window.

If the file cannot be opened for any reason, the value returned will be zero, and a secondary error code will be available by calling the routine `IoErr()`.

#### INPUTS

name - pointer to a null-terminated string  
accessMode - integer

#### OUTPUTS

file - BCPL pointer to a file handle

## 1.22 dos.library/Output

#### NAME

Output -- Identify the programs' initial output file handle

#### SYNOPSIS

```
file = Output()  
D0
```

```
struct FileHandle *file;
```

#### FUNCTION

`Output()` is used to identify the initial output stream allocated when the program was initiated.

#### OUTPUTS

file - BCPL pointer to a file handle

## 1.23 dos.library/ParentDir

#### NAME

ParentDir -- Obtain the parent of a directory or file

#### SYNOPSIS

```
newlock = ParentDir( lock )  
D0                D1
```

```
struct FileLock *newlock, *lock;
```

#### FUNCTION

The argument 'lock' is associated with a given file or directory. `ParentDir()` returns 'newlock' which is associated the parent directory of 'lock'.

---

Taking the ParentDir() of the root of the current filing system returns a NULL (0) lock. Note this 0 lock represents the root of file system that you booted from (which is, in effect, the parent of all other file system roots.)

**INPUTS**

lock - BCPL pointer to a lock

**OUTPUTS**

newlock - BCPL pointer to a lock

## 1.24 dos.library/Read

**NAME**

Read -- Read bytes of data from a file

**SYNOPSIS**

```
actualLength = Read( file, buffer, length )
D0              D1    D2      D3
```

```
LONG actualLength;
struct FileHandle *file;
char *buffer;
LONG length;
```

**FUNCTION**

Data can be copied using a combination of Read() and Write(). Read() reads bytes of information from an opened file (represented here by the argument 'file') into the buffer given. The argument 'length' is the length of the buffer given.

The value returned is the length of the information actually read. So, when 'actualLength' is greater than zero, the value of 'actualLength' is the the number of characters read. Usually Read will try to fill up your buffer before returning. A value of zero means that end-of-file has been reached. Errors are indicated by a value of -1. In any case, the value of IoErr() is also modified by this call. If there was an error it gives more error information, otherwise it indicates whether there is any more data in the file.

**INPUTS**

file - BCPL pointer to a file handle  
buffer - pointer to buffer  
length - integer

**OUTPUTS**

actualLength - integer

## 1.25 dos.library/Rename

---

## NAME

Rename -- Rename a directory or file

## SYNOPSIS

```
success = Rename( oldName, newName )
D0          D1      D2
```

```
BOOL success;
char *oldName, *newName;
```

## FUNCTION

Rename() attempts to rename the file or directory specified as 'oldName' with the name 'newName'. If the file or directory 'newName' exists, Rename() fails and returns an error. Both 'oldName' and the 'newName' can contain a directory specification. In this case, the file will be moved from one directory to another.

Note: it is impossible to Rename() a file from one volume to another.

## INPUTS

```
oldName - pointer to a null-terminated string
newName - pointer to a null-terminated string
```

## OUTPUTS

```
success - boolean
```

## 1.26 dos.library/Seek

## NAME

Seek -- Find and point at the logical position in a file

## SYNOPSIS

```
oldPosition = Seek( file, position, mode )
D0          D1      D2          D3
```

```
LONG oldPosition, position, mode;
struct FileHandle *file;
```

## FUNCTION

Seek() sets the read/write cursor for the file 'file' to the position 'position'. This position is used by both Read() and Write() as a place to start reading or writing. The result is the current absolute position in the file, or -1 if an error occurs, in which case IoErr() can be used to find more information. 'mode' can be OFFSET\_BEGINNING, OFFSET\_CURRENT or OFFSET\_END. It is used to specify the relative start position. For example, 20 from current is a position 20 bytes forward from current, -20 is 20 bytes back from current.

So that to find out where you are, seek zero from current. The end of the file is a Seek() positioned by zero from end. You cannot Seek() beyond the end of a file.

## INPUTS

file - BCPL pointer to a file handle  
position - integer  
mode - integer

## OUTPUTS

oldPosition - integer

## 1.27 dos.library/SetComment

## NAME

SetComment -- Change a files' comment string

## SYNOPSIS

```
success = SetComment( name, comment )  
D0                D1    D2
```

```
BOOL success;  
char *name;  
char *comment;
```

## FUNCTION

SetComment() sets a comment on a file or directory. The comment is a pointer to a null-terminated string of up to 80 characters.

## INPUTS

name - pointer to a null-terminated string  
comment - pointer to a null-terminated string

## 1.28 dos.library/SetProtection

## NAME

SetProtection -- Set protection for a file or directory

## SYNOPSIS

```
success = SetProtection( name, mask )  
D0                D1    D2:4
```

```
BOOL success;  
char *name;  
LONG mask;
```

## FUNCTION

SetProtection() sets the protection attributes on a file or directory. The lower bits of the mask are as follows:

Bits 31-4 Reserved.

|                                   |                           |
|-----------------------------------|---------------------------|
| bit 4: 1 = file has not changed   | 0 = file has been changed |
| bit 3: 1 = reads not allowed,     | 0 = reads allowed.        |
| bit 2: 1 = writes not allowed,    | 0 = writes allowed.       |
| bit 1: 1 = execution not allowed, | 0 = execution allowed.    |
| bit 0: 1 = deletion not allowed,  | 0 = deletion allowed.     |

Only delete is checked for by the Old Filing System. The archive bit is cleared by the file system whenever the file is changed. Backup utilities will generally set the bit after backing up each file.

The new Fast Filing System looks at the read and write bits, and the Shell looks at the execute bit, and will refuse to start a file as a binary executable if it is set.

Other bits may will be defined in the "libraries/dos.h" include files. Rather than referring to bits by number you should use the definitions in "dos.h".

#### INPUTS

name - pointer to a null-terminated string  
mask - the protection mask required

#### OUTPUTS

success - boolean

## 1.29 dos.library/UnLoadSeg

#### NAME

UnLoadSeg -- Unload a segment previously loaded by LoadSeg()

#### SYNOPSIS

```
error = UnLoadSeg( segment )  
D0          D1
```

```
BOOL error;  
BPTR segment;
```

#### FUNCTION

Unload a segment loaded by LoadSeg(). 'segment' may be zero.

#### INPUTS

segment - BCPL pointer to a segment identifier

#### OUTPUTS

error - boolean

## 1.30 dos.library/UnLock

#### NAME

UnLock -- Unlock a directory or file

#### SYNOPSIS

```
UnLock( lock )  
D1
```

```
struct FileLock *lock;
```

---

## FUNCTION

The filing system lock [obtained from Lock(), DupLock(), or CreateDir()] is removed and deallocated.

## INPUTS

lock - BCPL pointer to a lock

## NOTE

passing zero to UnLock() is harmless

## 1.31 dos.library/WaitForChar

## NAME

WaitForChar -- Determine if chars arrive within a time limit

## SYNOPSIS

```
status = WaitForChar( file, timeout )
D0          D1      D2
```

```
BOOL status;
struct FileHandle *file;
LONG timeout;
```

## FUNCTION

If a character is available to be read from 'file' within a the time (in microseconds) indicated by 'timeout', WaitForChar() returns -1 (TRUE). If a character is available, you can use Read() to read it. Note that WaitForChar() is only valid when the I/O stream is connected to a virtual terminal device. If a character is not available within 'timeout', a 0 (FALSE) is returned.

## BUGS

Due to a bug in the timer.device in V1.2/V1.3, specifying a timeout of zero for WaitForChar() can cause the unreliable timer & floppy disk operation.

## INPUTS

file - BCPL pointer to a file handle  
timeout - integer

## OUTPUTS

status - boolean

## 1.32 dos.library/Write

## NAME

Write -- Write bytes of data to a file

## SYNOPSIS

```
returnedLength = Write( file, buffer, length )
D0          D1      D2      D3
```

```
LONG returnedLength;  
struct FileHandle *file;  
char *buffer;  
LONG length;
```

#### FUNCTION

Write() writes bytes of data to the opened file 'file'. 'length' indicates the length of data to be transferred; 'buffer' is a pointer to the buffer. The value returned is the length of information actually written. So, when 'length' is greater than zero, the value of 'length' is the number of characters written. Errors are indicated by a value of -1.

#### INPUTS

file - BCPL pointer to a file handle  
buffer - pointer to the buffer  
length - integer

#### OUTPUTS

returnedLength - integer

---