

**dos**

|                      |
|----------------------|
| <b>COLLABORATORS</b> |
|----------------------|

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

|                         |
|-------------------------|
| <b>REVISION HISTORY</b> |
|-------------------------|

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

# Contents

|          |                                |          |
|----------|--------------------------------|----------|
| <b>1</b> | <b>dos</b>                     | <b>1</b> |
| 1.1      | dos.doc                        | 1        |
| 1.2      | dos.library/AbortPkt           | 2        |
| 1.3      | dos.library/AddBuffers         | 2        |
| 1.4      | dos.library/AddDosEntry        | 3        |
| 1.5      | dos.library/AddPart            | 4        |
| 1.6      | dos.library/AddSegment         | 5        |
| 1.7      | dos.library/AllocDosObject     | 5        |
| 1.8      | dos.library/AssignAdd          | 6        |
| 1.9      | dos.library/AssignLate         | 7        |
| 1.10     | dos.library/AssignLock         | 7        |
| 1.11     | dos.library/AssignPath         | 8        |
| 1.12     | dos.library/AttemptLockDosList | 8        |
| 1.13     | dos.library/ChangeMode         | 9        |
| 1.14     | dos.library/CheckSignal        | 10       |
| 1.15     | dos.library/Cli                | 10       |
| 1.16     | dos.library/CliInitNewcli      | 10       |
| 1.17     | dos.library/CliInitRun         | 11       |
| 1.18     | dos.library/Close              | 12       |
| 1.19     | dos.library/CompareDates       | 13       |
| 1.20     | dos.library/CreateDir          | 13       |
| 1.21     | dos.library/CreateNewProc      | 14       |
| 1.22     | dos.library/CreateProc         | 15       |
| 1.23     | dos.library/CurrentDir         | 16       |
| 1.24     | dos.library/DateStamp          | 16       |
| 1.25     | dos.library/DateToStr          | 17       |
| 1.26     | dos.library/Delay              | 18       |
| 1.27     | dos.library/DeleteFile         | 19       |
| 1.28     | dos.library/DeleteVar          | 19       |
| 1.29     | dos.library/DeviceProc         | 20       |

|  |    |
|--|----|
| 1.30 dos.library/DoPkt . . . . .             | 21 |
| 1.31 dos.library/DupLock . . . . .           | 21 |
| 1.32 dos.library/DupLockFromFH . . . . .     | 22 |
| 1.33 dos.library/EndNotify . . . . .         | 22 |
| 1.34 dos.library/ErrorReport . . . . .       | 23 |
| 1.35 dos.library/ExAll . . . . .             | 24 |
| 1.36 dos.library/Examine . . . . .           | 26 |
| 1.37 dos.library/ExamineFH . . . . .         | 27 |
| 1.38 dos.library/Execute . . . . .           | 27 |
| 1.39 dos.library/Exit . . . . .              | 29 |
| 1.40 dos.library/ExNext . . . . .            | 29 |
| 1.41 dos.library/Fault . . . . .             | 30 |
| 1.42 dos.library/FGetC . . . . .             | 31 |
| 1.43 dos.library/FGets . . . . .             | 32 |
| 1.44 dos.library/FilePart . . . . .          | 32 |
| 1.45 dos.library/FindArg . . . . .           | 33 |
| 1.46 dos.library/FindCliProc . . . . .       | 33 |
| 1.47 dos.library/FindDosEntry . . . . .      | 34 |
| 1.48 dos.library/FindSegment . . . . .       | 34 |
| 1.49 dos.library/FindVar . . . . .           | 35 |
| 1.50 dos.library/Flush . . . . .             | 36 |
| 1.51 dos.library/Format . . . . .            | 36 |
| 1.52 dos.library/FPutC . . . . .             | 37 |
| 1.53 dos.library/FPuts . . . . .             | 38 |
| 1.54 dos.library/FRead . . . . .             | 38 |
| 1.55 dos.library/FreeArgs . . . . .          | 39 |
| 1.56 dos.library/FreeDeviceProc . . . . .    | 39 |
| 1.57 dos.library/FreeDosEntry . . . . .      | 40 |
| 1.58 dos.library/FreeDosObject . . . . .     | 40 |
| 1.59 dos.library/FWrite . . . . .            | 41 |
| 1.60 dos.library/GetArgStr . . . . .         | 41 |
| 1.61 dos.library/GetConsoleTask . . . . .    | 42 |
| 1.62 dos.library/GetCurrentDirName . . . . . | 42 |
| 1.63 dos.library/GetDeviceProc . . . . .     | 43 |
| 1.64 dos.library/GetFileSysTask . . . . .    | 44 |
| 1.65 dos.library/GetProgramDir . . . . .     | 44 |
| 1.66 dos.library/GetProgramName . . . . .    | 45 |
| 1.67 dos.library/GetPrompt . . . . .         | 45 |
| 1.68 dos.library/GetVar . . . . .            | 46 |

---

|  |    |
|--|----|
| 1.69 dos.library/Info . . . . .                | 47 |
| 1.70 dos.library/Inhibit . . . . .             | 47 |
| 1.71 dos.library/Input . . . . .               | 48 |
| 1.72 dos.library/InternalLoadSeg . . . . .     | 48 |
| 1.73 dos.library/InternalUnLoadSeg . . . . .   | 49 |
| 1.74 dos.library/IOErr . . . . .               | 49 |
| 1.75 dos.library/IsFileSystem . . . . .        | 50 |
| 1.76 dos.library/IsInteractive . . . . .       | 51 |
| 1.77 dos.library/LoadSeg . . . . .             | 51 |
| 1.78 dos.library/Lock . . . . .                | 52 |
| 1.79 dos.library/LockDosList . . . . .         | 52 |
| 1.80 dos.library/LockRecord . . . . .          | 54 |
| 1.81 dos.library/LockRecords . . . . .         | 55 |
| 1.82 dos.library/MakeDosEntry . . . . .        | 56 |
| 1.83 dos.library/MakeLink . . . . .            | 56 |
| 1.84 dos.library/MatchEnd . . . . .            | 57 |
| 1.85 dos.library/MatchFirst . . . . .          | 57 |
| 1.86 dos.library/MatchNext . . . . .           | 59 |
| 1.87 dos.library/MatchPattern . . . . .        | 60 |
| 1.88 dos.library/MatchPatternNoCase . . . . .  | 60 |
| 1.89 dos.library/MaxCli . . . . .              | 61 |
| 1.90 dos.library/NameFromFH . . . . .          | 61 |
| 1.91 dos.library/NameFromLock . . . . .        | 62 |
| 1.92 dos.library/NewLoadSeg . . . . .          | 62 |
| 1.93 dos.library/NextDosEntry . . . . .        | 63 |
| 1.94 dos.library/Open . . . . .                | 64 |
| 1.95 dos.library/OpenFromLock . . . . .        | 64 |
| 1.96 dos.library/Output . . . . .              | 65 |
| 1.97 dos.library/ParentDir . . . . .           | 66 |
| 1.98 dos.library/ParentOfFH . . . . .          | 66 |
| 1.99 dos.library/ParsePattern . . . . .        | 67 |
| 1.100 dos.library/ParsePatternNoCase . . . . . | 68 |
| 1.101 dos.library/PathPart . . . . .           | 68 |
| 1.102 dos.library/PrintFault . . . . .         | 69 |
| 1.103 dos.library/PutStr . . . . .             | 70 |
| 1.104 dos.library/Read . . . . .               | 70 |
| 1.105 dos.library/ReadArgs . . . . .           | 71 |
| 1.106 dos.library/ReadItem . . . . .           | 73 |
| 1.107 dos.library/ReadLink . . . . .           | 73 |

|  |    |
|--|----|
| 1.108dos.library/Relabel . . . . .           | 74 |
| 1.109dos.library/RemAssignList . . . . .     | 75 |
| 1.110dos.library/RemDosEntry . . . . .       | 75 |
| 1.111dos.library/RemSegment . . . . .        | 76 |
| 1.112dos.library/Rename . . . . .            | 76 |
| 1.113dos.library/ReplyPkt . . . . .          | 77 |
| 1.114dos.library/RunCommand . . . . .        | 77 |
| 1.115dos.library/SameDevice . . . . .        | 78 |
| 1.116dos.library/SameLock . . . . .          | 79 |
| 1.117dos.library/Seek . . . . .              | 79 |
| 1.118dos.library/SelectInput . . . . .       | 80 |
| 1.119dos.library/SelectOutput . . . . .      | 80 |
| 1.120dos.library/SendPkt . . . . .           | 81 |
| 1.121dos.library/SetArgStr . . . . .         | 82 |
| 1.122dos.library/SetComment . . . . .        | 82 |
| 1.123dos.library/SetConsoleTask . . . . .    | 83 |
| 1.124dos.library/SetCurrentDirName . . . . . | 83 |
| 1.125dos.library/SetFileDate . . . . .       | 84 |
| 1.126dos.library/SetFileSize . . . . .       | 84 |
| 1.127dos.library/SetFileSysTask . . . . .    | 85 |
| 1.128dos.library/SetIoErr . . . . .          | 85 |
| 1.129dos.library/SetMode . . . . .           | 86 |
| 1.130dos.library/SetProgramDir . . . . .     | 86 |
| 1.131dos.library/SetProgramName . . . . .    | 87 |
| 1.132dos.library/SetPrompt . . . . .         | 87 |
| 1.133dos.library/SetProtection . . . . .     | 88 |
| 1.134dos.library/SetVar . . . . .            | 89 |
| 1.135dos.library/SetVBuf . . . . .           | 89 |
| 1.136dos.library/SplitName . . . . .         | 90 |
| 1.137dos.library/StartNotify . . . . .       | 91 |
| 1.138dos.library/StrToDate . . . . .         | 91 |
| 1.139dos.library/StrToLong . . . . .         | 93 |
| 1.140dos.library/SystemTagList . . . . .     | 93 |
| 1.141dos.library/UnGetC . . . . .            | 94 |
| 1.142dos.library/UnLoadSeg . . . . .         | 95 |
| 1.143dos.library/UnLock . . . . .            | 95 |
| 1.144dos.library/UnLockDosList . . . . .     | 96 |
| 1.145dos.library/UnLockRecord . . . . .      | 96 |
| 1.146dos.library/UnLockRecords . . . . .     | 97 |

---

|  |     |
|--|-----|
| 1.147dos.library/VFPrintf . . . . .    | 98  |
| 1.148dos.library/VFWritef . . . . .    | 98  |
| 1.149dos.library/VPrintf . . . . .     | 99  |
| 1.150dos.library/WaitForChar . . . . . | 100 |
| 1.151dos.library/WaitPkt . . . . .     | 101 |
| 1.152dos.library/Write . . . . .       | 101 |
| 1.153dos.library/WriteChars . . . . .  | 102 |

---

# Chapter 1

## dos

### 1.1 dos.doc

|                       |                       |                      |
|-----------------------|-----------------------|----------------------|
| AbortPkt ()           | Fputs ()              | Read ()              |
| AddBuffers ()         | FRead ()              | ReadArgs ()          |
| AddDosEntry ()        | FreeArgs ()           | ReadItem ()          |
| AddPart ()            | FreeDeviceProc ()     | ReadLink ()          |
| AddSegment ()         | FreeDosEntry ()       | Relabel ()           |
| AllocDosObject ()     | FreeDosObject ()      | RemAssignList ()     |
| AssignAdd ()          | FWrite ()             | RemDosEntry ()       |
| AssignLate ()         | GetArgStr ()          | RemSegment ()        |
| AssignLock ()         | GetConsoleTask ()     | Rename ()            |
| AssignPath ()         | GetCurrentDirName ()  | ReplyPkt ()          |
| AttemptLockDosList () | GetDeviceProc ()      | RunCommand ()        |
| ChangeMode ()         | GetFileSysTask ()     | SameDevice ()        |
| CheckSignal ()        | GetProgramDir ()      | SameLock ()          |
| Cli ()                | GetProgramName ()     | Seek ()              |
| CliInitNewcli ()      | GetPrompt ()          | SelectInput ()       |
| CliInitRun ()         | GetVar ()             | SelectOutput ()      |
| Close ()              | Info ()               | SendPkt ()           |
| CompareDates ()       | Inhibit ()            | SetArgStr ()         |
| CreateDir ()          | Input ()              | SetComment ()        |
| CreateNewProc ()      | InternalLoadSeg ()    | SetConsoleTask ()    |
| CreateProc ()         | InternalUnLoadSeg ()  | SetCurrentDirName () |
| CurrentDir ()         | IoErr ()              | SetFileDate ()       |
| DateStamp ()          | IsFileSystem ()       | SetFileSize ()       |
| DateToStr ()          | IsInteractive ()      | SetFileSysTask ()    |
| Delay ()              | LoadSeg ()            | SetIoErr ()          |
| DeleteFile ()         | Lock ()               | SetMode ()           |
| DeleteVar ()          | LockDosList ()        | SetProgramDir ()     |
| DeviceProc ()         | LockRecord ()         | SetProgramName ()    |
| DoPkt ()              | LockRecords ()        | SetPrompt ()         |
| DupLock ()            | MakeDosEntry ()       | SetProtection ()     |
| DupLockFromFH ()      | MakeLink ()           | SetVar ()            |
| EndNotify ()          | MatchEnd ()           | SetVBuf ()           |
| ErrorReport ()        | MatchFirst ()         | SplitName ()         |
| ExAll ()              | MatchNext ()          | StartNotify ()       |
| Examine ()            | MatchPattern ()       | StrToDate ()         |
| ExamineFH ()          | MatchPatternNoCase () | StrToLong ()         |
| Execute ()            | MaxCli ()             | SystemTagList ()     |
| Exit ()               | NameFromFH ()         | UnGetC ()            |



|                |                      |                 |
|----------------|----------------------|-----------------|
| ExNext()       | NameFromLock()       | UnLoadSeg()     |
| Fault()        | NewLoadSeg()         | UnLock()        |
| FGetC()        | NextDosEntry()       | UnLockDosList() |
| FGets()        | Open()               | UnLockRecord()  |
| FilePart()     | OpenFromLock()       | UnLockRecords() |
| FindArg()      | Output()             | VFPprintf()     |
| FindCliProc()  | ParentDir()          | VFWwritef()     |
| FindDosEntry() | ParentOfFH()         | VPrintf()       |
| FindSegment()  | ParsePattern()       | WaitForChar()   |
| FindVar()      | ParsePatternNoCase() | WaitPkt()       |
| Flush()        | PathPart()           | Write()         |
| Format()       | PrintFault()         | WriteChars()    |
| FPutC()        | PutStr()             |                 |

## 1.2 dos.library/AbortPkt

### NAME

AbortPkt -- Aborts an asynchronous packet, if possible. (V36)

### SYNOPSIS

```
AbortPkt(port, pkt)
        D1      D2
```

```
void AbortPkt(struct MsgPort *, struct DosPacket *)
```

### FUNCTION

This attempts to abort a packet sent earlier with SendPkt to a handler. There is no guarantee that any given handler will allow a packet to be aborted, or if it is aborted whether function requested completed first or completely. After calling AbortPkt(), you must wait for the packet to return before reusing it or deallocating it.

### INPUTS

port - port the packet was sent to  
pkt - the packet you wish aborted

### BUGS

As of V37, this function does nothing.

### SEE ALSO

SendPkt(), DoPkt(), WaitPkt()

## 1.3 dos.library/AddBuffers

### NAME

AddBuffers -- Changes the number of buffers for a filesystem (V36)

### SYNOPSIS

```
success = AddBuffers(filesystem, number)
        D0              D1              D2
```

```
BOOL AddBuffers(char *, LONG)
```

#### FUNCTION

Adds buffers to a filesystem. If it succeeds, the number of current buffers is returned in `IoErr()`. Note that "number" may be negative. The amount of memory used per buffer, and any limits on the number of buffers, are dependant on the filesystem in question. If the call succeeds, the number of buffers in use on the filesystem will be returned by `IoErr()`.

#### INPUTS

`filesystem` - Name of device to add buffers to (with ':').  
`number` - Number of buffers to add. May be negative.

#### RESULT

`success` - Success or failure of command.

#### BUGS

The V36 ROM filesystem (FFS/OFS) doesn't return the right number of buffers unless preceded by an `AddBuffers(fs,-1)` (in-use buffers aren't counted). This is fixed in V37.

The V37 and before ROM filesystem doesn't return success, it returns the number of buffers. The best way to test for this is to consider 0 (FALSE) failure, -1 (DOSTRUE) to mean that `IoErr()` will have the number of buffers, and any other positive value to be the number of buffers. It may be fixed in some future ROM revision.

#### SEE ALSO

`IoErr()`

## 1.4 dos.library/AddDosEntry

#### NAME

`AddDosEntry` -- Add a Dos List entry to the lists (V36)

#### SYNOPSIS

```
success = AddDosEntry(dlist)
D0                      D1
```

```
LONG AddDosEntry(struct DosList *)
```

#### FUNCTION

Adds a device, volume or assign to the dos devicelist. Can fail if it conflicts with an existing entry (such as another assign to the same name or another device of the same name). Volume nodes with different dates and the same name CAN be added, or with names that conflict with devices or assigns. Note: the dos list does NOT have to be locked to call this. Do not access `dlist` after adding unless you have locked the Dos Device list.

An additional note concerning calling this from within a handler: in order to avoid deadlocks, your handler must either be multi-threaded, or it must attempt to lock the list before calling this function. The code would look something like this:

```

if (AttemptLockDosList(LDF_xxx|LDF_WRITE))
{
    rc = AddDosEntry(...);
    UnLockDosList(LDF_xxx|LDF_WRITE);
}

```

If AttemptLockDosList() fails (i.e. it's locked already), check for messages at your filesystem port (don't wait!) and try the AttemptLockDosList() again.

#### INPUTS

dlist - Device list entry to be added.

#### RESULT

success - Success/Failure indicator

#### SEE ALSO

RemDosEntry(), FindDosEntry(), NextDosEntry(), LockDosList(),  
MakeDosEntry(), FreeDosEntry(), AttemptLockDosList()

## 1.5 dos.library/AddPart

#### NAME

AddPart -- Appends a file/dir to the end of a path (V36)

#### SYNOPSIS

```

success = AddPart( dirname, filename, size )
D0          D1          D2          D3

```

```

BOOL AddPart( UBYTE *, UBYTE *, ULONG )

```

#### FUNCTION

This function adds a file, directory, or subpath name to a directory path name taking into account any required separator characters. If filename is a fully-qualified path it will totally replace the current value of dirname.

#### INPUTS

dirname - the path to add a file/directory name to.  
filename - the filename or directory name to add. May be a relative pathname from the current directory (example: foo/bar). Can deal with leading '/'(s), indicating one directory up per '/', or with a ':', indicating it's relative to the root of the appropriate volume.  
size - size in bytes of the space allocated for dirname. Must not be 0.

#### RESULT

success - non-zero for ok, FALSE if the buffer would have overflowed. If an overflow would have occurred, dirname will not be changed.

#### BUGS

Doesn't check if a subpath is legal (i.e. doesn't check for ':'s) and

doesn't handle leading '/'s in 2.0 through 2.02 (V36). V37 fixes this, allowing filename to be any path, including absolute.

SEE ALSO

Filepart(), PathPart()

## 1.6 dos.library/AddSegment

NAME

AddSegment - Adds a resident segment to the resident list (V36)

SYNOPSIS

```
success = AddSegment(name, seglist, type)
D0                      D1      D2      D3
```

```
BOOL AddSegment(char *, BPTR, LONG)
```

FUNCTION

Adds a segment to the Dos resident list, with the specified Seglist and type (stored in seg\_UC - normally 0). NOTE: currently unused types may cause it to interpret other registers (d4-?) as additional parameters in the future.

Do NOT build Segment structures yourself!

INPUTS

```
name      - name for the segment
seglist   - Dos seglist of code for segment
type      - initial usecount, normally 0
```

RESULT

```
success - success or failure
```

SEE ALSO

FindSegment(), RemSegment(), LoadSeg()

## 1.7 dos.library/AllocDosObject

NAME

AllocDosObject -- Creates a dos object (V36)

SYNOPSIS

```
ptr = AllocDosObject(type, tags)
D0                      D1      D2
```

```
void *AllocDosObject(ULONG, struct TagItem *)
```

```
ptr = AllocDosObjectTagList(type, tags)
D0                      D1      D2
```

```
void *AllocDosObjectTagList(ULONG, struct TagItem *)
```

---

```
ptr = AllocDosObjectTags(type, Tag1, ...)
```

```
void *AllocDosObjectTags(ULONG, ULONG, ...)
```

#### FUNCTION

Create one of several dos objects, initializes it, and returns it to you. Note the DOS\_STDPKT returns a pointer to the sp\_Pkt of the structure.

#### INPUTS

type - type of object requested  
tags - pointer to taglist with additional information

#### RESULT

packet - pointer to the object or NULL

#### SEE ALSO

FreeDosObject(), <dos/dostags.h>, <dos/dos.h>

## 1.8 dos.library/AssignAdd

#### NAME

AssignAdd -- Adds a lock to an assign for multi-directory assigns (V36)

#### SYNOPSIS

```
success = AssignAdd(name, lock)
D0                      D1    D2
```

```
BOOL AssignAdd(char *,BPTR)
```

#### FUNCTION

Adds a lock to an assign, making or adding to a multi-directory assign. Note that this only will succeed on an assign created with AssignLock(), or an assign created with AssignLate() which has been resolved (converted into a AssignLock()-assign).

NOTE: you should not use the lock in any way after making this call successfully. It becomes the part of the assign, and will be unlocked by the system when the assign is removed. If you need to keep the lock, pass a lock from DupLock() to AssignLock().

#### INPUTS

name - Name of device to assign lock to (without trailing ':')  
lock - Lock associated with the assigned name

#### RESULT

success - Success/failure indicator. On failure, the lock is not unlocked.

#### SEE ALSO

Lock(), AssignLock(), AssignPath(), AssignLate(), DupLock(), RemAssignList()

## 1.9 dos.library/AssignLate

### NAME

AssignLate -- Creates an assignment to a specified path later (V36)

### SYNOPSIS

```
success = AssignLate(name,path)
D0                      D1   D2
```

```
BOOL AssignLate(char *,char *)
```

### FUNCTION

Sets up a assignment that is expanded upon the FIRST reference to the name. The path (a string) would be attached to the node. When the name is referenced (Open("FOO:xyzzzy"...), the string will be used to determine where to set the assign to, and if the directory can be locked, the assign will act from that point on as if it had been created by AssignLock().

A major advantage is assigning things to unmounted volumes, which will be requested upon access (useful in startup sequences).

### INPUTS

name - Name of device to be assigned (without trailing ':')  
path - Name of late assignment to be resolved on the first reference.

### RESULT

success - Success/failure indicator of the operation

### SEE ALSO

Lock(), AssignAdd(), AssignPath(), AssignLock(),

## 1.10 dos.library/AssignLock

### NAME

AssignLock -- Creates an assignment to a locked object (V36)

### SYNOPSIS

```
success = AssignLock(name,lock)
D0                      D1   D2
```

```
BOOL AssignLock(char *,BPTR)
```

### FUNCTION

Sets up an assign of a name to a given lock. Passing NULL for a lock cancels any outstanding assign to that name. If an assign entry of that name is already on the list, this routine replaces that entry. If an entry is on the list that conflicts with the new assign, then a failure code is returned.

NOTE: you should not use the lock in any way after making this call successfully. It becomes the assign, and will be unlocked by the system when the assign is removed. If you need to keep the lock, pass a lock from DupLock() to AssignLock().

---

## INPUTS

name - Name of device to assign lock to (without trailing ':')  
 lock - Lock associated with the assigned name

## RESULT

success - Success/failure indicator. On failure, the lock is not unlocked.

## SEE ALSO

Lock(), AssignAdd(), AssignPath(), AssignLate(), DupLock(),  
 RemAssignList()

## 1.11 dos.library/AssignPath

## NAME

AssignPath -- Creates an assignment to a specified path (V36)

## SYNOPSIS

```
success = AssignPath(name,path)
D0          D1    D2
```

```
BOOL AssignPath(char *,char *)
```

## FUNCTION

Sets up a assignment that is expanded upon EACH reference to the name. This is implemented through a new device list type (DLT\_ASSIGNPATH, or some such). The path (a string) would be attached to the node. When the name is referenced (Open("FOO:xyzzzy"...), the string will be used to determine where to do the open. No permanent lock will be part of it. For example, you could AssignPath() c2: to df2:c, and references to c2: would go to df2:c, even if you change disks.

The other major advantage is assigning things to unmounted volumes, which will be requested upon access (useful in startup sequences).

## INPUTS

name - Name of device to be assigned (without trailing ':')  
 path - Name of late assignment to be resolved at each reference

## RESULT

success - Success/failure indicator of the operation

## SEE ALSO

AssignAdd(), AssignLock(), AssignLate(), Open()

## 1.12 dos.library/AttemptLockDosList

## NAME

AttemptLockDosList -- Attempt to lock the Dos Lists for use (V36)

## SYNOPSIS

```
dlist = AttemptLockDosList(flags)
D0                                     D1
```

```
struct DosList *AttemptLockDosList(ULONG)
```

#### FUNCTION

Locks the dos device list in preparation to walk the list. If the list is 'busy' then this routine will return NULL. See LockDosList() for more information.

#### INPUTS

flags - Flags stating which types of nodes you want to lock.

#### RESULT

dlist - Pointer to the beginning of the list or NULL. Not a valid node!

#### SEE ALSO

LockDosList(), UnLockDosList(), Forbid(), NextDosEntry()

## 1.13 dos.library/ChangeMode

#### NAME

ChangeMode - Change the current mode of a lock or filehandle (V36)

#### SYNOPSIS

```
success = ChangeMode(type, object, newmode)
D0                                     D1      D2      D3
```

```
BOOL ChangeMode(ULONG, BPTR, ULONG)
```

#### FUNCTION

This allows you to attempt to change the mode in use by a lock or filehandle. For example, you could attempt to turn a shared lock into an exclusive lock. The handler may well reject this request. Warning: if you use the wrong type for the object, the system may crash.

#### INPUTS

type - Either CHANGE\_FH or CHANGE\_LOCK  
object - A lock or filehandle  
newmode - The new mode you want

#### RESULT

success - Boolean

#### BUGS

Did not work in 2.02 or before (V36). Works in V37. In the earlier versions, it can crash the machine.

#### SEE ALSO

Lock(), Open()



## 1.14 dos.library/CheckSignal

### NAME

CheckSignal -- Checks for break signals (V36)

### SYNOPSIS

```
signals = CheckSignal(mask)
D0                      D1
```

```
ULONG CheckSignals(ULONG)
```

### FUNCTION

This function checks to see if any signals specified in the mask have been set and if so, returns them. Otherwise it returns FALSE. All signals specified in mask will be cleared.

### INPUTS

mask      - Signals to check for.

### RESULT

signals - Signals specified in mask that were set.

### SEE ALSO

## 1.15 dos.library/Cli

### NAME

Cli -- Returns a pointer to the CLI structure of the process (V36)

### SYNOPSIS

```
cli_ptr = Cli()
D0
```

```
struct CommandLineInterface *Cli(void)
```

### FUNCTION

Returns a pointer to the CLI structure of the current process, or NULL if the process has no CLI structure.

### RESULT

cli\_ptr - pointer to the CLI structure, or NULL.

### SEE ALSO

## 1.16 dos.library/CliInitNewcli

### NAME

CliInitNewcli -- Set up a process to be a shell from initial packet

### SYNOPSIS

```
flags = CliInitNewcli( packet )
D0                      A0
```

---

```
LONG CliInitNewcli( struct DosPacket * )
```

#### FUNCTION

This function initializes a process and CLI structure for a new shell, from parameters in an initial packet passed by the system (NewShell or NewCLI, etc). The format of the data in the packet is purposely not defined. The setup includes all the normal fields in the structures that are required for proper operation (current directory, paths, input streams, etc).

It returns a set of flags containing information about what type of shell invocation this is.

Definitions for the values of fn:

|        |                                    |
|--------|------------------------------------|
| Bit 31 | Set to indicate flags are valid    |
| Bit 3  | Set to indicate asynch system call |
| Bit 2  | Set if this is a System() call     |
| Bit 1  | Set if user provided input stream  |
| Bit 0  | Set if RUN provided output stream  |

If Bit 31 is 0, then you must check IoErr() to determine if an error occurred. If IoErr() returns a pointer to your process, there has been an error, and you should clean up and exit. The packet will have already been returned by CliInitNewcli(). If it isn't a pointer to your process and Bit 31 is 0, reply the packet immediately. (Note: this is different from what you do for CliInitRun().)

This function is very similar to CliInitRun().

#### INPUTS

packet - the initial packet sent to your process MsgPort

#### RESULT

fn - flags or a pointer

#### SEE ALSO

CliInitRun(), ReplyPkt(), WaitPkt(), IoErr()

## 1.17 dos.library/CliInitRun

#### NAME

CliInitRun -- Set up a process to be a shell from initial packet

#### SYNOPSIS

```
flags = CliInitRun( packet )
D0                                A0
```

```
LONG CliInitRun( struct DosPacket * )
```

#### FUNCTION

This function initializes a process and CLI structure for a new shell, from parameters in an initial packet passed by the system (Run, System(), Execute()). The format of the data in the packet is purposely not defined. The setup includes all the normal fields

in the structures that are required for proper operation (current directory, paths, input streams, etc).

It returns a set of flags containing information about what type of shell invocation this is.

Definitions for the values of fn:

|        |                                    |
|--------|------------------------------------|
| Bit 31 | Set to indicate flags are valid    |
| Bit 3  | Set to indicate asynch system call |
| Bit 2  | Set if this is a System() call     |
| Bit 1  | Set if user provided input stream  |
| Bit 0  | Set if RUN provided output stream  |

If Bit 31 is 0, then you must check IoErr() to determine if an error occurred. If IoErr() returns a pointer to your process, there has been an error, and you should clean up and exit. The packet will have already been returned by CliInitNewcli(). If it isn't a pointer to your process and Bit 31 is 0, you should wait before replying the packet until after you've loaded the first command (or when you exit). This helps avoid disk "gronking" with the Run command. (Note: this is different from what you do for CliInitNewcli().)

If Bit 31 is 1, then if Bit 3 is one, ReplyPkt() the packet immediately (Asynch System()), otherwise wait until your shell exits (Sync System(), Execute()). (Note: this is different from what you do for CliInitNewcli().)

This function is very similar to CliInitNewcli().

#### INPUTS

packet - the initial packet sent to your process MsgPort

#### RESULT

fn - flags or a pointer

#### SEE ALSO

CliInitNewcli(), ReplyPkt(), WaitPkt(), System(), Execute(), IoErr()

## 1.18 dos.library/Close

#### NAME

Close -- Close an open file

#### SYNOPSIS

```
success = Close( file )
           D0          D1
```

BOOL Close(BPTR)

#### 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). If Close() fails, the file handle is still deallocated and should not be used.

## INPUTS

file - BCPL pointer to a file handle

## RESULTS

success - returns if Close() succeeded. Note that it might fail depending on buffering and whatever IO must be done to close a file being written to. NOTE: this return value did not exist before V36!

## SEE ALSO

Open(), OpenFromLock()

## 1.19 dos.library/CompareDates

## NAME

CompareDates -- Compares two timestamps (V36)

## SYNOPSIS

```
result = CompareDates(date1,date2)
D0                      D1      D2
```

```
LONG CompareDates(struct DateStamp *,struct DateStamp *)
```

## FUNCTION

Compares two times for relative magnitude. <0 is returned if date1 is later than date2, 0 if they are equal, or >0 if date2 is later than date1. NOTE: this is NOT the same ordering as strcmp!

## INPUTS

date1, date2 - DateStamps to compare

## RESULT

result - <0, 0, or >0 based on comparison of two date stamps

## SEE ALSO

DateStamp(), DateToStr(), StrToDate()

## 1.20 dos.library/CreateDir

## NAME

CreateDir -- Create a new directory

## SYNOPSIS

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

```
BPTR CreateDir(char *)
```

## 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. CreateDir returns an exclusive lock on the new directory if it succeeds.

#### INPUTS

name - pointer to a null-terminated string

#### RESULTS

lock - BCPL pointer to a lock or NULL for failure.

#### SEE ALSO

Lock(), UnLock()

## 1.21 dos.library/CreateNewProc

#### NAME

CreateNewProc -- Create a new process (V36)

#### SYNOPSIS

```
process = CreateNewProc(tags)
D0                      D1

struct Process *CreateNewProc(struct TagItem *)

process = CreateNewProcTagList(tags)
D0                      D1

struct Process *CreateNewProcTagList(struct TagItem *)

process = CreateNewProcTags(Tag1, ...)

struct Process *CreateNewProcTags(ULONG, ...)
```

#### FUNCTION

This creates a new process according to the tags passed in. See dos/dostags.h for the tags.

You must specify one of NP\_Seglist or NP\_Entry. NP\_Seglist takes a seglist (as returned by LoadSeg()). NP\_Entry takes a function pointer for the routine to call.

There are many options, as you can see by examining dos/dostags.h. The defaults are for a non-CLI process, with copies of your CurrentDir, HomeDir (used for PROGDIR:), priority, consoletask, windowptr, and variables. The input and output filehandles default to opens of NIL:, stack to 4000, and others as shown in dostags.h. This is a fairly reasonable default setting for creating threads, though you may wish to modify it (for example, to give a descriptive name to the process.)

CreateNewProc() is callable from a task, though any actions that require doing Dos I/O (DupLock() of currentdir, for example) will not occur.

NOTE: if you call CreateNewProc() with both NP\_Arguments, you must not specify an NP\_Input of NULL. When NP\_Arguments is specified, it

needs to modify the input filehandle to make ReadArgs() work properly.

#### INPUTS

tags - a pointer to a TagItem array.

#### RESULT

process - The created process, or NULL. Note that if it returns NULL, you must free any items that were passed in via tags, such as if you passed in a new current directory with NP\_CurrentDir.

#### BUGS

In V36, NP\_Arguments was broken in a number of ways, and probably should be avoided (instead you should start a small piece of your own code, which calls RunCommand() to run the actual code you wish to run). In V37, NP\_Arguments works, though see the note above.

#### SEE ALSO

LoadSeg(), CreateProc(), ReadArgs(), RunCommand(), <dos/dostags.h>

## 1.22 dos.library/CreateProc

#### NAME

CreateProc -- Create a new process

#### SYNOPSIS

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

```
struct MsgPort *CreateProc(char *, LONG, BPTR, LONG)
```

#### FUNCTION

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

A seglist, 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 seglist (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 msgport address of the new process, or zero if the routine failed. The argument 'name' specifies the new process name. A zero return code indicates error.

The seglist passed to CreateProc() is not freed when it exits; it is up to the parent process to free it, or for the code to unload itself.

Under V36 and later, you probably should use CreateNewProc() instead.

#### INPUTS

name - pointer to a null-terminated string  
 pri - signed long (range -128 to +127)  
 seglist - BCPL pointer to a seglist  
 stackSize - integer (must be a multiple of 4 bytes)

#### RESULTS

process - pointer to new process msgport

#### SEE ALSO

CreateNewProc(), LoadSeg(), UnLoadSeg()

## 1.23 dos.library/CurrentDir

#### NAME

CurrentDir -- Make a directory lock the current directory

#### SYNOPSIS

```
oldLock = CurrentDir( lock )
D0                      D1
```

```
BPTR CurrentDir(BPTR)
```

#### 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.

Any call that has to Open() or Lock() files (etc) requires that the current directory be a valid lock or 0.

#### INPUTS

lock - BCPL pointer to a lock

#### RESULTS

oldLock - BCPL pointer to a lock

#### SEE ALSO

Lock(), UnLock(), Open(), DupLock()

## 1.24 dos.library/DateStamp

#### NAME

DateStamp -- Obtain the date and time in internal format

---

## SYNOPSIS

```
ds = DateStamp( ds );
D0          D1
```

```
struct DateStamp *DateStamp(struct DateStamp *)
```

## FUNCTION

DateStamp() takes a structure 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.

Time is measured from Jan 1, 1978.

## INPUTS

ds - pointer a struct DateStamp

## RESULTS

The array is filled as described and returned (for pre-V36 compabability).

## SEE ALSO

DateToStr(), StrToDate(), SetFileDate(), CompareDates()

## 1.25 dos.library/DateToStr

## NAME

DateToStr -- Converts a DateStamp to a string (V36)

## SYNOPSIS

```
success = DateToStr( datetime )
D0          D1
```

```
BOOL DateToStr(struct DateTime *)
```

## FUNCTION

StampToStr converts an AmigaDOS DateStamp to a human readable ASCII string as requested by your settings in the DateTime structure.

## INPUTS

DateTime - a pointer to an initialized DateTime structure.

The DateTime structure should be initialized as follows:

dat\_Stamp - a copy of the datestamp you wish to convert to  
ascii.

dat\_Format - a format byte which specifies the format of the



dat\_StrDate. This can be any of the following (note: If value used is something other than those below, the default of FORMAT\_DOS is used):

FORMAT\_DOS: AmigaDOS format (dd-mmm-yy).

FORMAT\_INT: International format (yy-mmm-dd).

FORMAT\_USA: American format (mm-dd-yy).

FORMAT\_CDN: Canadian format (dd-mm-yy).

FORMAT\_DEF: default format for locale.

dat\_Flags - a flags byte. The only flag which affects this function is:

DTF\_SUBST: If set, a string such as Today, Monday, etc., will be used instead of the dat\_Format specification if possible.

DTF\_FUTURE: Ignored by this function.

dat\_StrDay - pointer to a buffer to receive the day of the week string. (Monday, Tuesday, etc.). If null, this string will not be generated.

dat\_StrDate - pointer to a buffer to receive the date string, in the format requested by dat\_Format, subject to possible modifications by DTF\_SUBST. If null, this string will not be generated.

dat\_StrTime - pointer to a buffer to receive the time of day string. If NULL, this will not be generated.

#### RESULT

success - a zero return indicates that the DateStamp was invalid, and could not be converted. Non-zero indicates that the call succeeded.

#### SEE ALSO

DateStamp(), StrToDate(), <dos/datetime.h>

## 1.26 dos.library/Delay

#### NAME

Delay -- Delay a process for a specified time

#### SYNOPSIS

```
Delay( ticks )
    D1
```

```
void Delay(ULONG)
```

#### FUNCTION

---

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

#### INPUTS

ticks - integer

#### 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. This is fixed in V36 and later.

#### SEE ALSO

## 1.27 dos.library/DeleteFile

#### NAME

DeleteFile -- Delete a file or directory

#### SYNOPSIS

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

```
BOOL DeleteFile(char *)
```

#### 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

#### RESULTS

success - boolean

#### SEE ALSO

## 1.28 dos.library/DeleteVar

#### NAME

DeleteVar -- Deletes a local or environment variable (V36)

#### SYNOPSIS

```
success = DeleteVar( name, flags )  
D0                      D1      D2
```

```
BOOL DeleteVar(UBYTE *, ULONG )
```

#### FUNCTION

Deletes a local or environment variable.

---

## INPUTS

name - pointer to an variable name. Note variable names follow filesystem syntax and semantics.

flags - combination of type of var to delete (low 8 bits), and flags to control the behavior of this routine. Currently defined flags include:

GVF\_LOCAL\_ONLY - delete a local (to your process) variable.  
GVF\_GLOBAL\_ONLY - delete a global environment variable.

The default is to delete a local variable if found, otherwise a global environment variable if found (only for LV\_VAR).

## RESULT

success - If non-zero, the variable was successfully deleted, FALSE indicates failure.

## BUGS

LV\_VAR is the only type that can be global

## SEE ALSO

GetVar(), SetVar(), FindVar(), DeleteFile(), <dos/var.h>

## 1.29 dos.library/DeviceProc

## NAME

DeviceProc -- Return the process MsgPort of specific I/O handler

## SYNOPSIS

```
process = DeviceProc( name )
D0                      D1

struct MsgPort *DeviceProc (char *)
```

## 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 an assign then a directory lock is returned in IoErr(). This lock should not be Unlock()ed or Examine()ed (if you wish to do so, DupLock() it first).

## BUGS

In V36, if you try to DeviceProc() something relative to an assign made with AssignPath(), it will fail. This is because there's no way to know when to unlock the lock. If you're writing code for V36 or later, it is highly advised you use GetDeviceProc() instead, or make your code conditional on V36 to use GetDeviceProc()/FreeDeviceProc().

## SEE ALSO

GetDeviceProc(), FreeDeviceProc(), DupLock(), Unlock(), Examine()

## 1.30 dos.library/DoPkt

### NAME

DoPkt -- Send a dos packet and wait for reply (V36)

### SYNOPSIS

```
result1 = DoPkt(port,action,arg1,arg2,arg3,arg4,arg5)
D0          D1      D2      D3      D4      D5      D6      D7
```

```
LONG DoPkt(struct MsgPort *,LONG,LONG,LONG,LONG,LONG,LONG)
```

### FUNCTION

Sends a packet to a handler and waits for it to return. Any secondary return will be available in D1 AND from IoErr(). DoPkt() will work even if the caller is an exec task and not a process; however it will be slower, and may fail for some additional reasons, such as being unable to allocate a signal. DoPkt() uses your pr\_MsgPort for the reply, and will call pr\_PktWait. (See BUGS regarding tasks, though).

Only allows 5 arguments to be specified. For more arguments (packets support a maximum of 7) create a packet and use SendPkt()/WaitPkt().

### INPUTS

port - pr\_MsgPort of the handler process to send to.  
 action - the action requested of the filesystem/handler  
 arg1, arg2, arg3, arg4, arg5 - arguments, depend on the action, may not be required.

### RESULT

result1 - the value returned in dp\_Res1, or FALSE if there was some problem in sending the packet or receiving it.  
 result2 - Available from IoErr() AND in register D1.

### BUGS

Using DoPkt() from tasks doesn't work in V36. Use AllocDosObject(), PutMsg(), and WaitPort()/GetMsg() for a workaround, or you can call CreateNewProc() to start a process to do Dos I/O for you. In V37, DoPkt() will allocate, use, and free the MsgPort required.

### NOTES

Callable from a task (under V37 and above).

### SEE ALSO

AllocDosObject(), FreeDosObject(), SendPkt(), WaitPkt(), CreateNewProc(), AbortPkt()

## 1.31 dos.library/DupLock

### NAME

DupLock -- Duplicate a lock

### SYNOPSIS

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

BPTR DupLock (BPTR)

#### 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 exclusive lock.

A zero return indicates failure.

#### INPUTS

lock - BCPL pointer to a lock

#### RESULTS

newLock - BCPL pointer to a lock

#### SEE ALSO

Lock(), UnLock(), DupLockFromFH(), ParentOfFH()

## 1.32 dos.library/DupLockFromFH

#### NAME

DupLockFromFH -- Gets a lock on an open file (V36)

#### SYNOPSIS

```
lock = DupLockFromFH(fh)
D0                                     D1
```

BPTR DupLockFromFH (BPTR)

#### FUNCTION

Obtain a lock on the object associated with fh. Only works if the file was opened using a non-exclusive mode. Other restrictions may be placed on success by the filesystem.

#### INPUTS

fh - Opened file for which to obtain the lock

#### RESULT

lock - Obtained lock or NULL for failure

#### SEE ALSO

DupLock(), Lock(), UnLock()

## 1.33 dos.library/EndNotify

#### NAME

EndNotify -- Ends a notification request (V36)

---

## SYNOPSIS

```
EndNotify(notifystructure)
        D1
```

```
VOID EndNotify(struct NotifyRequest *)
```

## FUNCTION

Removes a notification request. Safe to call even if StartNotify() failed. For NRF\_SEND\_MESSAGE, it searches your port for any messages about the object in question and removes and replies them before returning.

## INPUTS

notifystructure - a structure passed to StartNotify()

## SEE ALSO

StartNotify(), <dos/notify.h>

## 1.34 dos.library/ErrorReport

## NAME

ErrorReport -- Displays a Retry/Cancel requester for an error (V36)

## SYNOPSIS

```
status = ErrorReport(code, type, arg1, device)
        D0          D1      D2      D3      A0
```

```
BOOL ErrorReport(LONG, LONG, ULONG, struct MsgPort *)
```

## FUNCTION

Based on the request type, this routine formats the appropriate requester to be displayed. If the code is not understood, it returns DOS\_TRUE immediately. Returns DOS\_TRUE if the user selects CANCEL or if the attempt to put up the requester fails, or if the process pr\_WindowPtr is -1. Returns FALSE if the user selects Retry. The routine will retry on DISKINSERTED for appropriate error codes. These return values are the opposite of what AutoRequest returns.

Note: this routine sets IoErr() to code before returning.

## INPUTS

```
code    - Error code to put a requester up for.
          Current valid error codes are:
            ERROR_DISK_NOT_VALIDATED
            ERROR_DISK_WRITE_PROTECTED
            ERROR_DISK_FULL
            ERROR_DEVICE_NOT_MOUNTED
            ERROR_NOT_A_DOS_DISK
            ERROR_NO_DISK
            ABORT_DISK_ERROR      /* read/write error */
            ABORT_BUSY           /* you MUST replace... */

type    - Request type:
            REPORT_LOCK    - arg1 is a lock (BPTR).
            REPORT_FH      - arg1 is a filehandle (BPTR).
            REPORT_VOLUME  - arg1 is a volumenode (C pointer).
```

REPORT\_INSERT - arg1 is the string for the volumename  
 (will be split on a ':').  
 With ERROR\_DEVICE\_NOT\_MOUNTED puts  
 up the "Please insert..." requester.

arg1 - variable parameter (see type)

device - (Optional) Address of handler task for which report is to be  
 made. Only required for REPORT\_LOCK, and only if arg1==NULL.

#### RESULT

status - Cancel/Retry indicator (0 means Retry)

#### SEE ALSO

Fault(), IoErr()

## 1.35 dos.library/ExAll

#### NAME

ExAll -- Examine an entire directory (V36)

#### SYNOPSIS

```
continue = ExAll(lock, buffer, size, type, control)
D0          D1      D2      D3      D4      D5
```

```
BOOL ExAll(BPTR, UBYTE *, LONG, LONG, struct ExAllControl *)
```

#### FUNCTION

Examines an entire directory.

Lock must be on a directory. Size is the size of the buffer supplied.

The buffer will be filled with (partial) ExAllData structures, as  
 specified by the type field.

Type is a value from those shown below that determines which information is  
 to be stored in the buffer. Each higher value adds a new thing to the list  
 as described in the table below:-

|               |                                      |
|---------------|--------------------------------------|
| ED_NAME       | FileName                             |
| ED_TYPE       | Type                                 |
| ED_SIZE       | Size in bytes                        |
| ED_PROTECTION | Protection bits                      |
| ED_DATE       | 3 longwords of date                  |
| ED_COMMENT    | Comment (will be NULL if no comment) |

Thus, ED\_NAME gives only filenames, and ED\_COMMENT gives everything.

The ead\_Next entry gives a pointer to the next entry in the buffer. The  
 last entry will have NULL in ead\_Next.

The control structure is required so that FFS can keep track if more than  
 one call to ExAll is required. This happens when there are more names in  
 a directory than will fit into the buffer. The format of the control  
 structure is as follows:-

NOTE: the control structure MUST be allocated by AllocDosObject!!!

**Entries:** This field tells the calling application how many entries are in the buffer after calling `ExAll`. Note: make sure your code handles the 0 entries case, including 0 entries with continue non-zero.

**LastKey:** This field ABSOLUTELY MUST be initialised to 0 before calling `ExAll` for the first time. Any other value will cause nasty things to happen. If `ExAll` returns non-zero, then this field should not be touched before making the second and subsequent calls to `ExAll`. Whenever `ExAll` returns non-zero, there are more calls required before all names have been received.

As soon as a FALSE return is received then `ExAll` has completed (if `IoErr()` returns `ERROR_NO_MORE_ENTRIES` - otherwise it returns the error that occurred, similar to `ExNext`.)

#### MatchString

If this field is NULL then all filenames will be returned. If this field is non-null then it is interpreted as a pointer to a string that is used to pattern match all file names before accepting them and putting them into the buffer. The default AmigaDOS caseless pattern match routine is used. This string MUST have been parsed by `ParsePatternNoCase()`!

#### MatchFunc:

Contains a pointer to a hook for a routine to decide if the entry will be included in the returned list of entries. The entry is filled out first, and then passed to the hook. If no `MatchFunc` is to be called then this entry should be NULL. The hook is called with the following parameters (as is standard for hooks):

```

BOOL = MatchFunc( hookptr, data, typeptr )
                   a0      a1      a2
(a0 = ptr to hook, a1 = ptr to filled in ExAllData, a2 = ptr
to longword of type).
```

`MatchFunc` should return FALSE if the entry is not to be accepted, otherwise return TRUE.

Note that Dos will emulate `ExAll()` using `Examine()` and `ExNext()` if the handler in question doesn't support the `ExAll()` packet.

#### INPUTS

`lock` - Lock on directory to be examined.  
`buffer` - Buffer for data returned (MUST be at least word-aligned, preferably long-word aligned).  
`size` - Size in bytes of 'buffer'.  
`type` - Type of data to be returned.  
`control` - Control data structure (see notes above). MUST have been allocated by `AllocDosObject`!

#### RESULT

`continue` - Whether or not `ExAll` is done. If FALSE is returned, either `ExAll` has completed (`IoErr() == ERROR_NO_MORE_ENTRIES`), or an error occurred (check `IoErr()`). If non-zero is returned, you MUST call `ExAll` again until it returns FALSE.



## EXAMPLE

```

eac = AllocDosObject(DOS_EXALLCONTROL, NULL);
if (!eac) ...
...
eac->eac_LastKey = 0;
do {
    more = ExAll(lock, EADData, sizeof(EADData), ED_FOO, eac);
    if ((!more) && (IoErr() != ERROR_NO_MORE_ENTRIES)) {
        /* ExAll failed abnormally */
        break;
    }
    if (eac->eac_Entries == 0) {
        /* ExAll failed normally with no entries */
        continue;          /* ("more" is *usually* zero) */
    }
    ead = (struct ExAllData *) EADData;
    do {
        /* use ead here */
        ...
        /* get next ead */
        ead = ead->ed_Next;
    } while (ead);
} while (more);
...

FreeDosObject(DOS_EXALLCONTROL, eac);

```

## BUGS

In V36, there were problems with ExAll (particularly with eac\_MatchString, and ed\_Next with the ramdisk and the emulation of it in Dos for handlers that do not support the packet. It is advised you only use this under V37 and later.

## SEE ALSO

Examine(), ExNext(), ExamineFH(), MatchPatternNoCase(), ParsePatternNoCase(), AllocDosObject()

## 1.36 dos.library/Examine

## NAME

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

## SYNOPSIS

```

success = Examine( lock, FileInfoBlock )
D0                D1                D2

```

```

BOOL Examine(BPTR, struct FileInfoBlock *)

```

## 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 to ExNext().

#### INPUTS

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

#### RESULTS

success - boolean

#### SPECIAL NOTE

FileInfoBlock must be longword-aligned. AllocDosObject() will allocate them correctly for you.

#### SEE ALSO

Lock(), UnLock(), ExNext(), ExamineFH(), <dos/dos.h>, AllocDosObject(), ExAll()

## 1.37 dos.library/ExamineFH

#### NAME

ExamineFH -- Gets information on an open file (V36)

#### SYNOPSIS

success = ExamineFH(fh, fib)  
D0 D1 D2

BOOL ExamineFH(BPTR, struct FileInfoBlock \*)

#### FUNCTION

Examines a filehandle and returns information about the file in the FileInfoBlock. There are no guarantees as to whether the fib\_Size field will reflect any changes made to the file size it was opened, though filesystems should attempt to provide up-to-date information for it.

#### INPUTS

fh - Filehandle you wish to examine  
fib - FileInfoBlock, must be longword aligned.

#### RESULT

success - Success/failure indication

#### SEE ALSO

Examine(), ExNext(), ExAll(), Open(), AllocDosObject()

## 1.38 dos.library/Execute

#### NAME

Execute -- Execute a CLI command

---

## SYNOPSIS

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

```
BOOL Execute(char *, BPTR, BPTR)
```

## FUNCTION

This function attempts to execute the string `commandString` as a Shell command and arguments. The string can contain any valid input that you could type directly in a Shell, including input and output redirection using `<` and `>`. Note that `Execute()` doesn't return until the command(s) in `commandString` have returned.

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 empty) `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 Shell 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 Shell process just like those created with the `NewShell` command. 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 Shell will read commands from the new window, and will use the same window for output. This new Shell window can only be terminated by using the `EndCLI` command.

Under V37, if an input filehandle is passed, and it's either interactive or a `NIL:filehandle`, the `pr_ConsoleTask` of the new process will be set to that filehandle's process (the same applies to `SystemTagList()`).

For this command to work the program `Run` must be present in `C:` in versions before V36 (except that in 1.3.2 and any later 1.3 versions, the system first checks the resident list for `Run`).

## INPUTS

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

## RESULTS

```
success - BOOLEAN indicating whether Execute was successful
         in finding and starting the specified program. Note this
         is NOT the return code of the command(s).
```

## SEE ALSO

```
SystemTagList(), NewShell, EndCLI, Run
```

## 1.39 dos.library/Exit

### NAME

Exit -- Exit from a program

### SYNOPSIS

```
Exit( returnCode )
    D1
```

```
void Exit(LONG)
```

### 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 with their original stack ptr.

### 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

### SEE ALSO

CreateProc(), CreateNewProc()

## 1.40 dos.library/ExNext

### NAME

ExNext -- Examine the next entry in a directory

### SYNOPSIS

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

```
BOOL ExNext(BPTR, struct FileInfoBlock *)
```

---

**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 `fib_DirEntryType` 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.

**RESULTS**

`success` - boolean

**SPECIAL NOTE**

FileInfoBlock must be longword-aligned. `AllocDosObject()` will allocate them correctly for you.

**SEE ALSO**

`Examine()`, `Lock()`, `UnLock()`, `IoErr()`, `ExamineFH()`, `AllocDosObject()`, `ExAll()`

## 1.41 dos.library/Fault

**NAME**

`Fault` -- Returns the text associated with a DOS error code (V36)

## SYNOPSIS

```
success = Fault(code, header, buffer, len)
D0          D1      D2      D3      D4
```

```
BOOL Fault(LONG, UBYTE *, UBYTE *, LONG)
```

## FUNCTION

This routine obtains the error message text for the given error code. The header is prepended to the text of the error message, followed by a colon. Puts a null-terminated string for the error message into the buffer. By convention, error messages should be no longer than 80 characters (+1 for termination), and preferably no more than 60. The value returned by IoErr() is set to the code passed in. If there is no message for the error code, the message will be "Error code <number>\n".

## INPUTS

```
code    - Error code
header  - header to output before error text
buffer  - Buffer to receive error message.
len     - Length of the buffer.
```

## RESULT

```
success - Success/failure code.
```

## SEE ALSO

```
IoErr(), SetIoErr(), PrintFault()
```

## 1.42 dos.library/FGetC

## NAME

```
FGetC -- Read a character from the specified input (buffered) (V36)
```

## SYNOPSIS

```
char = FGetC(fh)
D0          D1
```

```
LONG FGetC(BPTR)
```

## FUNCTION

Reads the next character from the input stream. A -1 is returned when EOF or an error is encountered. This call is buffered. Use Flush() between buffered and unbuffered I/O on a filehandle.

## INPUTS

```
fh - filehandle to use for buffered I/O
```

## RESULT

```
char - character read (0-255) or -1
```

## BUGS

In V36, after an EOF was read, EOF would always be returned from FGetC() from then on. Starting in V37, it tries to read from the handler again each time (unless UnGetC(fh,-1) was called).

SEE ALSO

FPutC(), UnGetC(), Flush()

## 1.43 dos.library/FGets

NAME

FGets -- Reads a line from the specified input (buffered) (V36)

SYNOPSIS

buffer = FGets(fh, buf, len)

D0                    D1   D2   D3

UBYTE \*FGets(BPTR, UBYTE \*, ULONG)

FUNCTION

This routine reads in a single line from the specified input stopping at a NEWLINE character or EOF. In either event, UP TO the number of len specified bytes minus 1 will be copied into the buffer. Hence if a length of 50 is passed and the input line is longer than 49 bytes, it will return 49 characters. It returns the buffer pointer normally, or NULL if EOF is the first thing read.

If terminated by a newline, the newline WILL be the last character in the buffer. This is a buffered read routine. The string read in IS null-terminated.

INPUTS

fh - filehandle to use for buffered I/O

buf - Area to read bytes into.

len - Number of bytes to read, must be > 0.

RESULT

buffer - Pointer to buffer passed in, or NULL for immediate EOF or for an error. If NULL is returned for an EOF, IoErr() will return 0.

SEE ALSO

FRead(), FPutS(), FGetC()

## 1.44 dos.library/FilePart

NAME

FilePart -- Returns the last component of a path (V36)

SYNOPSIS

fileptr = FilePart( path )

D0                    D1

UBYTE \*FilePart( UBYTE \* )

FUNCTION

---

This function returns a pointer to the last component of a string path specification, which will normally be the file name. If there is only one component, it returns a pointer to the beginning of the string.

**INPUTS**

path - pointer to an path string. May be relative to the current directory or the current disk.

**RESULT**

fileptr - pointer to the last component of the path.

**EXAMPLE**

FilePart("xxx:yyy/zzz/qqq") would return a pointer to the first 'q'.  
FilePart("xxx:yyy") would return a pointer to the first 'y').

**SEE ALSO**

PathPart(), AddPart()

## 1.45 dos.library/FindArg

**NAME**

FindArg - find a keyword in a template (V36)

**SYNOPSIS**

index = FindArg(template, keyword)

D0                      D1              D2

LONG FindArg(UBYTE \*, UBYTE \*)

**FUNCTION**

Returns the argument number of the keyword, or -1 if it is not a keyword for the template. Abbreviations are handled.

**INPUTS**

keyword - keyword to search for in template  
template - template string to search

**RESULT**

index - number of entry in template, or -1 if not found

**BUGS**

In earlier published versions of the autodoc, keyword and template were backwards.

**SEE ALSO**

ReadArgs(), ReadItem(), FreeArgs()

## 1.46 dos.library/FindCliProc

**NAME**

FindCliProc -- returns a pointer to the requested CLI process (V36)

---



## SYNOPSIS

```
proc = FindCliProc(num)
D0          D1
```

```
struct Process *FindCliProc(LONG)
```

## FUNCTION

This routine returns a pointer to the CLI process associated with the given CLI number. If the process isn't an active CLI process, NULL is returned. NOTE: should normally be called inside a Forbid(), if you must use this function at all.

## INPUTS

num - Task number of CLI process

## RESULT

proc - Pointer to given CLI process

## SEE ALSO

Cli(), Forbid(), MaxCli()

## 1.47 dos.library/FindDosEntry

## NAME

FindDosEntry -- Finds a specific Dos List entry (V36)

## SYNOPSIS

```
newdlist = FindDosEntry(dlist,name,flags)
D0          D1      D2      D3
```

```
struct DosList *FindDosEntry(struct DosList *,UBYTE *,ULONG)
```

## FUNCTION

Locates an entry on the device list. Starts with the entry dlist. NOTE: must be called with the device list locked, no references may be made to dlist after unlocking.

## INPUTS

dlist - The device entry to start with.  
 name - Name of device entry (without ':') to locate.  
 flags - Search control flags. Use the flags you passed to LockDosList, or a subset of them. LDF\_READ/LDF\_WRITE are not required for this call.

## RESULT

newdlist - The device entry or NULL

## SEE ALSO

AddDosEntry(), RemDosEntry(), NextDosEntry(), LockDosList(), MakeDosEntry(), FreeDosEntry()

## 1.48 dos.library/FindSegment

## NAME

FindSegment - Finds a segment on the resident list (V36)

## SYNOPSIS

```
segment = FindSegment(name, start, system)
```

```
D0                D1      D2      D3
```

```
struct Segment *FindSegment(char *, struct Segment *, LONG)
```

## FUNCTION

Finds a segment on the Dos resident list by name and type, starting at the segment AFTER 'start', or at the beginning if start is NULL. If system is zero, it will only return nodes with a seg\_UC of 0 or more. It does NOT increment the seg\_UC, and it does NOT do any locking of the list. You must Forbid() lock the list to use this call.

To use an entry you have found, you must: if the seg\_UC is 0 or more, increment it, and decrement it (under Forbid()!) when you're done the the seglist.

The other values for seg\_UC are:

- 1 - system module, such as a filesystem or shell
- 2 - resident shell command
- 999 - disabled internal command, ignore

Negative values should never be modified. All other negative values between 0 and -32767 are reserved to AmigaDos and should not be used.

## INPUTS

```
name    - name of segment to find
start   - segment to start the search after
system  - true for system segment, false for normal segments
```

## RESULT

```
segment - the segment found or NULL
```

## SEE ALSO

```
AddSegment(), RemSegment(), Forbid()
```

## 1.49 dos.library/FindVar

## NAME

FindVar -- Finds a local variable (V36)

## SYNOPSIS

```
var = FindVar( name, type )
```

```
D0                D1      D2
```

```
struct LocalVar * FindVar(UBYTE *, ULONG )
```

## FUNCTION

Finds a local variable structure.

## INPUTS

name - pointer to an variable name. Note variable names follow filesystem syntax and semantics.

type - type of variable to be found (see <dos/var.h>)

## RESULT

var - pointer to a LocalVar structure or NULL

## SEE ALSO

GetVar(), SetVar(), DeleteVar(), <dos/var.h>

## 1.50 dos.library/Flush

## NAME

Flush -- Flushes buffers for a buffered filehandle (V36)

## SYNOPSIS

```
success = Flush(fh)
D0                      D1
```

```
LONG Flush(BPTR)
```

## FUNCTION

Flushes any pending buffered writes to the filehandle. All buffered writes will also be flushed on Close(). If the filehandle was being used for input, it drops the buffer, and tries to Seek() back to the last read position (so subsequent reads or writes will occur at the expected position in the file).

## INPUTS

fh - Filehandle to flush.

## RESULT

success - Success or failure.

## BUGS

Before V37 release, Flush() returned a random value. As of V37, it always returns success (this will be fixed in some future release).

## SEE ALSO

Fputc(), Fgetc(), Ungetc(), Seek(), Close()

## 1.51 dos.library/Format

## NAME

Format -- Causes a filesystem to initialize itself (V36)

## SYNOPSIS

```
success = Format(filesystem, volumename, dostype)
D0                      D1                      D2                      D3
```

BOOL Format(UBYTE \*, UBYTE \*, ULONG)

#### FUNCTION

Interface for initializing new media on a device. This causes the filesystem to write out an empty disk structure to the media, which should then be ready for use. This assumes the media has been low-level formatted and verified already.

The filesystem should be inhibited before calling Format() to make sure you don't get an ERROR\_OBJECT\_IN\_USE.

#### INPUTS

filesystem - Name of device to be formatted. ':' must be supplied.  
volumename - Name for volume (if supported). No ':'.  
dostype - Type of format, if filesystem supports multiple types.

#### RESULT

success - Success/failure indicator.

#### BUGS

Existed, but was non-functional in V36 dos. (The volumename wasn't converted to a BSTR.) Workaround: require V37, or under V36 convert volumename to a BPTR to a BSTR before calling Format().  
Note: a number of printed packet docs for ACTION\_FORMAT are wrong as to the arguments.

#### SEE ALSO

## 1.52 dos.library/FPutC

#### NAME

FPutC -- Write a character to the specified output (buffered) (V36)

#### SYNOPSIS

char = FPutC(fh, char)  
D0            D1    D2

LONG FPutC(BPTR, UBYTE)

#### FUNCTION

Writes a single character to the output stream. This call is buffered. Use Flush() between buffered and unbuffered I/O on a filehandle. Interactive filehandles are flushed automatically on a newline, return, '\0', or line feed.

#### INPUTS

fh - filehandle to use for buffered I/O  
char - character to write

#### RESULT

char - either the character written, or EOF for an error.

#### SEE ALSO

---

FGetC(), UnGetC(), Flush()

## 1.53 dos.library/Fputs

### NAME

Fputs -- Writes a string the the specified output (buffered) (V36)

### SYNOPSIS

```
error = Fputs(fh, str)
D0          D1  D2
```

LONG Fputs(BPTR, UBYTE \*)

### FUNCTION

This routine writes an unformatted string to the filehandle. No newline is appended to the string and the length actually written is returned. This routine is buffered.

### INPUTS

fh - filehandle to use for buffered I/O  
str - Null-terminated string to be written to default output

### RESULT

error - 0 normally, otherwise -1. Note that this is opposite of most other Dos functions, which return success.

### SEE ALSO

FGets(), FPutC(), FWrite(), PutStr()

## 1.54 dos.library/FRead

### NAME

FRead -- Reads a number of blocks from an input (buffered) (V36)

### SYNOPSIS

```
count = FRead(fh, buf, blocklen, blocks)
D0          D1  D2      D3          D4
```

LONG FRead(BPTR, UBYTE \*, ULONG, ULONG)

### FUNCTION

Attempts to read a number of blocks, each blocklen long, into the specified buffer from the input stream. May return less than the number of blocks requested, either due to EOF or read errors. This call is buffered.

### INPUTS

fh - filehandle to use for buffered I/O  
buf - Area to read bytes into.  
blocklen - number of bytes per block. Must be > 0.  
blocks - number of blocks to read. Must be > 0.

---

**RESULT**

count - Number of `_blocks_` read, or 0 for EOF. On an error, the number of blocks actually read is returned.

**BUGS**

Doesn't clear `IoErr()` before starting. If you want to find out about errors, use `SetIoErr(0L)` before calling.

**SEE ALSO**

`FGetC()`, `FWrite()`, `FGets()`

## 1.55 dos.library/FreeArgs

**NAME**

`FreeArgs` - Free allocated memory after `ReadArgs()` (V36)

**SYNOPSIS**

```
FreeArgs(rdargs)
    D1
```

```
void FreeArgs(struct RArgs *)
```

**FUNCTION**

Frees memory allocated to return arguments in from `ReadArgs()`. If `ReadArgs` allocated the `RArgs` structure it will be freed.

**INPUTS**

rdargs - structure returned from `ReadArgs()`

**SEE ALSO**

`ReadArgs()`, `ReadItem()`, `FindArg()`

## 1.56 dos.library/FreeDeviceProc

**NAME**

`FreeDeviceProc` -- Releases port returned by `GetDeviceProc()` (V36)

**SYNOPSIS**

```
FreeDeviceProc(devproc)
    D1
```

```
void FreeDeviceProc(struct DevProc *)
```

**FUNCTION**

Frees up the structure created by `GetDeviceProc()`, and any associated temporary locks.

Decrements the counter incremented by `GetDeviceProc()`. The counter is in an extension to the 1.3 process structure. After calling `FreeDeviceProc()`, do not use the port or lock again! It is safe to call `FreeDeviceProc(NULL)`.

---

## INPUTS

devproc - A value returned by GetDeviceProc()

## BUGS

Counter not currently active in 2.0.

## SEE ALSO

GetDeviceProc(), DeviceProc(), AssignLock(), AssignLate(),  
AssignPath()

## 1.57 dos.library/FreeDosEntry

## NAME

FreeDosEntry -- Frees an entry created by MakeDosEntry (V36)

## SYNOPSIS

```
FreeDosEntry(dlist)
             D1
```

```
void FreeDosEntry(struct DosList *)
```

## FUNCTION

Frees an entry created by MakeDosEntry(). This routine should be eliminated and replaced by a value passed to FreeDosObject()!

## INPUTS

dlist - DosList to free.

## SEE ALSO

AddDosEntry(), RemDosEntry(), FindDosEntry(), LockDosList(),  
NextDosEntry(), MakeDosEntry()

## 1.58 dos.library/FreeDosObject

## NAME

FreeDosObject -- Frees an object allocated by AllocDosObject() (V36)

## SYNOPSIS

```
FreeDosObject(type, ptr)
             D1  D2
```

```
void FreeDosObject(ULONG, void *)
```

## FUNCTION

Frees an object allocated by AllocDosObject(). Do NOT call for objects allocated in any other way.

## INPUTS

type - type passed to AllocDosObject()  
ptr - ptr returned by AllocDosObject()

## SEE ALSO

AllocDosObject(), <dos/dos.h>

## 1.59 dos.library/FWrite

### NAME

FWrite -- Writes a number of blocks to an output (buffered) (V36)

### SYNOPSIS

```
count = FWrite(fh, buf, blocklen, blocks)
D0                      D1  D2      D3      D4
```

LONG FWrite(BPTR, UBYTE \*, ULONG, ULONG)

### FUNCTION

Attempts to write a number of blocks, each blocklen long, from the specified buffer to the output stream. May return less than the number of blocks requested, if there is some error such as a full disk or r/w error. This call is buffered.

### INPUTS

fh           - filehandle to use for buffered I/O  
buf           - Area to write bytes from.  
blocklen   - number of bytes per block. Must be > 0.  
blocks      - number of blocks to read. Must be > 0.

### RESULT

count - Number of `_blocks_` written. On an error, the number of blocks actually written is returned.

### BUGS

Doesn't clear IoErr() before starting. If you want to find out about errors, use SetIoErr(0L) before calling.

### SEE ALSO

FPutC(), FRead(), Fputs()

## 1.60 dos.library/GetArgStr

### NAME

GetArgStr -- Returns the arguments for the process (V36)

### SYNOPSIS

```
ptr = GetArgStr()
D0
```

UBYTE \*GetArgStr(void)

### FUNCTION

Returns a pointer to the (null-terminated) arguments for the program (process). This is the same string passed in a0 on startup from CLI.

### RESULT

---



ptr - pointer to arguments

SEE ALSO

SetArgStr(), RunCommand()

## 1.61 dos.library/GetConsoleTask

NAME

GetConsoleTask -- Returns the default console for the process (V36)

SYNOPSIS

```
port = GetConsoleTask()
D0
```

```
struct MsgPort *GetConsoleTask(void)
```

FUNCTION

Returns the default console task's port (pr\_ConsoleTask) for the current process.

RESULT

port - The pr\_MsgPort of the console handler, or NULL.

SEE ALSO

SetConsoleTask(), Open()

## 1.62 dos.library/GetCurrentDirName

NAME

GetCurrentDirName -- returns the current directory name (V36)

SYNOPSIS

```
success = GetCurrentDirName(buf, len)
D0                      D1  D2
```

```
BOOL GetCurrentDirName(char *, LONG)
```

FUNCTION

Extracts the current directory name from the CLI structure and puts it into the buffer. If the buffer is too small, the name is truncated appropriately and a failure code returned. If no CLI structure is present, a null string is returned in the buffer, and failure from the call (with IoErr() == ERROR\_OBJECT\_WRONG\_TYPE);

INPUTS

buf - Buffer to hold extracted name  
len - Number of bytes of space in buffer

RESULT

success - Success/failure indicator

BUGS

---

In V36, this routine didn't handle 0-length buffers correctly.

SEE ALSO

SetCurrentDirName()

## 1.63 dos.library/GetDeviceProc

NAME

GetDeviceProc -- Finds a handler to send a message to (V36)

SYNOPSIS

```
devproc = GetDeviceProc(name, devproc)
      D0                      D1      D2
```

```
struct DevProc *GetDeviceProc(UBYTE *, struct DevProc *)
```

FUNCTION

Finds the handler/filesystem to send packets regarding 'name' to. This may involve getting temporary locks. It returns a structure that includes a lock and msgport to send to to attempt your operation. It also includes information on how to handle multiple-directory assigns (by passing the DevProc back to GetDeviceProc() until it returns NULL).

The initial call to GetDeviceProc() should pass NULL for devproc. If after using the returned DevProc, you get an ERROR\_OBJECT\_NOT\_FOUND, and (devproc->devp\_Flags & DVPF\_ASSIGN) is true, you should call GetDeviceProc() again, passing it the devproc structure. It will either return a modified devproc structure, or NULL (with ERROR\_NO\_MORE\_ENTRIES in IoErr()). Continue until it returns NULL.

This call also increments the counter that locks a handler/fs into memory. After calling FreeDeviceProc(), do not use the port or lock again!

INPUTS

name - name of the object you wish to access. This can be a relative path ("foo/bar"), relative to the current volume (":foo/bar"), or relative to a device/volume/assign ("foo:bar").

devproc - A value returned by GetDeviceProc() before, or NULL

RESULT

devproc - a pointer to a DevProc structure or NULL

BUGS

Counter not currently active in 2.0.  
In 2.0 and 2.01, you HAD to check DVPF\_ASSIGN before calling it again. This was fixed for the 2.02 release of V36.

SEE ALSO

FreeDeviceProc(), DeviceProc(), AssignLock(), AssignLate(), AssignPath()

## 1.64 dos.library/GetFileSysTask

### NAME

GetFileSysTask -- Returns the default filesystem for the process (V36)

### SYNOPSIS

```
port = GetFileSysTask()  
D0
```

```
struct MsgPort *GetFileSysTask(void)
```

### FUNCTION

Returns the default filesystem task's port (pr\_FileSystemTask) for the current process.

### RESULT

port - The pr\_MsgPort of the filesystem, or NULL.

### SEE ALSO

SetFileSysTask(), Open()

## 1.65 dos.library/GetProgramDir

### NAME

GetProgramDir -- Returns a lock on the directory the program was loaded from (V36)

### SYNOPSIS

```
lock = GetProgramDir()  
D0
```

```
BPTR GetProgramDir(void)
```

### FUNCTION

Returns a shared lock on the directory the program was loaded from. This can be used for a program to find data files, etc, that are stored with the program, or to find the program file itself. NULL returns are valid, and may occur, for example, when running a program from the resident list. You should NOT unlock the lock.

### RESULT

lock - A lock on the directory the current program was loaded from, or NULL if loaded from resident list, etc.

### BUGS

Should return a lock for things loaded via resident. Perhaps should return currentdir if NULL.

### SEE ALSO

SetProgramDir(), Open()

---

## 1.66 dos.library/GetProgramName

### NAME

GetProgramName -- Returns the current program name (V36)

### SYNOPSIS

```
success = GetProgramName(buf, len)
D0                      D1    D2
```

```
BOOL GetProgramName(char *, LONG)
```

### FUNCTION

Extracts the program name from the CLI structure and puts it into the buffer. If the buffer is too small, the name is truncated present, a null string is returned in the buffer, and failure from the call (with IoErr() == ERROR\_OBJECT\_WRONG\_TYPE);

### INPUTS

buf        - Buffer to hold extracted name  
len        - Number of bytes of space in buffer

### RESULT

success - Success/failure indicator

### SEE ALSO

SetProgramName()

## 1.67 dos.library/GetPrompt

### NAME

GetPrompt -- Returns the prompt for the current process (V36)

### SYNOPSIS

```
success = GetPrompt(buf, len)
D0                      D1    D2
```

```
BOOL GetPrompt(char *, LONG)
```

### FUNCTION

Extracts the prompt string from the CLI structure and puts it into the buffer. If the buffer is too small, the string is truncated appropriately and a failure code returned. If no CLI structure is present, a null string is returned in the buffer, and failure from the call (with IoErr() == ERROR\_OBJECT\_WRONG\_TYPE);

### INPUTS

buf        - Buffer to hold extracted prompt  
len        - Number of bytes of space in buffer

### RESULT

success - Success/failure indicator

### SEE ALSO

SetPrompt()

---

## 1.68 dos.library/GetVar

### NAME

GetVar -- Returns the value of a local or global variable (V36)

### SYNOPSIS

```
len = GetVar( name, buffer, size, flags )
D0          D1      D2      D3      D4
```

```
LONG GetVar( UBYTE *, UBYTE *, LONG, ULONG )
```

### FUNCTION

Gets the value of a local or environment variable. It is advised to only use ASCII strings inside variables, but not required. This stops putting characters into the destination when a \n is hit, unless GVF\_BINARY\_VAR is specified. (The \n is not stored in the buffer.)

### INPUTS

name - pointer to a variable name.  
buffer - a user allocated area which will be used to store the value associated with the variable.  
size - length of the buffer region in bytes.  
flags - combination of type of var to get value of (low 8 bits), and flags to control the behavior of this routine. Currently defined flags include:

```
GVF_GLOBAL_ONLY - tries to get a global env variable.
GVF_LOCAL_ONLY  - tries to get a local variable.
GVF_BINARY_VAR  - don't stop at \n
```

The default is to try to get a local variable first, then to try to get a global environment variable.

### RESULT

len - Size of environment variable. -1 indicates that the variable was not defined (if IoErr() returns ERROR\_OBJECT\_NOT\_FOUND - it returns ERROR\_BAD\_NUMBER if you specify a size of 0). If the value would overflow the user buffer, the buffer is truncated. The buffer returned is null-terminated (even if GVF\_BINARY\_VAR is used). The number of characters put in the buffer (not including '\0') is returned, and IoErr() will return the size of the variable.

### BUGS

LV\_VAR is the only type that can be global.  
Under V36, we documented (and it returned) the size of the variable, not the number of characters transferred. For V37 this was changed to the number of characters put in the buffer, and the total size of the variable is put in IoErr().

### SEE ALSO

SetVar(), DeleteVar(), FindVar(), <dos/var.h>

---

## 1.69 dos.library/Info

### NAME

Info -- Returns information about the disk

### SYNOPSIS

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

```
BOOL Info(BPTR, struct InfoData *)
```

### 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)

### RESULTS

success - boolean

### SPECIAL NOTE:

Note that InfoData structure must be longword aligned.

## 1.70 dos.library/Inhibit

### NAME

Inhibit -- Inhibits access to a filesystem (V36)

### SYNOPSIS

```
success = Inhibit(filesystem, flag)
D0          D1      D2
```

```
BOOL Inhibit(char *,LONG)
```

### FUNCTION

Sends an ACTION\_INHIBIT packet to the indicated handler. This stops all activity by the handler until uninhibited. When uninhibited, anything may have happened to the disk in the drive, or there may no longer be one.

### INPUTS

filesystem - Name of device to inhibit (with ':')  
flag - New status. DOSTRUE = inhibited, FALSE = uninhibited

### RESULT

success - Success/failure indicator

### SEE ALSO

---

## 1.71 dos.library/Input

### NAME

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

### SYNOPSIS

```
file = Input()
D0
```

```
BPTR Input(void)
```

### FUNCTION

Input() is used to identify the initial input stream allocated when the program was initiated. Never close the filehandle returned by Input!

### RESULTS

file - BCPL pointer to a file handle

### SEE ALSO

Output(), SelectInput()

## 1.72 dos.library/InternalLoadSeg

### NAME

InternalLoadSeg -- Low-level load routine (V36)

### SYNOPSIS

```
seglist = InternalLoadSeg(fh,table,functionarray,stack)
D0                      D0  A0          A1          A2
```

```
BPTR InternalLoadSeg(BPTR,BPTR, LONG *,LONG *)
```

### FUNCTION

Loads from fh. Table is used when loading an overlay, otherwise should be NULL. Functionarray is a pointer to an array of functions. Note that the current Seek position after loading may be at any point after the last hunk loaded. The filehandle will not be closed. If a stacksize is encoded in the file, the size will be stuffed in the LONG pointed to by stack. This LONG should be initialized to your default value: InternalLoadSeg() will not change it if no stacksize is found. Clears unused portions of Code and Data hunks (as well as BSS hunks). (This also applies to LoadSeg() and NewLoadSeg()).

If the file being loaded is an overlaid file, this will return -(seglist). All other results will be positive.

NOTE to overlay users: InternalLoadSeg() does NOT return seglist in both D0 and D1, as LoadSeg does. The current ovs.asm uses LoadSeg(), and assumes returns are in D1. We will support this for LoadSeg() ONLY.

### INPUTS

fh - Filehandle to load from.

```

table          - When loading an overlay, otherwise ignored.
functionarray - Array of function to be used for read, alloc, and free.
  FuncTable[0] -> Actual = ReadFunc(readhandle,buffer,length),DOSBase
                  D0          D1          A0          D0          A6
  FuncTable[1] -> Memory = AllocFunc(size,flags), Execbase
                  D0          D0          D1          a6
  FuncTable[2] -> FreeFunc(memory,size), Execbase
                  A1          D0          a6
stack          - Pointer to storage (ULONG) for stacksize.

```

#### RESULT

```

seglist        - Seglist loaded or NULL. NOT returned in D1!

```

#### BUGS

```

Really should use tags.

```

#### SEE ALSO

```

LoadSeg(), UnLoadSeg(), NewLoadSeg(), InternalUnLoadSeg()

```

## 1.73 dos.library/InternalUnLoadSeg

#### NAME

```

InternalUnLoadSeg -- Unloads a seglist loaded with InternalLoadSeg()
                    (V36)

```

#### SYNOPSIS

```

success = InternalUnLoadSeg(seglist,FreeFunc)
          D0          D1          A1

```

```

BOOL InternalUnLoadSeg(BPTR,void (*)(char *,ULONG))

```

#### FUNCTION

```

Unloads a seglist using freefunc to free segments. Freefunc is called
as for InternalLoadSeg. NOTE: will call Close() for overlaid
seglists.

```

#### INPUTS

```

seglist - Seglist to be unloaded
FreeFunc - Function called to free memory

```

#### RESULT

```

success - returns whether everything went OK (since this may close
files). Also returns FALSE if seglist was NULL.

```

#### BUGS

```

Really should use tags

```

#### SEE ALSO

```

LoadSeg(), UnLoadSeg(), InternalLoadSeg(), NewUnLoadSeg(), Close()

```

## 1.74 dos.library/loErr

---



## NAME

IoErr -- Return extra information from the system

## SYNOPSIS

```
error = IoErr()  
D0
```

```
LONG IoErr(void)
```

## FUNCTION

Most I/O routines return zero to indicate an error. When this happens (or whatever the defined error return for the routine) this routine may be called to determine more information. It is also used in some routines to pass back a secondary result.

Note: there is no guarantee as to the value returned from IoErr() after a successful operation, unless to specified by the routine.

## RESULTS

error - integer

## SEE ALSO

Fault(), PrintFault(), SetIoErr()

## 1.75 dos.library/IsFileSystem

## NAME

IsFileSystem -- returns whether a Dos handler is a filesystem (V36)

## SYNOPSIS

```
result = IsFileSystem(name)  
D0 D1
```

```
BOOL IsFileSystem(char *)
```

## FUNCTION

Returns whether the device is a filesystem or not. A filesystem supports separate files storing information. It may also support sub-directories, but is not required to. If the filesystem doesn't support this new packet, IsFileSystem() will use Lock(":",...) as an indicator.

## INPUTS

name - Name of device in question, with trailing ':'.

## RESULT

result - Flag to indicate if device is a file system

## SEE ALSO

Lock()

## 1.76 dos.library/IsInteractive

### NAME

IsInteractive -- Discover whether a file is "interactive"

### SYNOPSIS

```
status = IsInteractive( file )
D0                      D1
```

```
BOOL IsInteractive(BPTR)
```

### 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

### RESULTS

status - boolean

### SEE ALSO

## 1.77 dos.library/LoadSeg

### NAME

LoadSeg -- Scatterload a loadable file into memory

### SYNOPSIS

```
seglist = LoadSeg( name )
D0                      D1
```

```
BPTR LoadSeg(char *)
```

### FUNCTION

The file 'name' should be a load module produced by the linker. LoadSeg() scatterloads 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. There can be any number of segments in a file. All necessary relocation is handled by LoadSeg().

In the event of an error any blocks loaded will be unloaded and a NULL 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

### RESULTS

seglist - BCPL pointer to a seglist

---

SEE ALSO

UnLoadSeg(), InternalLoadSeg(), InternalUnLoadSeg(), CreateProc(),  
CreateNewProc(), NewLoadSeg().

## 1.78 dos.library/Lock

NAME

Lock -- Lock a directory or file

SYNOPSIS

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

BPTR Lock(char \*, LONG)

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. Do not use random values for mode.

If Lock() fails (that is, if it cannot obtain a filing system lock on the file or directory) it returns a zero.

Tricky assumptions about the internal format of a lock are unwise, as are any attempts to use the fl\_Link or fl\_Access fields.

INPUTS

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

RESULTS

lock - BCPL pointer to a lock

SEE ALSO

UnLock(), DupLock(), ChangeMode(), NameFromLock(), DupLockFromFH()

## 1.79 dos.library/LockDosList

NAME

LockDosList -- Locks the specified Dos Lists for use (V36)

SYNOPSIS

```
dlist = LockDosList(flags)
D0          D1
```

struct DosList \*LockDosList(ULONG)

FUNCTION

---

Locks the dos device list in preparation to walk the list. If the list is 'busy' then this routine will not return until it is available. This routine "nests": you can call it multiple times, and then must unlock it the same number of times. The dlist returned is NOT a valid entry: it is a special value. Note that for 1.3 compatibility, it also does a Forbid() - this will probably be removed at some future time. The 1.3 Forbid() locking of this list had some race conditions. The pointer returned by this is NOT an actual DosList pointer - you should use one of the other DosEntry calls to get actual pointers to DosList structures (such as NextDosEntry()), passing the value returned by LockDosList() as the dlist value.

Note for handler writers: you should never call this function with LDF\_WRITE, since it can deadlock you (if someone has it read-locked and they're trying to send you a packet). Use AttemptLockDosList() instead, and effectively busy-wait with delays for the list to be available. The other option is that you can spawn a process to add the entry safely.

As an example, here's how you can search for all volumes of a specific name and do something with them:

2.0 way:

```
dl = LockDosList(LDF_VOLUMES|LDF_READ);
while (dl = FindDosEntry(dl,name,LDF_VOLUMES))
{
    Add to list of volumes to process or break out of
    the while loop.
    (You could try using it here, but I advise
    against it for compatibility reasons if you
    are planning on continuing to examine the list.)
}

process list of volumes saved above, or current entry if
you're only interested in the first one of that name.

UnLockDosList(); /* must not use dl after this! */
```

1.3/2.0 way:

```
if (version >= 36)
    dl = LockDosList(LDF_VOLUMES|LDF_READ);
else {
    Forbid();
    /* tricky! note dol_Next is at offset 0! */
    dl = &(...->di_DeviceList);
}

while (version >= 36 ?
        dl = FindDosEntry(dl,name,LDF_VOLUMES) :
        dl = yourfindentry(dl,name,DLT_VOLUME))
{
    Add to list of volumes to process, or break out of
    the while loop.
    Do NOT lock fool/foo2 here if you will continue
```

```

        to examine the list - it breaks the forbid
        and the list may change on you.
    }

    process list of volumes saved above, or current entry if
    you're only interested in the first one of that name.

    if (version >= 36)
        UnLockDosList();
    else
        Permit();
    /* must not use dl after this! */
    ...

    struct DosList *
    yourfindentry (struct DosList *dl, char *name, type)
    {
        /* tricky - depends on dol_Next being at offset 0,
        and the initial ptr being a ptr to di_DeviceList! */
        while (dl = dl->dol_Next)
        {
            if (dl->dol_Type == type &&
                stricmp(name,BADDR(dl->dol_Name)+1) == 0)
            {
                break;
            }
        }
        return dl;
    }

```

#### INPUTS

flags - Flags stating which types of nodes you want to lock.

#### RESULT

dlist - Pointer to the head of the list. NOT a valid node!

#### SEE ALSO

AttemptLockDosList(), UnLockDosList(), Forbid(), NextDosEntry()

## 1.80 dos.library/LockRecord

#### NAME

LockRecord -- Locks a portion of a file (V36)

#### SYNOPSIS

```

success = LockRecord(fh,offset,length,mode,timeout)
D0          D1    D2    D3    D4    D5

```

```

ULONG LockRecord(BPTR,ULONG,ULONG,ULONG,ULONG)

```

#### FUNCTION

This locks a portion of a file for exclusive access. Timeout is how long to wait in ticks (1/50 sec) for the record to be available.

Valid modes are:

```

REC_EXCLUSIVE
REC_EXCLUSIVE_IMMED
REC_SHARED
REC_SHARED_IMMED

```

For the IMMED modes, the timeout is ignored.

Record locks are tied to the filehandle used to create them. The same filehandle can get any number of exclusive locks on the same record, for example. These are cooperative locks, they only affect other people calling LockRecord().

#### INPUTS

```

fh      - File handle for which to lock the record
offset  - Record start position
length  - Length of record in bytes
mode    - Type of lock requester
timeout - Timeout interval in ticks. 0 is legal.

```

#### RESULT

```

success - Success or failure

```

#### BUGS

In 2.0 through 2.02 (V36), LockRecord() only worked in the ramdisk. Attempting to lock records on the disk filesystem causes a crash. This was fixed for V37.

#### SEE ALSO

```

LockRecords(), UnlockRecord(), UnlockRecords()

```

## 1.81 dos.library/LockRecords

#### NAME

```

LockRecords -- Lock a series of records (V36)

```

#### SYNOPSIS

```

success = LockRecords(record_array, timeout)
D0      D1      D2

```

```

BOOL LockRecords(struct RecordLock *, ULONG)

```

#### FUNCTION

This locks several records within a file for exclusive access. Timeout is how long to wait in ticks for the records to be available. The wait is applied to each attempt to lock each record in the list. It is recommended that you always lock a set of records in the same order to reduce possibilities of deadlock.

The array of RecordLock structures is terminated by an entry with rec\_FH of NULL.

#### INPUTS

```

record_array - List of records to be locked
timeout      - Timeout interval. 0 is legal

```

#### RESULT

success        - Success or failure

#### BUGS

See LockRecord()

#### SEE ALSO

LockRecord(), UnLockRecord(), UnLockRecords()

## 1.82 dos.library/MakeDosEntry

#### NAME

MakeDosEntry -- Creates a DosList structure (V36)

#### SYNOPSIS

```
newdlist = MakeDosEntry(name, type)
D0                D1      D2
```

```
struct DosList *MakeDosEntry(UBYTE *, LONG)
```

#### FUNCTION

Create a DosList structure, including allocating a name and correctly null-terminating the BSTR. It also sets the dol\_Type field, and sets all other fields to 0. This routine should be eliminated and replaced by a value passed to AllocDosObject()!

#### INPUTS

name - name for the device/volume/assign node.  
type - type of node.

#### RESULT

newdlist - The new device entry or NULL.

#### SEE ALSO

AddDosEntry(), RemDosEntry(), FindDosEntry(), LockDosList(),  
NextDosEntry(), FreeDosEntry()

## 1.83 dos.library/MakeLink

#### NAME

MakeLink -- Creates a filesystem link (V36)

#### SYNOPSIS

```
success = MakeLink( name, dest, soft )
D0                D1      D2      D3
```

```
BOOL MakeLink( UBYTE *, LONG, LONG )
```

#### FUNCTION

Create a filesystem link from 'name' to dest. For "soft-links", dest is a pointer to a null-terminated path string. For "hard-links", dest is a lock (BPTR). 'soft' is FALSE for hard-links, non-zero otherwise.

Soft-links are resolved at access time by a combination of the filesystem (by returning `ERROR_IS_SOFT_LINK` to dos), and by Dos (using `ReadLink()` to resolve any links that are hit).

Hard-links are resolved by the filesystem in question. A series of hard-links to a file are all equivalent to the file itself. If one of the links (or the original entry for the file) is deleted, the data remains until there are no links left.

#### INPUTS

name - Name of the link to create  
dest - CPTR to path string, or BPTR lock  
soft - FALSE for hard-links, non-zero for soft-links

#### RESULT

Success - boolean

#### BUGS

In V36, soft-links didn't work in the ROM filesystem. This was fixed for V37.

#### SEE ALSO

`ReadLink()`, `Open()`, `Lock()`

## 1.84 dos.library/MatchEnd

#### NAME

MatchEnd -- Free storage allocated for `MatchFirst()`/`MatchNext()` (V36)

#### SYNOPSIS

```
MatchEnd(AnchorPath)
        D1
```

```
VOID MatchEnd(struct AnchorPath *)
```

#### FUNCTION

Return all storage associated with a given search.

#### INPUTS

AnchorPath - Anchor used for `MatchFirst()`/`MatchNext()`  
MUST be longword aligned!

#### SEE ALSO

`MatchFirst()`, `ParsePattern()`, `Examine()`, `CurrentDir()`, `Examine()`,  
`MatchNext()`, `ExNext()`, <dos/dosasl.h>

## 1.85 dos.library/MatchFirst

#### NAME

MatchFirst -- Finds file that matches pattern (V36)

---



## SYNOPSIS

```
error = MatchFirst(pat, AnchorPath)
D0                      D1          D2
```

```
LONG MatchFirst(UBYTE *, struct AnchorPath *)
```

## FUNCTION

Locates the first file or directory that matches a given pattern. MatchFirst() is passed your pattern (you do not pass it through ParsePattern() - MatchFirst() does that for you), and the control structure. MatchFirst() normally initializes your AnchorPath structure for you, and returns the first file that matched your pattern, or an error. Note that MatchFirst()/MatchNext() are unusual for Dos in that they return 0 for success, or the error code (see <dos/dos.h>), instead of the application getting the error code from IoErr().

When looking at the result of MatchFirst()/MatchNext(), the ap\_Info field of your AnchorPath has the results of an Examine() of the object. You normally get the name of the object from fib\_FileName, and the directory it's in from ap\_Current->an\_Lock. To access this object, normally you would temporarily CurrentDir() to the lock, do an action to the file/dir, and then CurrentDir() back to your original directory. This makes certain you affect the right object even when two volumes of the same name are in the system. You can use ap\_Buf (with ap\_Strlen) to get a name to report to the user.

To initialize the AnchorPath structure (particularly when reusing it), set ap\_BreakBits to the signal bits (CDEF) that you want to take a break on, or NULL, if you don't want to convenience the user. ap\_Flags should be set to any flags you need or all 0's otherwise. ap\_FoundBreak should be cleared if you'll be using breaks.

If you want to have the FULL PATH NAME of the files you found, allocate a buffer at the END of this structure, and put the size of it into ap\_Strlen. If you don't want the full path name, make sure you set ap\_Strlen to zero. In this case, the name of the file, and stats are available in the ap\_Info, as per usual.

Then call MatchFirst() and then afterwards, MatchNext() with this structure. You should check the return value each time (see below) and take the appropriate action, ultimately calling MatchEnd() when there are no more files or you are done. You can tell when you are done by checking for the normal AmigaDOS return code ERROR\_NO\_MORE\_ENTRIES.

Note: patterns with trailing slashes may cause MatchFirst()/MatchNext() to return with an ap\_Current->an\_Lock on the object, and a filename of the empty string ("").

See ParsePattern() for more information on the patterns.

## INPUTS

```
pat          - Pattern to search for
AnchorPath - Place holder for search. MUST be longword aligned!
```

## RESULT

error - 0 for success or error code. (Opposite of most Dos calls!)

#### BUGS

In V36, there were a number of bugs with MatchFirst()/MatchNext(). One was that if you entered a directory with a name like "df0:l" using DODIR, it would re-lock the full string "df0:l", which can cause problems if the disk has changed. It also had problems with patterns such as #?/abc/def - the ap\_Current->an\_Lock would not be on the directory def is found in. ap\_Buf would be correct, however. It had similar problems with patterns with trailing slashes. These have been fixed for V37 and later.

A bug that has not been fixed for V37 concerns a pattern of a single directory name (such as "l"). If you enter such a directory via DODIR, it re-locks l relative to the current directory. Thus you must not change the current directory before calling MatchNext() with DODIR in that situation. If you aren't using DODIR to enter directories you can ignore this. This may be fixed in some upcoming release.

#### SEE ALSO

MatchNext(), ParsePattern(), Examine(), CurrentDir(), Examine(), MatchEnd(), ExNext(), <dos/dosasl.h>

## 1.86 dos.library/MatchNext

#### NAME

MatchNext - Finds the next file or directory that matches pattern (V36)

#### SYNOPSIS

```
error = MatchNext(AnchorPath)
D0                      D1
```

```
LONG MatchNext(struct AnchorPath *)
```

#### FUNCTION

Locates the next file or directory that matches a given pattern. See <dos/dosasl.h> for more information. Various bits in the flags allow the application to control the operation of MatchNext().

See MatchFirst() for other notes.

#### INPUTS

AnchorPath - Place holder for search. MUST be longword aligned!

#### RESULT

error - 0 for success or error code. (Opposite of most Dos calls)

#### BUGS

See MatchFirst().

#### SEE ALSO

MatchFirst(), ParsePattern(), Examine(), CurrentDir(), Examine(), MatchEnd(), ExNext(), <dos/dosasl.h>

## 1.87 dos.library/MatchPattern

### NAME

MatchPattern -- Checks for a pattern match with a string (V36)

### SYNOPSIS

```
match = MatchPattern(pat, str)
D0                                D1    D2
```

```
BOOL MatchPattern(UBYTE *, UBYTE *)
```

### FUNCTION

Checks for a pattern match with a string. The pattern must be a tokenized string output by ParsePattern(). This routine is case-sensitive.

NOTE: this routine is highly recursive. You must have at least 1500 free bytes of stack to call this (it will cut off it's recursion before going any deeper than that and return failure). That's currently enough for about 100 levels deep of #, (, ~, etc.

### INPUTS

pat - Special pattern string to match as returned by ParsePattern()  
str - String to match against given pattern

### RESULT

match - success or failure of pattern match. On failure, IoErr() will return 0 or ERROR\_TOO\_MANY\_LEVELS (starting with V37 - before that there was no stack checking).

### SEE ALSO

ParsePattern(), MatchPatternNoCase(), MatchFirst(), MatchNext()

## 1.88 dos.library/MatchPatternNoCase

### NAME

MatchPatternNoCase -- Checks for a pattern match with a string (V37)

### SYNOPSIS

```
match = MatchPatternNoCase(pat, str)
D0                                D1    D2
```

```
BOOL MatchPatternNoCase(UBYTE *, UBYTE *)
```

### FUNCTION

Checks for a pattern match with a string. The pattern must be a tokenized string output by ParsePatternNoCase(). This routine is case-insensitive.

NOTE: this routine is highly recursive. You must have at least 1500 free bytes of stack to call this (it will cut off it's recursion before going any deeper than that and return failure). That's currently enough for about 100 levels deep of #, (, ~, etc.

---

## INPUTS

pat - Special pattern string to match as returned by  
ParsePatternNoCase()  
str - String to match against given pattern

## RESULT

match - success or failure of pattern match. On failure,  
IoErr() will return 0 or ERROR\_TOO\_MANY\_LEVELS (starting  
with V37 - before that there was no stack checking).

## SEE ALSO

ParsePatternNoCase(), MatchPattern(), MatchFirst(), MatchNext()

## 1.89 dos.library/MaxCli

## NAME

MaxCli -- returns the highest CLI process number possibly in use (V36)

## SYNOPSIS

```
number = MaxCli()
D0
```

```
LONG MaxCli(void)
```

## FUNCTION

Returns the highest CLI number that may be in use. CLI numbers are reused, and are usually as small as possible. To find all CLIs, scan using FindCliProc() from 1 to MaxCLI(). The number returned by MaxCli() may change as processes are created and destroyed.

## RESULT

number - The highest CLI number that may be in use.

## SEE ALSO

FindCliProc(), Cli()

## 1.90 dos.library/NameFromFH

## NAME

NameFromFH -- Get the name of an open filehandle (V36)

## SYNOPSIS

```
success = NameFromFH(fh, buffer, len)
D0                D1    D2    D3
```

```
BOOL NameFromFH(BPTR, char *, LONG)
```

## FUNCTION

Returns a fully qualified path for the filehandle. This routine is guaranteed not to write more than len characters into the buffer. The name will be null-terminated. See NameFromLock() for more information.

## INPUTS

fh - Lock of object to be examined.  
buffer - Buffer to store name.  
len - Length of buffer.

## RESULT

success - Success/failure indicator.

## SEE ALSO

NameFromLock()

## 1.91 dos.library/NameFromLock

## NAME

NameFromLock -- Returns the name of a locked object (V36)

## SYNOPSIS

```
success = NameFromLock(lock, buffer, len)
D0                      D1      D2      D3
```

```
BOOL NameFromLock(BPTR, char *, LONG)
```

## FUNCTION

Returns a fully qualified path for the lock. This routine is guaranteed not to write more than len characters into the buffer. The name will be null-terminated. NOTE: if the volume is not mounted, the system will request it (unless of course you set pr\_WindowPtr to -1). If the volume is not mounted or inserted, it will return an error. If the lock passed in is NULL, "SYS:" will be returned. If the buffer is too short, an error will be returned, and IoErr() will return ERROR\_LINE\_TOO\_LONG.

## INPUTS

lock - Lock of object to be examined.  
buffer - Buffer to store name.  
len - Length of buffer.

## RESULT

success - Success/failure indicator.

## BUGS

Should return the name of the boot volume instead of SYS: for a NULL lock.

## SEE ALSO

NameFromFH(), Lock()

## 1.92 dos.library/NewLoadSeg

## NAME

NewLoadSeg -- Improved version of LoadSeg for stacksizes (V36)

---

## SYNOPSIS

```

seglist = NewLoadSeg(file, tags)
D0                      D1      D2

BPTR NewLoadSeg(UBYTE *, struct TagItem *)

seglist = NewLoadSegTagList(file, tags)
D0                      D1      D2

BPTR NewLoadSegTagList(UBYTE *, struct TagItem *)

seglist = NewLoadSegTags(file, ...)

BPTR NewLoadSegTags(UBYTE *, ...)
```

## FUNCTION

Does a LoadSeg on a file, and takes additional actions based on the tags supplied.

Clears unused portions of Code and Data hunks (as well as BSS hunks). (This also applies to InternalLoadSeg() and LoadSeg()).

NOTE to overlay users: NewLoadSeg() does NOT return seglist in both D0 and D1, as LoadSeg does. The current ovs.asm uses LoadSeg(), and assumes returns are in D1. We will support this for LoadSeg() ONLY.

## INPUTS

file - Filename of file to load  
tags - pointer to tagitem array

## RESULT

seglist - Seglist loaded, or NULL

## BUGS

No tags are currently defined.

## SEE ALSO

LoadSeg(), UnLoadSeg(), InternalLoadSeg(), InternalUnLoadSeg()

## 1.93 dos.library/NextDosEntry

## NAME

NextDosEntry -- Get the next Dos List entry (V36)

## SYNOPSIS

```

newdlist = NextDosEntry(dlist, flags)
D0                      D1      D2

struct DosList *NextDosEntry(struct DosList *, ULONG)
```

## FUNCTION

Find the next Dos List entry of the right type. You MUST have locked the types you're looking for. Returns NULL if there are no more of that type in the list.

## INPUTS

dlist        - The current device entry.  
 flags       - What type of entries to look for.

## RESULT

newdlist - The next device entry of the right type or NULL.

## SEE ALSO

AddDosEntry(), RemDosEntry(), FindDosEntry(), LockDosList(),  
 MakeDosEntry(), FreeDosEntry()

## 1.94 dos.library/Open

## NAME

Open -- Open a file for input or output

## SYNOPSIS

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

```
BPTR Open(char *, LONG)
```

## 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 a file with an shared lock, but creates it if it didn't exist. Open types are documented in the <dos/dos.h> or <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. Note that as of V36, "\*" is obsolete, and CONSOLE: should be used instead.

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

## RESULTS

file - BCPL pointer to a file handle

## SEE ALSO

Close(), ChangeMode(), NameFromFH(), ParentOfFH(), ExamineFH()

## 1.95 dos.library/OpenFromLock

## NAME

OpenFromLock -- Opens a file you have a lock on (V36)

## SYNOPSIS

```
fh = OpenFromLock(lock)
D0                                D1
```

BPTR OpenFromLock (BPTR)

## FUNCTION

Given a lock, this routine performs an open on that lock. If the open succeeds, the lock is (effectively) relinquished, and should not be UnLock()ed or used. If the open fails, the lock is still usable. The lock associated with the file internally is of the same access mode as the lock you gave up - shared is similar to MODE\_OLDFILE, exclusive is similar to MODE\_NEWFILE.

## INPUTS

lock - Lock on object to be opened.

## RESULT

fh - Newly opened file handle or NULL for failure

## BUGS

In the original V36 autodocs, this was shown (incorrectly) as taking a Mode parameter as well. The prototypes and pragmas were also wrong.

## SEE ALSO

Open(), Close(), Lock(), UnLock()

## 1.96 dos.library/Output

## NAME

Output -- Identify the programs' initial output file handle

## SYNOPSIS

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

BPTR Output (void)

## FUNCTION

Output() is used to identify the initial output stream allocated when the program was initiated. Never close the filehandle returned by Output().

## RESULTS

file - BCPL pointer to a file handle

## SEE ALSO

Input()



## 1.97 dos.library/ParentDir

### NAME

ParentDir -- Obtain the parent of a directory or file

### SYNOPSIS

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

```
BPTR ParentDir(BPTR)
```

### 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

### RESULTS

newlock - BCPL pointer to a lock

### SEE ALSO

Lock(), DupLock(), UnLock(), ParentOfFH(), DupLockFromFH()

## 1.98 dos.library/ParentOfFH

### NAME

ParentOfFH -- returns a lock on the parent directory of a file (V36)

### SYNOPSIS

```
lock = ParentOfFH(fh)
D0                      D1
```

```
BPTR ParentOfFH(BPTR)
```

### FUNCTION

Returns a shared lock on the parent directory of the filehandle.

### INPUTS

fh - Filehandle you want the parent of.

### RESULT

lock - Lock on parent directory of the filehandle or NULL for failure.

### SEE ALSO

Parent(), Lock(), UnLock() DupLockFromFH()

---

## 1.99 dos.library/ParsePattern

### NAME

ParsePattern -- Create a tokenized string for MatchPattern() (V36)

### SYNOPSIS

```
IsWild = ParsePattern(Source, Dest, DestLength)
d0          D1      D2      D3
```

```
LONG ParsePattern(UBYTE *, UBYTE *, LONG)
```

### FUNCTION

Tokenizes a pattern, for use by MatchPattern(). Also indicates if there are any wildcards in the pattern (i.e. whether it might match more than one item). Note that Dest must be at least 2 times as large as Source plus bytes to be (almost) 100% certain of no buffer overflow. This is because each input character can currently expand to 2 tokens (with one exception that can expand to 3, but only once per string). Note: this implementation may change in the future, so you should handle error returns in all cases, but the size above should still be a reasonable upper bound for a buffer allocation.

The patterns are fairly extensive, and approximate some of the ability of Unix/grep "regular expression" patterns. Here are the available tokens:

|         |  |
|---------|--|
| ?       | Matches a single character.  |
| #       | Matches the following expression 0 or more times.  |
| (ab cd) | Matches any one of the items seperated by ' '.   |
| ~       | Negates the following expression. It matches all strings that do not match the expression (aka ~(foo) matches all strings that are not exactly "foo"). |
| [abc]   | Character class: matches any of the characters in the class.   |
| [~bc]   | Character class: matches any of the characters not in the class.   |
| a-z     | Character range (only within character classes).   |
| %       | Matches 0 characters always (useful in "(foo bar %)").   |
| *       | Synonym for "#?", not available by default in 2.0. Available as an option that can be turned on.   |

"Expression" in the above table means either a single character (ex: "#?"), or an alternation (ex: "#(ab|cd|ef)"), or a character class (ex: "#[a-zA-Z]").

### INPUTS

```
source      - unparsed wildcard string to search for.
dest        - output string, gets tokenized version of input.
DestLength  - length available in destination (should be at least as
              twice as large as source + 2 bytes).
```

### RESULT

```
IsWild - 1 means there were wildcards in the pattern,
        0 means there were no wildcards in the pattern,
       -1 means there was a buffer overflow or other error
```

## BUGS

Should set IoErr() to something useful (not currently set) on an error.

## SEE ALSO

ParsePatternNoCase(), MatchPattern(), MatchFirst(), MatchNext()

## 1.100 dos.library/ParsePatternNoCase

## NAME

ParsePatternNoCase -- Create a tokenized string for  
MatchPatternNoCase() (V37)

## SYNOPSIS

```
IsWild = ParsePatternNoCase(Source, Dest, DestLength)
d0                      D1      D2      D3
```

```
LONG ParsePatternNoCase(UBYTE *, UBYTE *, LONG)
```

## FUNCTION

Tokenizes a pattern, for use by MatchPatternNoCase(). Also indicates if there are any wildcards in the pattern (i.e. whether it might match more than one item). Note that Dest must be at least 2 times as large as Source plus 2 bytes.

For a description of the wildcards, see ParsePattern().

## INPUTS

source        - unparsed wildcard string to search for.  
dest          - output string, gets tokenized version of input.  
DestLength - length available in destination (should be at least as  
             twice as large as source + 2 bytes).

## RESULT

IsWild - 1 means there were wildcards in the pattern,  
         0 means there were no wildcards in the pattern,  
         -1 means there was a buffer overflow or other error

## BUGS

Should set IoErr() to something useful (not currently set) on an error.

## SEE ALSO

ParsePattern(), MatchPatternNoCase(), MatchFirst(), MatchNext()

## 1.101 dos.library/PathPart

## NAME

PathPart -- Returns a pointer to the end of the next-to-last (V36)  
          component of a path.

## SYNOPSIS

```
fileptr = PathPart( path )
D0                                     D1
```

```
UBYTE *PathPart( UBYTE * )
```

#### FUNCTION

This function returns a pointer to the character after the next-to-last component of a path specification, which will normally be the directory name. If there is only one component, it returns a pointer to the beginning of the string. The only real difference between this and FilePart() is the handling of '/'.

#### INPUTS

path - pointer to an path string. May be relative to the current directory or the current disk.

#### RESULT

fileptr - pointer to the end of the next-to-last component of the path.

#### EXAMPLE

PathPart("xxx:yyy/zzz/qqq") would return a pointer to the last '/'.  
 PathPart("xxx:yyy") would return a pointer to the first 'y').

#### SEE ALSO

FilePart(), AddPart()

## 1.102 dos.library/PrintFault

#### NAME

PrintFault -- Returns the text associated with a DOS error code (V36)

#### SYNOPSIS

```
success = PrintFault(code, header)
D0                                     D1      D2
```

```
BOOL PrintFault(LONG, UBYTE *)
```

#### FUNCTION

This routine obtains the error message text for the given error code. This is similar to the Fault() function, except that the output is written to the default output channel with buffered output. The value returned by IoErr() is set to the code passed in.

#### INPUTS

code - Error code  
 header - header to output before error text

#### RESULT

success - Success/failure code.

#### SEE ALSO

IoErr(), Fault(), SetIoErr(), Output(), Fputs()

## 1.103 dos.library/PutStr

### NAME

PutStr -- Writes a string the the default output (buffered) (V36)

### SYNOPSIS

```
error = PutStr(str)
D0          D1
```

```
LONG PutStr(UBYTE *)
```

### FUNCTION

This routine writes an unformatted string to the default output. No newline is appended to the string and any error is returned. This routine is buffered.

### INPUTS

str - Null-terminated string to be written to default output

### RESULT

error - 0 for success, -1 for any error. NOTE: this is opposite most Dos function returns!

### SEE ALSO

Fputs(), FPutC(), FWrite(), WriteChars()

## 1.104 dos.library/Read

### NAME

Read -- Read bytes of data from a file

### SYNOPSIS

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

```
LONG Read(BPTR, void *, LONG)
```

### 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.

Note: this is an unbuffered routine (the request is passed directly to the filesystem.) Buffered I/O is more efficient for small reads and writes; see FGetC().

---

## INPUTS

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

## RESULTS

actualLength - integer

## SEE ALSO

Open(), Close(), Write(), Seek(), FGetC()

## 1.105 dos.library/ReadArgs

## NAME

ReadArgs - Parse the command line input (V36)

## SYNOPSIS

```
result = ReadArgs(template, array, rdargs)
D0                      D1          D2          D3
```

```
struct RDArgs * ReadArgs(UBYTE *, LONG *, struct RDArgs *)
```

## FUNCTION

Parses and argument string according to a template. Normally gets the arguments by reading buffered IO from Input(), but also can be made to parse a string. MUST be matched by a call to FreeArgs().

ReadArgs() parses the commandline according to a template that is passed to it. This specifies the different command-line options and their types. A template consists of a list of options. Options are named in "full" names where possible (for example, "Quick" instead of "Q"). Abbreviations can also be specified by using "abbrev=option" (for example, "Q=Quick").

Options in the template are separated by commas. To get the results of ReadArgs(), you examine the array of longwords you passed to it (one entry per option in the template). This array should be cleared (or initialized to your default values) before passing to ReadArgs(). Exactly what is put in a given entry by ReadArgs() depends on the type of option. The default is a string (a sequence of non-whitespace characters, or delimited by quotes, which will be stripped by ReadArgs()), in which case the entry will be a pointer.

Options can be followed by modifiers, which specify things such as the type of the option. Modifiers are specified by following the option with a '/' and a single character modifier. Multiple modifiers can be specified by using multiple '/'s. Valid modifiers are:

/S - Switch. This is considered a boolean variable, and will be set if the option name appears in the command-line. The entry is the boolean (0 for not set, non-zero for set).

/K - Keyword. This means that the option will not be filled unless the keyword appears. For example if the template is "Name/K", then unless "Name=<string>" or "Name <string>" appears in the

command line, Name will not be filled.

- /N - Number. This parameter is considered a decimal number, and will be converted by ReadArgs. If an invalid number is specified, an error will be returned. The entry will be a pointer to the longword number (this is how you know if a number was specified).
- /T - Toggle. This is similar to a switch, but when specified causes the boolean value to "toggle". Similar to /S.
- /A - Required. This keyword must be given a value during command-line processing, or an error is returned.
- /F - Rest of line. If this is specified, the entire rest of the line is taken as the parameter for the option, even if other option keywords appear in it.
- /M - Multiple strings. This means the argument will take any number of strings, returning them as an array of strings. Any arguments not considered to be part of another option will be added to this option. Only one /M should be specified in a template. Example: for a template "Dir/M,All/S" the command-line "foo bar all qwe" will set the boolean "all", and return an array consisting of "foo", "bar", and "qwe". The entry in the array will be a pointer to an array of string pointers, the last of which will be NULL.

There is an interaction between /M parameters and /A parameters. If there are unfilled /A parameters after parsing, it will grab strings from the end of a previous /M parameter list to fill the /A's. This is used for things like Copy ("From/A/M,To/A").

ReadArgs() returns a struct RArgs if it succeeds. This serves as an "anchor" to allow FreeArgs() to free the associated memory. You can also pass in a struct RArgs to control the operation of ReadArgs() (normally you pass NULL for the parameter, and ReadArgs() allocates one for you). This allows providing different sources for the arguments, providing your own string buffer space for temporary storage, and extended help text. See <dos/rdargs.h> for more information on this. Note: if you pass in a struct RArgs, you must still call FreeArgs() to release storage that gets attached to it, but you are responsible for freeing the RArgs yourself.

See BUGS regarding passing in strings.

#### INPUTS

- template - formatting string
- array - array of longwords for results, 1 per template entry
- rdargs - optional rdargs structure for options. AllocDosObject should be used for allocating them if you pass one in.

#### RESULT

- result - a struct RArgs or NULL for failure.

#### BUGS

In V36, there were a couple of minor bugs with certain argument combinations (/M/N returned strings, /T didn't work, and /K and /F interacted). Also, a template with a /K before any non-switch

parameter will require the argument name to be given in order for line to be accepted (i.e. "parm/K,xyzzzy/A" would require "xyzzzy=xxxxx" in order to work - "xxxxx" would not work). If you need to avoid this for V36, put /K parameters after all non-switch parameters. These problems should be fixed for V37.

Currently (V37 and before) it requires any strings passed in to have newlines at the end of the string. This may or may not be fixed in the future.

SEE ALSO

FindArg(), ReadItem(), FreeArgs(), AllocDosObject()

## 1.106 dos.library/ReadItem

NAME

ReadItem - reads a single argument/name from command line (V36)

SYNOPSIS

```
value = ReadItem(buffer, maxchars, input)
D0          D1          D2          D3
```

```
LONG ReadItem(UBYTE *, LONG, struct CSource *)
```

FUNCTION

Reads a "word" from either Input() (buffered), or via CSource, if it is non-NULL (see <dos/rdargs.h> for more information). Handles quoting and some '\*' substitutions (\*e and \*n) inside quotes (only). See dos/dos.h for a listing of values returned by ReadItem() (ITEM\_XXXX). A "word" is delimited by whitespace, quotes, or an EOF.

ReadItem always unread the last thing read (UnGetC(fh,-1)) so the caller can find out what the terminator was.

INPUTS

```
buffer    - buffer to store word in.
maxchars  - size of the buffer
input     - CSource input or NULL (uses FGetC(Input()))
```

RESULT

value - See <dos/dos.h> for return values.

SEE ALSO

ReadArgs(), FindArg(), UnGetC(), FGetC(), Input(), <dos/dos.h>, <dos/rdargs.h>, FreeArgs()

## 1.107 dos.library/ReadLink

NAME

ReadLink -- Reads the path for a soft filesystem link (V36)

SYNOPSIS

---



```

success = ReadLink( port, lock, path, buffer, size)
D0                      D1      D2      D3      D4      D5

```

```

BOOL ReadLink( struct MsgPort *, BPTR, UBYTE *, UBYTE *, ULONG)

```

#### FUNCTION

ReadLink() takes a lock/name pair (usually from a failed attempt to use them to access an object with packets), and asks the filesystem to find the softlink and fill buffer with the modified path string. You then start the resolution process again by calling GetDeviceProc() with the new string from ReadLink().

Soft-links are resolved at access time by a combination of the filesystem (by returning ERROR\_IS\_SOFT\_LINK to dos), and by Dos (using ReadLink() to resolve any links that are hit).

#### INPUTS

port - msgport of the filesystem  
 lock - lock this path is relative to on the filesystem  
 path - path that caused the ERROR\_IS\_SOFT\_LINK  
 buffer - pointer to buffer for new path from handler.  
 size - size of buffer.

#### RESULT

Success - boolean

#### BUGS

In V36, soft-links didn't work in the ROM filesystem. This was fixed for V37.

#### SEE ALSO

MakeLink(), Open(), Lock(), GetDeviceProc()

## 1.108 dos.library/Relabel

#### NAME

Relabel -- Change the volume name of a volume (V36)

#### SYNOPSIS

```

success = Relabel(volumename,name)
D0                      D1      D2

```

```

BOOL Relabel(char *,char *)

```

#### FUNCTION

Changes the volumename of a volume, if supported by the filesystem.

#### INPUTS

volumename - Full name of device to rename (with ':')  
 newname - New name to apply to device (without ':')

#### RESULT

success - Success/failure indicator

---

SEE ALSO

## 1.109 dos.library/RemAssignList

### NAME

RemAssignList -- Remove an entry from a multi-dir assign (V36)

### SYNOPSIS

```
success = RemAssignList(name,lock)
```

```
D0                                D1    D2
```

```
BOOL RemAssignList(char *,BPTR)
```

### FUNCTION

Removes an entry from a multi-directory assign. The entry removed is the first one for which SameLock with 'lock' returns that they are on the same object. The lock for the entry in the list is unlocked (not the entry passed in).

### INPUTS

name - Name of device to remove lock from (without trailing ':')

lock - Lock associated with the object to remove from the list

### RESULT

success - Success/failure indicator.

### SEE ALSO

Lock(), AssignLock(), AssignPath(), AssignLate(), DupLock(),  
AssignAdd(), UnLock()

## 1.110 dos.library/RemDosEntry

### NAME

RemDosEntry -- Removes a Dos List entry from it's list (V36)

### SYNOPSIS

```
success = RemDosEntry(dlist)
```

```
D0                                D1
```

```
BOOL RemDosEntry(struct DosList *)
```

### FUNCTION

This removes an entry from the Dos Device list. The memory associated with the entry is NOT freed. NOTE: you must have locked the Dos List with the appropriate flags before calling this routine. Handler writers should see the AddDosEntry() caveats about locking and use a similar workaround to avoid deadlocks.

### INPUTS

dlist - Device list entry to be removed.

### RESULT

---

success - Success/failure indicator

SEE ALSO

AddDosEntry(), FindDosEntry(), NextDosEntry(), LockDosList(),  
MakeDosEntry(), FreeDosEntry()

## 1.111 dos.library/RemSegment

NAME

RemSegment - Removes a resident segment from the resident list (V36)

SYNOPSIS

success = RemSegment(segment)  
D0 D1

BOOL RemSegment(struct Segment \*)

FUNCTION

Removes a resident segment from the Dos resident segment list,  
unloads it, and does any other cleanup required. Will only succeed  
if the seg\_UC (usecount) is 0.

INPUTS

segment - the segment to be removed

RESULT

success - success or failure.

SEE ALSO

FindSegment(), AddSegment()

## 1.112 dos.library/Rename

NAME

Rename -- Rename a directory or file

SYNOPSIS

success = Rename( oldName, newName )  
D0 D1 D2

BOOL Rename(char \*, char \*)

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

## RESULTS

success - boolean

## SEE ALSO

Relabel()

## 1.113 dos.library/ReplyPkt

## NAME

ReplyPkt -- replies a packet to the person who sent it to you (V36)

## SYNOPSIS

```
ReplyPkt(packet, result1, result2)
          D1      D2      D3
```

```
void ReplyPkt(struct DosPacket *, LONG, LONG)
```

## FUNCTION

This returns a packet to the process which sent it to you. In addition, puts your pr\_MsgPort address in dp\_Port, so using ReplyPkt() again will send the message to you. (This is used in "ping-ponging" packets between two processes). It uses result 1 & 2 to set the dp\_Res1 and dp\_Res2 fields of the packet.

## INPUTS

packet - packet to reply, assumed to set up correctly.  
 result1 - first result  
 result2 - secondary result

## SEE ALSO

DoPkt(), SendPkt(), WaitPkt(), IoErr()

## 1.114 dos.library/RunCommand

## NAME

RunCommand -- Runs a program using the current process (V36)

## SYNOPSIS

```
rc = RunCommand(seglist, stacksize, argptr, argsize)
D0      D1      D2      D3      D4
```

```
LONG RunCommand(BPTR, ULONG, char *, ULONG)
```

## FUNCTION

Runs a command on your process/cli. Seglist may be any language, including BCPL programs. Stacksize is in bytes. argptr is a null-terminated string, argsize is its length. Returns the returncode the program exited with in d0. Returns -1 if the stack couldn't be

allocated.

NOTE: the argument string MUST be terminated with a newline to work properly with ReadArgs() and other argument parsers.

RunCommand also takes care of setting up the current input filehandle in such a way that ReadArgs() can be used in the program, and restores the state of the buffering before returning. It also sets the value returned by GetArgStr(), and restores it before returning. NOTE: the setting of the argument string in the filehandle was added in V37.

It's usually appropriate to set the command name (via SetProgramName()) before calling RunCommand(). RunCommand() sets the value returned by GetArgStr() while the command is running.

#### INPUTS

seglist     - Seglist of command to run.  
stacksize   - Number of bytes to allocate for stack space  
argptr       - Pointer to argument command string.  
argsize      - Number of bytes in argument command.

#### RESULT

rc           - Return code from executed command. -1 indicates failure

#### SEE ALSO

CreateNewProc(), SystemTagList(), Execute(), GetArgStr(),  
SetProgramName(), ReadArgs()

## 1.115 dos.library/SameDevice

#### NAME

SameDevice -- Are two locks are on partitions of the device? (V37)

#### SYNOPSIS

```
same = SameDevice(lock1, lock2)
D0                   D1       D2
```

```
BOOL SameDevice( BPTR, BPTR )
```

#### FUNCTION

SameDevice() returns whether two locks refer to partitions that are on the same physical device (if it can figure it out). This may be useful in writing copy routines to take advantage of asynchronous multi-device copies.

Entry existed in V36 and always returned 0.

#### INPUTS

lock1,lock2 - locks

#### RESULT

same - whether they're on the same device as far as Dos can determine.

---

## 1.116 dos.library/SameLock

### NAME

SameLock -- returns whether two locks are on the same object (V36)

### SYNOPSIS

```
value = SameLock(lock1, lock2)
D0          D1      D2
```

```
LONG SameLock(BPTR, BPTR)
```

### FUNCTION

Compares two locks. Returns LOCK\_SAME if they are on the same object, LOCK\_SAME\_VOLUME if on different objects on the same volume, and LOCK\_DIFFERENT if they are on different volumes. Always compare for equality or non-equality with the results, in case new return values are added.

### INPUTS

lock1 - 1st lock for comparison  
lock2 - 2nd lock for comparison

### RESULT

value - LOCK\_SAME, LOCK\_SAME\_VOLUME, or LOCK\_DIFFERENT

### BUGS

Should do more extensive checks for NULL against a real lock, checking to see if the real lock is a lock on the root of the boot volume.

In V36, it would return LOCK\_SAME\_VOLUME for different volumes on the same handler. Also, LOCK\_SAME\_VOLUME was LOCK\_SAME\_HANDLER (now an obsolete define, see <dos/dos.h>).

### SEE ALSO

<dos/dos.h>

## 1.117 dos.library/Seek

### NAME

Seek -- Set the current position for reading and writing

### SYNOPSIS

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

```
LONG Seek(BPTR, LONG, LONG)
```

### 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
```

## RESULTS

oldPosition - integer

## BUGS

The V36 and V37 ROM filesystem (and V36/V37 1:fastfilesystem) returns the current position instead of -1 on an error. It sets IoErr() to 0 on success, and an error code on an error.

SEE ALSO

Read(), Write(), SetFileSize()

### 1.118 dos.library/SelectInput

## NAME

```
SelectInput -- Select a filehandle as the default input channel (V36)
```

## SYNOPSIS

```
old_fh = SelectInput(fh)
D0      D1
```

BPTR SelectInput (BPTR)

## FUNCTION

Set the current input as the default input for the process. This changes the value returned by `Input()`. `old_fh` should be closed or saved as needed.

## INPUTS

```
fh      - Newly default input handle
```

## RESULT

old\_fh - Previous default input filehandle

SEE ALSO

Input(), SelectOutput(), Output()

### 1.119 dos.library/SelectOutput

## NAME

SelectOutput -- Select a filehandle as the default input channel (V36)

## SYNOPSIS

```
old_fh = SelectOutput(fh)
D0          D1
```

BPTR SelectOutput (BPTR)

## FUNCTION

Set the current output as the default output for the process. This changes the value returned by Output(). old\_fh should be closed or saved as needed.

## INPUTS

fh - Newly desired output handle

## RESULT

old\_fh - Previous current output

## SEE ALSO

Output(), SelectInput(), Input()

## 1.120 dos.library/SendPkt

## NAME

SendPkt -- Sends a packet to a handler (V36)

## SYNOPSIS

```
SendPkt(packet, port, replyport)
D1      D2      D3
```

void SendPkt(struct DosPacket \*,struct MsgPort \*,struct MsgPort \*)

## FUNCTION

Sends a packet to a handler and does not wait. All fields in the packet must be initialized before calling this routine. The packet will be returned to replyport. If you wish to use this with WaitPkt(), use the address or your pr\_MsgPort for replyport.

## INPUTS

packet - packet to send, must be initialized and have a message.  
port - pr\_MsgPort of handler process to send to.  
replyport - MsgPort for the packet to come back to.

## NOTES

Callable from a task.

## SEE ALSO

DoPkt(), WaitPkt(), AllocDosObject(), FreeDosObject(), AbortPkt()

---



## 1.121 dos.library/SetArgStr

### NAME

SetArgStr -- Sets the arguments for the current process (V36)

### SYNOPSIS

```
oldptr = SetArgStr(ptr)
D0                      D1
```

```
UBYTE *SetArgStr(UBYTE *)
```

### FUNCTION

Sets the arguments for the current program. The ptr MUST be reset to it's original value before process exit.

### INPUTS

ptr - pointer to new argument string.

### RESULT

oldptr - the previous argument string

### SEE ALSO

GetArgStr(), RunCommand()

## 1.122 dos.library/SetComment

### NAME

SetComment -- Change a files' comment string

### SYNOPSIS

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

```
BOOL SetComment(char *, char *)
```

### 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 in the current ROM filesystem (and RAM:). Note that not all filesystems will support comments (for example, NFS usually will not), or the size of comment supported may vary.

### INPUTS

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

### RESULTS

success - boolean

### SEE ALSO

Examine(), ExNext(), SetProtection()

---

## 1.123 dos.library/SetConsoleTask

### NAME

SetConsoleTask -- Sets the default console for the process (V36)

### SYNOPSIS

```
oldport = SetConsoleTask(port)
D0                      D1
```

```
struct MsgPort *SetConsoleTask(struct MsgPort *)
```

### FUNCTION

Sets the default console task's port (pr\_ConsoleTask) for the current process.

### INPUTS

port - The pr\_MsgPort of the default console handler for the process

### RESULT

oldport - The previous ConsoleTask value.

### SEE ALSO

GetConsoleTask(), Open()

## 1.124 dos.library/SetCurrentDirName

### NAME

SetCurrentDirName -- Sets the directory name for the process (V36)

### SYNOPSIS

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

```
BOOL SetCurrentDirName(char *)
```

### FUNCTION

Sets the name for the current dir in the cli structure. If the name is too long to fit, a failure is returned, and the old value is left intact. It is advised that you inform the user of this condition. This routine is safe to call even if there is no CLI structure.

### INPUTS

name - Name of directory to be set.

### RESULT

success - Success/failure indicator

### BUGS

This clips to a fixed (1.3 compatible) size.

### SEE ALSO

GetCurrentDirName()

---

## 1.125 dos.library/SetFileDate

### NAME

SetFileDate -- Sets the modification date for a file or dir (V36)

### SYNOPSIS

```
success = SetFileDate(name, date)
D0                      D1    D2
```

```
BOOL SetFileDate(char *, struct DateStamp *)
```

### FUNCTION

Sets the file date for a file or directory. Note that for the Old File System and the Fast File System, the date of the root directory cannot be set. Other filesystems may not support setting the date for all files/directories.

### INPUTS

name - Name of object  
date - New modification date

### RESULT

success - Success/failure indication

### SEE ALSO

DateStamp(), Examine(), ExNext(), ExAll()

## 1.126 dos.library/SetFileSize

### NAME

SetFileSize -- Sets the size of a file (V36)

### SYNOPSIS

```
newsize = SetFileSize(fh, offset, mode)
D0                      D1    D2    D3
```

```
LONG SetFileSize(BPTR, LONG, LONG)
```

### FUNCTION

Changes the file size, truncating or extending as needed. Not all handlers may support this; be careful and check the return code. If the file is extended, no values should be assumed for the new bytes. If the new position would be before the filehandle's current position in the file, the filehandle will end with a position at the end-of-file. If there are other filehandles open onto the file, the new size will not leave any filehandle pointing past the end-of-file. You can check for this by looking at the new size.

Do NOT count on any specific values to be in the extended area.

### INPUTS

fh - File to be truncated/extended.  
offset - Offset from position determined by mode.  
mode - One of OFFSET\_BEGINNING, OFFSET\_CURRENT, or OFFSET\_END.

---

## RESULT

newsize - position of new end-of-file or -1 for error.

## SEE ALSO

Seek()

## 1.127 dos.library/SetFileSysTask

## NAME

SetFileSysTask -- Sets the default filesystem for the process (V36)

## SYNOPSIS

```
oldport = SetFileSysTask(port)
D0                      D1
```

```
struct MsgPort *SetFileSysTask(struct MsgPort *)
```

## FUNCTION

Sets the default filesystem task's port (pr\_FileSystemTask) for the current process.

## INPUTS

port - The pr\_MsgPort of the default filesystem for the process

## RESULT

oldport - The previous FileSysTask value

## SEE ALSO

GetFileSysTask(), Open()

## 1.128 dos.library/SetIoErr

## NAME

SetIoErr -- Sets the value returned by IoErr() (V36)

## SYNOPSIS

```
oldcode = SetIoErr(code)
D0                      D1
```

```
LONG SetIoErr(LONG);
```

## FUNCTION

This routine sets up the secondary result (pr\_Result2) return code (returned by the IoErr() function).

## INPUTS

code - Code to be returned by a call to IoErr.

## RESULT

oldcode - The previous error code.

---

SEE ALSO  
IoErr(), Fault(), PrintFault()

## 1.129 dos.library/SetMode

### NAME

SetMode - Set the current behavior of a handler (V36)

### SYNOPSIS

```
success = SetMode(fh, mode)
D0                      D1  D2
```

BOOL SetMode(BPTR, LONG)

### FUNCTION

SetMode() sends an ACTION\_SCREEN\_MODE packet to the handler in question, normally for changing a CON: handler to raw mode or vice-versa. For CON:, use 1 to go to RAW: mode, 0 for CON: mode.

### INPUTS

fh - filehandle  
mode - The new mode you want

### RESULT

success - Boolean

SEE ALSO

## 1.130 dos.library/SetProgramDir

### NAME

SetProgramDir -- Sets the directory returned by GetProgramDir (V36)

### SYNOPSIS

```
oldlock = SetProgramDir(lock)
D0                      D1
```

BPTR SetProgramDir(BPTR)

### FUNCTION

Sets a shared lock on the directory the program was loaded from. This can be used for a program to find data files, etc, that are stored with the program, or to find the program file itself. NULL is a valid input. This can be accessed via GetProgramDir() or by using paths relative to PROGDIR:.

### INPUTS

lock - A lock on the directory the current program was loaded from

### RESULT

oldlock - The previous ProgramDir.

---

SEE ALSO

GetProgramDir(), Open()

## 1.131 dos.library/SetProgramName

NAME

SetProgramName -- Sets the name of the program being run (V36)

SYNOPSIS

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

```
BOOL SetProgramName(char *)
```

FUNCTION

Sets the name for the program in the cli structure. If the name is too long to fit, a failure is returned, and the old value is left intact. It is advised that you inform the user if possible of this condition, and/or set the program name to an empty string. This routine is safe to call even if there is no CLI structure.

INPUTS

name - Name of program to use.

RESULT

success - Success/failure indicator.

BUGS

This clips to a fixed (1.3 compatible) size.

SEE ALSO

GetProgramName()

## 1.132 dos.library/SetPrompt

NAME

SetPrompt -- Sets the CLI/shell prompt for the current process (V36)

SYNOPSIS

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

```
BOOL SetPrompt(char *)
```

FUNCTION

Sets the text for the prompt in the cli structure. If the prompt is too long to fit, a failure is returned, and the old value is left intact. It is advised that you inform the user of this condition. This routine is safe to call even if there is no CLI structure.

INPUTS

name - Name of prompt to be set.

---

## RESULT

success - Success/failure indicator.

## BUGS

This clips to a fixed (1.3 compatible) size.

## SEE ALSO

GetPrompt()

## 1.133 dos.library/SetProtection

## NAME

SetProtection -- Set protection for a file or directory

## SYNOPSIS

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

```
BOOL SetProtection (char *, LONG)
```

## FUNCTION

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

|                                   |                           |
|-----------------------------------|---------------------------|
| 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.     |

Before V36, the ROM filesystem didn't respect the Read and Write bits. In V36 or later and in the FFS, the Read and Write bits are respected.

The archive bit should be cleared by the filesystem whenever the file is changed. Backup utilities will generally set the bit after backing up each file.

The V36 Shell looks at the execute bit, and will refuse to execute a file if it is set.

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

## INPUTS

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

## RESULTS

success - boolean

## SEE ALSO

SetComment(), Examine(), ExNext(), <dos/dos.h>

## 1.134 dos.library/SetVar

### NAME

SetVar -- Sets a local or environment variable (V36)

### SYNOPSIS

```
success = SetVar( name, buffer, size, flags )
```

```
D0                D1      D2      D3      D4
```

```
BOOL SetVar(UBYTE *, UBYTE *, LONG, ULONG )
```

### FUNCTION

Sets a local or environment variable. It is advised to only use ASCII strings inside variables, but not required.

### INPUTS

name - pointer to an variable name. Note variable names follow filesystem syntax and semantics.

buffer - a user allocated area which contains a string that is the value to be associated with this variable.

size - length of the buffer region in bytes. -1 means buffer contains a null-terminated string.

flags - combination of type of var to set (low 8 bits), and flags to control the behavior of this routine. Currently defined flags include:

GVF\_LOCAL\_ONLY - set a local (to your process) variable.

GVF\_GLOBAL\_ONLY - set a global environment variable.

The default is to set a local environment variable.

### RESULT

success - If non-zero, the variable was sucessfully set, FALSE indicates failure.

### BUGS

LV\_VAR is the only type that can be global

### SEE ALSO

GetVar(), DeleteVar(), FindVar(), <dos/var.h>

## 1.135 dos.library/SetVBuf

### NAME

SetVBuf -- set buffering modes and size (V36)

### SYNOPSIS

```
error = SetVBuf(fh, buff, type, size)
```

```
D0                D1      D2      D3      D4
```



```
LONG SetVBuf(BPTR, UBYTE *, LONG, LONG)
```

#### FUNCTION

Changes the buffering modes and buffer size for a filehandle. With `buff == NULL`, the current buffer will be deallocated and a new one of (approximately) `size` will be allocated. If buffer is non-NULL, it will be used for buffering and must be at least `max(size, 208)` bytes long. If `buff` is NULL and `size` is -1, then only the buffering mode will be changed.

#### INPUTS

`fh` - Filehandle  
`buff` - buffer pointer for buffered I/O  
`type` - buffering mode (see `<dos/stdio.h>`)  
`size` - size of buffer for buffered I/O (sizes less than 208 bytes will be ignored).

#### RESULT

`error` - 0 if successful. NOTE: opposite of most dos functions!

#### BUGS

Not implemented yet, always returns 0.

#### SEE ALSO

`Fputc()`, `Fgetc()`, `Ungetc()`, `Flush()`, `Fread()`, `Fwrite()`, `Fgets()`, `Fputs()`.

## 1.136 dos.library/SplitName

#### NAME

`SplitName` -- splits out a component of a pathname into a buffer (V36)

#### SYNOPSIS

```
newpos = SplitName(name, separator, buf, oldpos, size)
D0          D1      D2      D3      D4      D5
```

```
WORD SplitName(UBYTE *, UBYTE, UBYTE *, WORD, LONG)
```

#### FUNCTION

This routine splits out the next piece of a name from a given file name. Each piece is copied into the buffer, truncating at `size-1` characters. The new position is then returned so that it may be passed in to the next call to `splitname`. If the separator is not found within '`size`' characters, then `size-1` characters plus a null will be put into the buffer, and the position of the next separator will be returned.

If a separator cannot be found, -1 is returned (but the characters from the old position to the end of the string are copied into the buffer, up to a maximum of `size-1` characters). Both strings are null-terminated.

This function is mainly intended to support handlers.

#### INPUTS

---

```

name      - Filename being parsed.
separator - Separator character to split by.
buf       - Buffer to hold separated name.
oldpos    - Current position in the file.
size      - Size of buf in bytes (including null termination);

```

#### RESULT

```
newpos    - New position for next call to splitname. -1 for last one.
```

#### SEE ALSO

```
FilePart(), PathPart(), AddPart()
```

## 1.137 dos.library/StartNotify

#### NAME

```
StartNotify -- Starts notification on a file or directory (V36)
```

#### SYNOPSIS

```

success = StartNotify(notifystructure)
D0                      D1

```

```
BOOL StartNotify(struct NotifyRequest *)
```

#### FUNCTION

Posts a notification request. Do not modify the notify structure while it is active. You will be notified when the file or directory changes. For files, you will be notified after the file is closed. Not all filesystems will support this: applications should NOT require it. In particular, most network filesystems won't support it.

#### INPUTS

```
notifystructure - A filled-in NotifyRequest structure
```

#### RESULT

```
success - Success/failure of request
```

#### BUGS

The V36 floppy/HD filesystem doesn't actually send notifications. The V36 ram handler (ram:) does. This has been fixed for V37.

#### SEE ALSO

```
EndNotify(), <dos/notify.h>
```

## 1.138 dos.library/StrToDate

#### NAME

```
StrToDate -- Converts a string to a DateStamp (V36)
```

#### SYNOPSIS

```

success = StrToDate( datetime )
D0                      D1

```

```
BOOL StrToDate( struct DateTime * )
```

#### FUNCTION

Converts a human readable ASCII string into an AmigaDOS DateStamp.

#### INPUTS

DateTime - a pointer to an initialized DateTime structure.

The DateTime structure should be initialized as follows:

dat\_Stamp - ignored on input.

dat\_Format - a format byte which specifies the format of the dat\_StrDate. This can be any of the following (note: If value used is something other than those below, the default of FORMAT\_DOS is used):

FORMAT\_DOS: AmigaDOS format (dd-mmm-yy).

FORMAT\_INT: International format (yy-mmm-dd).

FORMAT\_USA: American format (mm-dd-yy).

FORMAT\_CDN: Canadian format (dd-mmm-yy).

FORMAT\_DEF: default format for locale.

dat\_Flags - a flags byte. The only flag which affects this function is:

DTF\_SUBST: ignored by this function

DTF\_FUTURE: If set, indicates that strings such as (stored in dat\_StrDate) "Monday" refer to "next" monday. Otherwise, if clear, strings like "Monday" refer to "last" monday.

dat\_StrDay - ignored by this function.

dat\_StrDate - pointer to valid string representing the date. This can be a "DTF\_SUBST" style string such as "Today" "Tomorrow" "Monday", or it may be a string as specified by the dat\_Format byte. This will be converted to the ds\_Days portion of the DateStamp. If this pointer is NULL, DateStamp->ds\_Days will not be affected.

dat\_StrTime - Pointer to a buffer which contains the time in the ASCII format hh:mm:ss. This will be converted to the ds\_Minutes and ds\_Ticks portions of the DateStamp. If this pointer is NULL, ds\_Minutes and ds\_Ticks will be unchanged.

#### RESULT

success - a zero return indicates that a conversion could not be performed. A non-zero return indicates that the

DateTime.dat\_Stamp variable contains the converted values.

SEE ALSO

DateStamp(), DateToStr(), <dos/datetime.h>

## 1.139 dos.library/StrToLong

NAME

StrToLong -- string to long value (decimal) (V36)

SYNOPSIS

characters = StrToLong(string,value)

D0                                    D1        D2

LONG StrToLong(UBYTE \*, LONG \*)

FUNCTION

Converts decimal string into LONG value. Returns number of characters converted. Skips over leading spaces & tabs (included in count). If no decimal digits are found (after skipping leading spaces & tabs), StrToLong returns -1 for characters converted, and puts 0 into value.

INPUTS

string - Input string.

value - Pointer to long value. Set to 0 if no digits are converted.

RESULT

result - Number of characters converted or -1.

## 1.140 dos.library/SystemTagList

NAME

SystemTagList -- Have a shell execute a command line (V36)

SYNOPSIS

error = SystemTagList(command, tags)

D0                                    D1        D2

LONG SystemTagList(UBYTE \*, struct TagItem \*)

error = System(command, tags)

D0                                    D1        D2

LONG System(UBYTE \*, struct TagItem \*)

error = SystemTags(command, Tag1, ...)

LONG SystemTags(UBYTE \*, ULONG, ...)

FUNCTION

Similar to Execute(), but does not read commands from the input

---

filehandle. Spawns a Shell process to execute the command, and returns the returncode the command produced, or -1 if the command could not be run for any reason. The input and output filehandles will not be closed by System, you must close them (if needed) after System returns, if you specified them via SYS\_INPUT or SYS\_OUTPUT.

By default the new process will use your current Input() and Output() filehandles. Normal Shell command-line parsing will be done including redirection on 'command'. The current directory and path will be inherited from your process. Your path will be used to find the command (if no path is specified).

If used with the SYS\_Asynch flag, it WILL close both it's input and output filehandles after running the command (even if these were your Input() and Output()!)

Normally uses the boot (ROM) shell, but other shells can be specified via SYS\_UserShell and SYS\_CustomShell. Normally, you should send things written by the user to the UserShell. The UserShell defaults to the same shell as the boot shell.

The tags are passed through to CreateNewProc() (tags that conflict with SystemTagList() will be filtered out). This allows setting things like priority, etc for the new process.

#### INPUTS

command - Program and arguments  
tags - see <dos/dostags.h>. Note that both SystemTagList()-specific tags and tags from CreateNewProc() may be passed.

#### RESULT

error - 0 for success, result from command, or -1. Note that on error, the caller is responsible for any filehandles or other things passed in via tags.

#### SEE ALSO

Execute(), CreateNewProc(), <dos/dostags.h>, Input(), Output()

## 1.141 dos.library/UnGetC

#### NAME

UnGetC -- Makes a char available for reading again. (buffered) (V36)

#### SYNOPSIS

value = UnGetC(fh, character)  
D0                    D1            D2

LONG UnGetC(BPTR, LONG)

#### FUNCTION

Pushes the character specified back into the input buffer. Every time you use a buffered read routine, you can always push back 1 character. You may be able to push back more, though it is not recommended, since there is no guarantee on how many can be pushed back at a given moment.

---

Passing -1 for the character will cause the last character read to be pushed back. If the last character read was an EOF, the next character read will be an EOF.

Note: UnGetC can be used to make sure that a filehandle is set up as a read filehandle. This is only of importance if you are writing a shell, and must manipulate the filehandle's buffer.

#### INPUTS

fh - filehandle to use for buffered I/O  
character - character to push back or -1

#### RESULT

value - character pushed back, or FALSE if the character cannot be pushed back.

#### BUGS

In V36, UnGetC(fh,-1) after an EOF would not cause the next character read to be an EOF. This was fixed for V37.

#### SEE ALSO

FGetC(), FPutC(), Flush()

## 1.142 dos.library/UnLoadSeg

#### NAME

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

#### SYNOPSIS

```
success = UnLoadSeg( seglist )  
D0                                     D1
```

```
BOOL UnLoadSeg(BPTR)
```

#### FUNCTION

Unload a seglist loaded by LoadSeg(). 'seglist' may be zero. Overlaid segments will have all needed cleanup done, including closing files.

#### INPUTS

seglist - BCPL pointer to a segment identifier

#### RESULTS

success - returns 0 if a NULL seglist was passed or if it failed to close an overlay file. NOTE: this function returned a random value before V36!

#### SEE ALSO

LoadSeg(), InternalLoadSeg(), InternalUnLoadSeg()

## 1.143 dos.library/UnLock

---

## NAME

UnLock -- Unlock a directory or file

## SYNOPSIS

```
UnLock( lock )
        D1
```

```
void UnLock(BPTR)
```

## 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

## SEE ALSO

Lock(), DupLock(), ParentOfFH(), DupLockFromFH()

## 1.144 dos.library/UnLockDosList

## NAME

UnLockDosList -- Unlocks the Dos List (V36)

## SYNOPSIS

```
UnLockDosList(flags)
        D1
```

```
void UnLockDosList(ULONG)
```

## FUNCTION

Unlocks the access on the Dos Device List. You MUST pass the same flags you used to lock the list.

## INPUTS

flags - MUST be the same flags passed to (Attempt)LockDosList()

## SEE ALSO

AttemptLockDosList(), LockDosList(), Permit()

## 1.145 dos.library/UnLockRecord

## NAME

UnLockRecord -- Unlock a record (V36)

## SYNOPSIS

```
success = UnLockRecord(fh,offset,length)
D0                D1    D2    D3
```

BOOL UnLockRecord (BPTR, ULONG, ULONG)

## FUNCTION

This releases the specified lock on a file. Note that you must use the same filehandle you used to lock the record, and offset and length must be the same values used to lock it. Every LockRecord() call must be balanced with an UnlockRecord() call.

## INPUTS

```
fh      - File handle of locked file
offset  - Record start position
length  - Length of record in bytes
```

## RESULT

success - Success or failure.

## BUGS

See `LockRecord()`

SEE ALSO

LockRecords(), LockRecord(), UnLockRecords()

### 1.146 dos.library/UnLockRecords

## NAME

UnLockRecords -- Unlock a list of records (V36)

## SYNOPSIS

```
success = UnLockRecords(record_array)
```

```
BOOL UnLockRecords(struct RecordLock *)
```

## FUNCTION

This releases an array of record locks obtained using LockRecords. You should NOT modify the record\_array while you have the records locked. Every LockRecords() call must be balanced with an UnlockRecords() call.

## INPUTS

record\_array - List of records to be unlocked

## RESULT

success - Success or failure.

## BUGS

See `LockRecord()`

SEE ALSO

```
LockRecords(), LockRecord(), UnlockRecord()
```



## 1.147 dos.library/VFPrintf

### NAME

VFPrintf -- format and print a string to a file (buffered) (V36)

### SYNOPSIS

```
count = VFPrintf(fh, fmt, argv)
D0          D1 D2 D3

LONG VFPrintf(BPTR, char *, LONG *)

count = FPrintf(fh, fmt, ...)

LONG FPrintf(BPTR, char *, ...)
```

### FUNCTION

Writes the formatted string and values to the given file. This routine is assumed to handle all internal buffering so that the formatting string and resultant formatted values can be arbitrarily long. Any secondary error code is returned in IoErr(). This routine is buffered.

### INPUTS

```
fh      - Filehandle to write to
fmt     - RawDoFmt() style formatting string
argv    - Pointer to array of formatting values
```

### RESULT

count - Number of bytes written or -1 (EOF) for an error

### BUGS

The prototype for FPrintf() currently forces you to cast the first varargs parameter to LONG due to a deficiency in the program that generates fds, prototypes, and amiga.lib stubs.

### SEE ALSO

VPrintf(), VFWritef(), RawDoFmt(), FPutC()

## 1.148 dos.library/VFWritef

### NAME

VFWritef - write a BCPL formatted string to a file (buffered) (V36)

### SYNOPSIS

```
count = VFWritef(fh, fmt, argv)
D0          D1 D2 D3

LONG VFWritef(BPTR, char *, LONG *)

count = FWritef(fh, fmt, ...)

LONG FWritef(BPTR, char *, ...)
```

### FUNCTION

Writes the formatted string and values to the default output. This routine is assumed to handle all internal buffering so that the formatting string and resultant formatted values can be arbitrarily long. The formats are in BCPL form. This routine is buffered.

Supported formats are: (Note x is in base 36!)

- %S - string (CSTR)
- %Tx - writes a left-justified string in a field at least x bytes long.
- %C - writes a single character
- %Ox - writes a number in octal, maximum x characters wide
- %Xx - writes a number in hex, maximum x characters wide
- %Ix - writes a number in decimal, maximum x characters wide
- %N - writes a number in decimal, any length
- %Ux - writes an unsigned number, maximum x characters wide
- \$\$ - ignore parameter

Note: 'x' above is actually the character value - '0'.

#### INPUTS

- fmt - BCPL style formatting string
- argv - Pointer to array of formatting values

#### RESULT

- count - Number of bytes written or -1 for error

#### BUGS

As of V37, VFWritef() does NOT return a valid return value. In order to reduce possible errors, the prototypes supplied for the system as of V37 have it typed as VOID.

#### SEE ALSO

VFPrintf(), VFPrintf(), FPutC()

## 1.149 dos.library/VPrintf

#### NAME

VPrintf -- format and print string (buffered) (V36)

#### SYNOPSIS

```
count = VPrintf(fmt, argv)
      D0          D1    D2
```

```
LONG VPrintf(char *, LONG *)
```

```
count = Printf(fmt, ...)
```

```
LONG Printf(char *, ...)
```

#### FUNCTION

Writes the formatted string and values to Output(). This routine is assumed to handle all internal buffering so that the formatting string and resultant formatted values can be arbitrarily long. Any secondary error code is returned in IoErr(). This routine is buffered.

Note: RawDoFmt assumes 16 bit ints, so you will usually need 'l's in your formats (ex: %ld versus %d).

#### INPUTS

fmt - exec.library RawDoFmt() style formatting string  
argv - Pointer to array of formatting values

#### RESULT

count - Number of bytes written or -1 (EOF) for an error

#### BUGS

The prototype for Printf() currently forces you to cast the first varargs parameter to LONG due to a deficiency in the program that generates fds, prototypes, and amiga.lib stubs.

#### SEE ALSO

VFPrintf(), VFWritef(), RawDoFmt(), FPutC()

## 1.150 dos.library/WaitForChar

#### NAME

WaitForChar -- Determine if chars arrive within a time limit

#### SYNOPSIS

status = WaitForChar( file, timeout )  
D0                           D1    D2

BOOL WaitForChar(BPTR, LONG)

#### 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

#### RESULTS

status - boolean

#### SEE ALSO

Read(), FGetC()

---

## 1.151 dos.library/WaitPkt

### NAME

WaitPkt -- Waits for a packet to arrive at your pr\_MsgPort (V36)

### SYNOPSIS

```
packet = WaitPkt()
D0

struct DosPacket *WaitPkt(void);
```

### FUNCTION

Waits for a packet to arrive at your pr\_MsgPort. If anyone has installed a packet wait function in pr\_PktWait, it will be called. The message will be automatically GetMsg()ed so that it is no longer on the port. It assumes the message is a dos packet. It is NOT guaranteed to clear the signal for the port.

### RESULT

packet - the packet that arrived at the port (from ln\_Name of message).

### SEE ALSO

SendPkt(), DoPkt(), AbortPkt()

## 1.152 dos.library/Write

### NAME

Write -- Write bytes of data to a file

### SYNOPSIS

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

LONG Write (BPTR, void *, LONG)
```

### 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.

Note: this is an unbuffered routine (the request is passed directly to the filesystem.) Buffered I/O is more efficient for small reads and writes; see FPutC().

### INPUTS

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

### RESULTS

returnedLength - integer

---

SEE ALSO

`Read()`, `Seek()`, `Open()`, `Close()`, `FPutC`

## 1.153 dos.library/WriteChars

NAME

`WriteChars` -- Writes bytes to the the default output (buffered) (V36)

SYNOPSIS

`count = WriteChars(buf, buflen)`

D0                                   D1

LONG `WriteChars(UBYTE *, LONG)`

FUNCTION

This routine writes a number of bytes to the default output. The length is returned. This routine is buffered.

INPUTS

`buf` - buffer of characters to write

`buflen` - number of characters to write

RESULT

`count` - Number of bytes written. -1 (EOF) indicates an error

SEE ALSO

`Fputs()`, `FPutC()`, `FWrite()`, `PutStr()`

---