ADosUnsafe

COLLABORATORS			
	TITLE : ADosUnsafe		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		January 13, 2023	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

ADosUnsafe

1.1 AmigaTalk to AmigaDOS Help:

```
                It's advisable that the User of these AmigaTalk Methods
check their work before usage.

This classification is based on my judgement only, but here
is how I arrived at this:  The functions determined to be
Unsafe can change file Locks, traverse a directory,
or change something (such as an Assignment) that might not
be easily found or corrected.

Where it made sense to do so, the arguments the User supplies these
functions/Methods are also checked for valid ranges or
values, so even if you pass in a NULL pointer, AmigaTalk
should short-circuit your attempt to kill your system (I hope!).


UNSAFE AmigaDOS Functions/AmigaTalk Methods:


            writeChars

            vFWritef

            unLockRecords

            unLockRecord

            unLockDosList

            unLock

            startNotify

            setVar

            setProgramName

            setProgramDir
```

setOwner

setMode

setCurrentDirName

rename

relabel

parsePatternNoCase

parsePattern

output

openFromLock

openFile

nextDosEntry

nameFromLock

nameFromFH

matchPatternNoCase

matchPattern

makeLink

makeDosEntry

lockRecords

lockRecord

lockDosList

lock

input

infoDisk

fRead

flushFH

findSegment

findDosEntry

findArg

exNext

```
execute

examineFH

examine

exAllEnd

exAll

dupLockFromFH

dupLock

dateStamp

createDir

closeFile

checkSignal

changeMode

assignPath

assignLock

assignLate

assignAdd

addPart
```

1.2 writeChars (UNSAFE):

```
NAME
    WriteChars -- Writes bytes to the the default output (buffered)

SYNOPSIS
    LONG count = WriteChars( char *buf, LONG buflen );


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

AMIGATALK INTERFACE (UnSafeDOS Class):

  writeChars: aBuffer ofSize: length
     WARNING:  Make sure that aBuffer is a String of length bytes!
```

1.3 vFWritef (UNSAFE):

```
NAME
    VFWritef - write a BCPL formatted string to a file (buffered)

SYNOPSIS
    LONG count = VFWritef( BPTR fh, char *fmt, LONG *argv );


FUNCTION
    Writes the formatted string and values to the specified file.  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
    fh    - filehandle to write to
    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 ,  FPutC
```

```
AMIGATALK INTERFACE (UnSafeDOS Class):

vFWritef: bptrFileHandle format: formatString args: argv
```

1.4 unLockRecords (UNSAFE):

```
                NAME
    UnLockRecords -- Unlock a list of records

SYNOPSIS
    BOOL success = UnLockRecords( struct RecordLock *record_array );

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

BUGS
    See LockRecord

SEE ALSO

              LockRecords
             ,
              LockRecord
             ,

              UnLockRecord

AMIGATALK INTERFACE (UnSafeDOS Class):

unLockRecords: recordLockObject
```

1.5 unLockRecord (UNSAFE):

```
                NAME
    UnLockRecord -- Unlock a record

SYNOPSIS
    BOOL success = UnLockRecord( BPTR fh, ULONG offset, ULONG length )

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

BUGS
    See LockRecord

SEE ALSO

              LockRecords
             ,
              LockRecord
             ,

              UnLockRecords

AMIGATALK INTERFACE (UnSafeDOS Class):

unLockRecord: bptrFileHandle at: offset ofSize: length
```

1.6 unLockDosList (UNSAFE):

```
                NAME
    UnLockDosList -- Unlocks the Dos List

SYNOPSIS
    void UnLockDosList( ULONG flags );

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()

AMIGATALK INTERFACE (UnSafeDOS Class):

unLockDosList: flags
```

1.7 unLock (UNSAFE):

```
                NAME
    UnLock -- Unlock a directory or file

SYNOPSIS
    void UnLock( BPTR lock )

FUNCTION
    The filing system lock (obtained from
              Lock
             ,
              DupLock
             , or

              CreateDir
             ) is removed and deallocated.

INPUTS
    lock - BCPL pointer to a lock

NOTES
    passing zero to UnLock() is harmless

SEE ALSO

              Lock
             ,
              DupLock
             ,
     ParentOfFH ,
              DupLockFromFH

AMIGATALK INTERFACE (UnSafeDOS Class):

unLock: bptrLock
```

1.8 startNotify (UNSAFE):

```
NAME
    StartNotify -- Starts notification on a file or directory

SYNOPSIS
    BOOL success = StartNotify( struct NotifyRequest *nr );

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

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>

AMIGATALK INTERFACE (UnSafeDOS Class):

startNotify: notifyRequest
```

1.9 setVar (UNSAFE):

```
NAME
    SetVar -- Sets a local or environment variable

SYNOPSIS
    BOOL success = SetVar( char *name, char *buffer,
                           LONG size, ULONG flags );

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>

AMIGATALK INTERFACE (UnSafeDOS Class):
```

```
setVar: varName from: aBuffer ofSize: size flags: flags
```

1.10 setProgramName (UNSAFE):

```
NAME
    SetProgramName -- Sets the name of the program being run

SYNOPSIS
    BOOL success = SetProgramName( char *name )

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.

BUGS
    This clips to a fixed (1.3 compatible) size.

SEE ALSO
     GetProgramName

AMIGATALK INTERFACE (UnSafeDOS Class):

setProgramName: newProgramName
```

1.11 setProgramDir (UNSAFE):

```
                NAME
    SetProgramDir -- Sets the directory returned by GetProgramDir

SYNOPSIS
    BPTR oldlock = SetProgramDir( BPTR lock );

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

AMIGATALK INTERFACE (UnSafeDOS Class):

setProgramDirTo: bptrLock
```

1.12 setOwner (UNSAFE):

```
                NAME
    SetOwner -- Set owner information for a file or directory (V39)

SYNOPSIS
    BOOL success = SetOwner( char *name, LONG owner_info );

FUNCTION
    SetOwner() sets the owner information for the file or directory.
    This value is a 32-bit value that is normally split into 16 bits
of owner user id (bits 31-16), and 16 bits of owner group id (bits
15-0).  However, other than returning them as shown by
             Examine
            /

             ExNext
            /
             ExAll
            , the filesystem take no interest in the values.
These are primarily for use by networking software (clients and
hosts), in conjunction with the FIBF_OTR_xxx and FIBF_GRP_xxx
protection bits.

This entrypoint did not exist in V36, so you must open at least V37
dos.library to use it.  V37 dos.library will return FALSE to this
call.

INPUTS
    name      - pointer to a null-terminated string
    owner_info - owner uid (31:16) and group id (15:0)

SEE ALSO
     SetProtect ,
              Examine
             ,

              ExNext
             ,
              ExAll
             , <dos/dos.h>

AMIGATALK INTERFACE (UnSafeDOS Class):

setOwnderUID: name to: ownerUID
```

1.13 setMode (UNSAFE):

```
NAME
    SetMode - Set the current behavior of a handler

SYNOPSIS
    BOOL success = SetMode( BPTR fh, LONG mode );

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

AMIGATALK INTERFACE (UnSafeDOS Class):

setFileMode: bptrFileHandle to: mode
```

1.14 setCurrentDirName (UNSAFE):

```
NAME
    SetCurrentDirName -- Sets the directory name for the process

SYNOPSIS
    BOOL success = SetCurrentDirName( char *name );

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.

BUGS
    This clips to a fixed (1.3 compatible) size.

SEE ALSO
     GetCurrentDirName

AMIGATALK INTERFACE (UnSafeDOS Class):

setCurrentDirNameTo: dirName
```

1.15 rename (UNSAFE):

```
                NAME
    Rename -- Rename a directory or file
```

```
SYNOPSIS
    BOOL success = Rename( char *oldName, char *newName );

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

Note:  It is impossible to Rename a file from one volume to
another.

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

SEE ALSO

              Relabel

AMIGATALK INTERFACE (UnSafeDOS Class):

rename: oldFileOrDirName to: newName
```

1.16 relabel (UNSAFE):

```
NAME
    Relabel -- Change the volume name of a volume

SYNOPSIS
    BOOL success = Relabel( char *volumename, char *name )

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 :)

AMIGATALK INTERFACE (UnSafeDOS Class):

relabel: volumenName to: newName
```

1.17 parsePatternNoCase (UNSAFE):

```
                NAME
    ParsePatternNoCase -- Create a tokenized string for
                          MatchPatternNoCase
```

```
SYNOPSIS
    LONG IsWild = ParsePatternNoCase( char *Source, char *Dest, LONG DestLength ↩
        );

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
    In V37 this call didn't always set  IoErr  to something
    useful on an error.  Fixed in V39.
In V37, it didn't properly convert character-classes ([x-y]) to
upper case.  Workaround: convert the input pattern to upper case
using ToUpper() from utility.library before calling
ParsePatternNoCase().  Fixed in V39 dos.

SEE ALSO

             ParsePattern
            ,
             MatchPatternNoCase
            ,
     MatchFirst ,  MatchNext ,
    utility.library/ToUpper

AMIGATALK INTERFACE (UnSafeDOS Class):

parsePatternNoCase: source into: dest ofSize: destLength  " Tested "
```

1.18 parsePattern (UNSAFE):

```
               NAME
    ParsePattern -- Create a tokenized string for MatchPattern

SYNOPSIS
    LONG IsWild = ParsePattern( char *Source, char *Dest, LONG DestLength );
```

```
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
    In V37 this call didn't always set  IoErr  to something
    useful on an error.  Fixed in V39.

SEE ALSO

              ParsePatternNoCase
             ,
```

```
               MatchPattern
              ,
     MatchFirst ,  MatchNext

AMIGATALK INTERFACE (UnSafeDOS Class):

parsePattern: source into: dest ofSize: destLength
```

1.19 output (UNSAFE):

```
                NAME
    Output -- Identify the programs' initial output file handle

SYNOPSIS
    BPTR file = 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().

RESULT
    file - BCPL pointer to a file handle

SEE ALSO

               Input

AMIGATALK INTERFACE (UnSafeDOS Class):

getOutputHandle
```

1.20 openFromLock (UNSAFE):

```
                NAME
    OpenFromLock -- Opens a file you have a lock on

SYNOPSIS
    BPTR fh = OpenFromLock( BPTR lock )

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

               UnLocked
               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

AMIGATALK INTERFACE (UnSafeDOS Class):

openFileFromLock: bptrLock
```

1.21 openFile (UNSAFE):

```
                NAME
    Open -- Open a file for input or output

SYNOPSIS
    BPTR file = Open( char *name, LONG accessMode );

FUNCTION
    The named file is opened and a file handle returned.  If the
    accessMode is MODE_OLDFILE, an existing file is opened for reading
or writing. If the value is MODE_NEWFILE, a new file is created for
writing. MODE_READWRITE opens 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

RESULT
    file - BCPL pointer to a file handle

SEE ALSO

              Close
             ,
              ChangeMode
             ,

              NameFromFH
             ,  ParentOfFH ,

              ExamineFH

AMIGATALK INTERFACE (UnSafeDOS Class):

open: fileName mode: accessMode
```

1.22 nextDosEntry (UNSAFE):

```
                NAME
    NextDosEntry -- Get the next Dos List entry

SYNOPSIS
    struct DosList *newdlist = NextDosEntry( struct DosList *dlist,
                                               ULONG           flags
                                            );

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
```

```
AMIGATALK INTERFACE (UnSafeDOS Class):

getNextDosEntry: dosList flags: flags
```

1.23 nameFromLock (UNSAFE):

```
                NAME
    NameFromLock -- Returns the name of a locked object

SYNOPSIS
    BOOL success = NameFromLock( BPTR lock, char *buffer, LONG length );

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.

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

SEE ALSO

              NameFromFH
             ,
              Lock

AMIGATALK INTERFACE (UnSafeDOS Class):

getNameFromLock: bptrLock into: aBuffer ofSize: length
```

1.24 nameFromFH (UNSAFE):

```
                NAME
    NameFromFH -- Get the name of an open filehandle

SYNOPSIS
    BOOL success = NameFromFH( BPTR fh, char *buffer, LONG length );

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.

Note:  Older filesystems that don't support
              ExamineFH
              will cause
NameFromFH() to fail with ERROR_ACTION_NOT_SUPPORTED.

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

SEE ALSO

              NameFromLock
             ,
              ExamineFH

AMIGATALK INTERFACE (UnSafeDOS Class):

getNameFromFH: bptrFileHandle into: aBuffer ofSize: length
```

1.25 matchPatternNoCase (UNSAFE):

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

SYNOPSIS
    BOOL match = MatchPatternNoCase( char *pat, char *str );

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 nested levels 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).
```

```
BUGS
    See ParsePatternNoCase.

SEE ALSO

              ParsePatternNoCase
             ,
              MatchPattern
             ,
     MatchFirst ,  MatchNext

AMIGATALK INTERFACE (UnSafeDOS Class):

matchPatternNoCase: pattern in: string    " Tested "
```

1.26 matchPattern (UNSAFE):

```
                NAME
    MatchPattern --  Checks for a pattern match with a string

SYNOPSIS
    BOOL match = MatchPattern( char *pat, char *str );

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 nested levels 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

AMIGATALK INTERFACE (UnSafeDOS Class):
```

```
matchPattern: pattern in: string
```

1.27 makeLink (UNSAFE):

```
                NAME
    MakeLink -- Creates a filesystem link

SYNOPSIS
    BOOL success = MakeLink( char *name, LONG dest, LONG soft );

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

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

SEE ALSO
     ReadLink ,
              Open
             ,

              Lock

AMIGATALK INTERFACE (UnSafeDOS Class):

makeLink: linkName to: destPathBPTRLock flag: softFlag
```

1.28 makeDosEntry (UNSAFE):

```
                NAME
    MakeDosEntry -- Creates a DosList structure
```

```
SYNOPSIS
    struct DosList *newdlist = MakeDosEntry( char *name, LONG type );

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

AMIGATALK INTERFACE (UnSafeDOS Class):

makeDosEntry: name ofType: type
```

1.29 lockRecords (UNSAFE):

```
                NAME
    LockRecords -- Lock a series of records

SYNOPSIS
    BOOL success = LockRecords( struct RecordLock *record_array,
                                ULONG timeout );

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

AMIGATALK INTERFACE (UnSafeDOS Class):

lockRecords: recordLock expiring: timeout
```

1.30 lockRecord (UNSAFE):

```
                NAME
    LockRecord -- Locks a portion of a file

SYNOPSIS
    BOOL success = LockRecord( BPTR fh, ULONG offset, ULONG length,
                               ULONG mode, ULONG timeout
                             );

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.

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

AMIGATALK INTERFACE (UnSafeDOS Class):

lockRecord: bptrFileHandle at: offset ofSize: recordLen
      mode: lockType   expire: timeout
```

1.31 lockDosList (UNSAFE):

```
                NAME
    LockDosList -- Locks the specified Dos Lists for use

SYNOPSIS
    struct DosList *dlist = LockDosList( ULONG flags );

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 on 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 ) != NULL)
      {
      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 compatability 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( LDF_VOLUMES | LDF_READ );

   // 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 foo1/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( LDF_VOLUMES | LDF_READ );
   else
      Permit();

   // must not use dl after this!
   ...

   struct DosList *yourfindentry( struct DosList *dl, STRPTRname, 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
             ,

              NextDosEntry
             , Forbid()

AMIGATALK INTERFACE (UnSafeDOS Class):

lockDosList: flags
```

1.32 lock (UNSAFE):

```
                NAME
    Lock -- Lock a directory or file

SYNOPSIS
    BPTR lock  = Lock( char *name, LONG accessMode );

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

If the accessMode is ACCESS_READ, the lock is a shared read lock;
if the accessMode is ACCESS_WRITE then it is an exclusive write
lock.  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
```

```
RESULT
    lock - BCPL pointer to a lock

SEE ALSO

              UnLock
             ,
              DupLock
             ,

              ChangeMode
             ,
              NameFromLock
             ,

              DupLockFromFH

AMIGATALK INTERFACE (UnSafeDOS Class):

lockFile: fileName mode: accessMode  " Tested "
```

1.33 input (UNSAFE):

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

SYNOPSIS
    BPTR file = 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!

RESULT
    file - BCPL pointer to a file handle

SEE ALSO

              Output
             ,  SelectInput

AMIGATALK INTERFACE (UnSafeDOS Class):

getInputHandle
```

1.34 infoDisk (UNSAFE):

```
NAME
    Info -- Returns information about the disk
```

```
SYNOPSIS
    BOOL success = Info( BPTR lock, struct InfoData *parmBlock );

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
    parmBlock - pointer to an InfoData structure (longword aligned)

SPECIAL NOTE:
    Note that InfoData structure must be longword aligned.

AMIGATALK INTERFACE (UnSafeDOS Class):

diskInfo: bptrLock into: infoDataObject
```

1.35 fRead (UNSAFE):

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

SYNOPSIS
    LONG count = FRead( BPTR fh, char *buf, ULONG blocklen,
                        ULONG blocks );

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( 0 )  before calling.

SEE ALSO
     FGetC ,  FWrite ,
     FGets

AMIGATALK INTERFACE (UnSafeDOS Class):
```

```
fileRead: bptrFileHandle into: aBuffer blockSize: blkSize count: blkCount
```

1.36 flushFH (UNSAFE):

```
                NAME
    Flush -- Flushes buffers for a buffered filehandle

SYNOPSIS
    LONG success = Flush( BPTR fh );

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.

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

The V36 and V37 releases didn't properly flush filehandles which
have never had a buffered IO done on them.  This commonly occurs
on redirection of input of a command, or when opening a file for
input and then calling  CreateNewProc  with NP_Arguments, or when
using a new filehandle with  SelectInput  and then calling
 RunCommand .  This is fixed in V39.  A workaround would be to
do  FGetC , then  UnGetC , then Flush.

SEE ALSO
     FPutC , FGetC ,
     unGetC ,  Seek ,

              Close
             ,  CreateNewProc ,
     SelectInput ,  RunCommand

AMIGATALK INTERFACE (UnSafeDOS Class):

flushFileHandle: bptrFileHandle
```

1.37 findSegment (UNSAFE):

```
NAME
    FindSegment - Finds a segment on the resident list

SYNOPSIS
    struct Segment *s = FindSegment( char *name,
                                     struct Segment *start,
                                     LONG system );

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.  (Since I have NOT yet implemented a way to use Forbid(), you'd
better NOT use this AmigaTalk method! JTS)

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()

AMIGATALK INTERFACE (UnSafeDOS Class):

findSegment: segmentName startingAt: startSegment flag: systemOrUser
```

1.38 findDosEntry (UNSAFE):

```
                NAME
    FindDosEntry -- Finds a specific Dos List entry

SYNOPSIS
    struct DosList *newdlist = FindDosEntry( struct DosList *dlist,
                                             char *name, ULONG flags );
```

```
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

AMIGATALK INTERFACE (UnSafeDOS Class):

findDosEntry: devName in: dosList flags: flags
```

1.39 findArg (UNSAFE):

```
NAME
    FindArg - find a keyword in a template

SYNOPSIS
    LONG index = FindArg( char *template, char *keyword );

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

AMIGATALK INTERFACE (UnSafeDOS Class):

findArgumentIndex: keyword using: template
```

1.40 exNext (UNSAFE):

```
                NAME
    ExNext -- Examine the next entry in a directory

SYNOPSIS
    BOOL success = ExNext( BPTR lock, struct FileInfoBlock *fib );

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.

SPECIAL NOTE:
    FileInfoBlock must be longword-aligned.   AllocDosObject  will
    allocate them correctly for you.

SEE ALSO

                Examine
               ,
                Lock
               ,

                UnLock
               ,  IoErr ,

                ExamineFH
               ,  AllocDosObject ,

                ExAll

AMIGATALK INTERFACE (UnSafeDOS Class):

examineNext: bptrLock into: fileInfoBlock   " Tested "
```

1.41 execute (UNSAFE):

```
NAME
    Execute -- Execute a CLI command

SYNOPSIS
    BOOL success = Execute( char *commandString, BPTR input, BPTR output );

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

RESULT
    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

AMIGATALK INTERFACE (UnSafeDOS Class):

execute: commandString with: bptrInput and: bptrOutput
```

1.42 examineFH (UNSAFE):

```
                NAME
    ExamineFH -- Gets information on an open file

SYNOPSIS
    BOOL success = ExamineFH( BPTR fh, struct FileInfoBlock *fib );

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.
```

```
SEE ALSO

              Examine
             ,
              ExNext
             ,

              ExAll
             ,
              Open
             ,
     AllocDosObject

AMIGATALK INTERFACE (UnSafeDOS Class):

examineFileHandle: bptrFileHandle into: fileInfoBlock
```

1.43 examine (UNSAFE):

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

SYNOPSIS
    BOOL success = Examine( BPTR lock, struct FileInfoBlock *fib );

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)

RESULT
    success - boolean

SPECIAL NOTE:
    FileInfoBlock must be longword-aligned.   AllocDosObject  will
    allocate them correctly for you.

SEE ALSO

              Lock
             ,
              UnLock
```

```
              ,

               ExNext
              ,
               ExamineFH
              ,
     AllocDosObject ,
               ExAll
              ,
    <dos/dos.h>,

AMIGATALK INTERFACE (UnSafeDOS Class):

examine: bptrLock into: fileInfoBlock   " Tested "
```

1.44 exAllEnd (UNSAFE):

```
                NAME
    ExAllEnd -- Stop an ExAll (V39)

SYNOPSIS
    ExAllEnd( BPTR lock, char *buffer, LONG size,
              LONG type, struct ExAllControl *control );

FUNCTION
    Stops an ExAll() on a directory before it hits NO_MORE_ENTRIES.
    The full set of arguments that had been passed to ExAll() must be
passed to ExAllEnd(), so it can handle filesystems that can't abort
an
              ExAll
              directly.

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!

SEE ALSO

              ExAll
             ,  AllocDosObject

AMIGATALK INTERFACE (UnSafeDOS Class):

endExamine: exAllControl with: bptrLock from: aBuffer ofSize: size type: t
```

1.45 exAll (UNSAFE):

```
                NAME
    ExAll -- Examine an entire directory

SYNOPSIS
    BOOL continue = ExAll( BPTR lock, char *buffer,
                           LONG size, LONG type,
                           struct ExAllControl *control );

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)

     Note:  The V37 ROM/disk filesystem returns this
            incorrectly as a BSTR.  See BUGS for a workaround.

   ED_OWNER        owner user-id and group-id (if supported)  (V39)

Thus, ED_NAME gives only filenames, and ED_OWNER gives everything.

NOTE:  V37 dos.library, when doing ExAll() emulation, and RAM: filesystem
will return an error if passed ED_OWNER.  If you get ERROR_BAD_NUMBER,
retry with ED_COMMENT to get everything but owner info.  All filesystems
supporting ExAll() must support through ED_COMMENT, and must check Type
and return ERROR_BAD_NUMBER if they don't support the type.

The V37 ROM/disk filesystem doesn't fill in the comment field correctly
if you specify ED_OWNER.  See BUGS for a workaround if you need to use
ED_OWNER.

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 occured, 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, EAData, sizeof( EAData ), 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 *) EAData;

    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 (particularily 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.

The V37 ROM/disk filesystem incorrectly returned comments as BSTR's
(length plus characters) instead of CSTR's (null-terminated).  See
the next bug for a way to determine if the filesystem is a V37
ROM/disk filesystem.  Fixed in V39.

The V37 ROM/disk filesystem incorrectly handled values greater than
```

```
ED_COMMENT.  Because of this, ExAll() information is trashed if
ED_OWNER is passed to it.  Fixed in V39.  To work around this, use
the following code to identify if a filesystem is a V37 ROM/disk
filesystem:

// return TRUE if this is a V37 ROM filesystem, which doesn't (really)
// support ED_OWNER safely

BOOL CheckV37( BPTR lock )
{
   struct FileLock *l = BADDR( lock );
   struct Resident *resident;
   struct DosList  *dl;
   BOOL             result = FALSE;

   dl = LockDosList(LDF_READ|LDF_DEVICES);

   // if the lock has a volume and no device, we won't find it,
   // so we know it's not a V37 ROM/disk filesystem
   do {
      dl = NextDosEntry(dl,LDF_READ|LDF_DEVICES);
      if (dl && (dl->dol_Task == l->fl_Task))
         {
         // found the filesystem - test isn't actually required,
         // but we know the filesystem we're looking for will always
         // have a startup msg.  If we needed to examine the message,
         // we would need a _bunch_ of checks to make sure it's not
         // either a small value (like port-handler uses) or a BSTR.

         if (dl->dol_misc.dol_handler.dol_Startup)
            {
            // try to make sure it's the ROM fs or l:FastFileSystem
            if (resident =
                 FindRomTag( dl->dol_misc.dol_handler.dol_SegList ))
               {
               if (resident->rt_Version < 39
                  && (strncmp( resident->rt_IdString, "fs 37.",
                               strlen( "fs 37." )) == 0
                  || strncmp( resident->rt_Name, "ffs 37.",
                               strlen( "ffs 37." )) == 0))
                  {
                  result = TRUE;
                  }
               }
            }

         break;
         }

      } while (dl);

   UnLockDosList(LDF_READ|LDF_DEVICES);

   return result;
}

SEE ALSO
```

```
               Examine
              ,
               ExNext
              ,

               ExamineFH
              ,
               MatchPatternNoCase
              ,

               ParsePatternNoCase
              ,  AllocDosObject ,

               ExAllEnd

AMIGATALK INTERFACE (UnSafeDOS Class):

examineAll: exAllControl with: bptrLock into: aBuffer ofSize: size type: t
```

1.46 dupLockFromFH (UNSAFE):

```
                NAME
    DupLockFromFH -- Gets a lock on an open file

SYNOPSIS
    BPTR lock = DupLockFromFH( BPTR fh );

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

AMIGATALK INTERFACE (UnSafeDOS Class):

duplicateLockFromFH: bptrFileHandle
```

1.47 dupLock (UNSAFE):

```
                NAME
    DupLock -- Duplicate a lock

SYNOPSIS
    BPTR lock = DupLock( BPTR lock );

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

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

A zero return indicates failure.

INPUTS
    lock - BCPL pointer to a lock

RESULT
    newLock - BCPL pointer to a lock

SEE ALSO

             Lock
            ,
             UnLock
            ,

             DupLockFromFH
            ,  ParentOfFH

AMIGATALK INTERFACE (UnSafeDOS Class):

duplicateLock: bptrLock   " Tested "
```

1.48 dateStamp (UNSAFE):

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

SYNOPSIS
    struct DateStamp *ds = DateStamp( struct DateStamp *ds );

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

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

SEE ALSO
     DateToStr ,  StrToDate ,
     SetFileDate ,  CompareDates

AMIGATALK INTERFACE (UnSafeDOS Class):

makeDateStamp: dateStampObject
```

1.49 CreateDir (UNSAFE):

```
                NAME
    CreateDir -- Create a new directory

SYNOPSIS
    BPTR lock = CreateDir( char *name )

FUNCTION
    CreateDir creates a new directory with the specified name. An error
    is returned if it fails.  Directories can only be created on
    devices which support them, e.g. disks.  CreateDir returns an
    exclusive lock on the new directory if it succeeds.

INPUTS
    name - pointer to a null-terminated string

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

SEE ALSO

              Lock
             ,
              UnLock

AMIGATALK INTERFACE (UnSafeDOS Class):

createDir: dirName
```

1.50 closeFile (UNSAFE):

```
                NAME
    Close -- Close an open file

SYNOPSIS
    BOOL success = Close( BPTR file );

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

INPUTS
    file - BCPL pointer to a file handle

RESULT
    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

AMIGATALK INTERFACE (UnSafeDOS Class):

close: bptrFileHandle
```

1.51 checkSignal (UNSAFE):

```
NAME
    CheckSignal -- Checks for break signals

SYNOPSIS
    ULONG signals = CheckSignal( ULONG mask );

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.
```

```
AMIGATALK INTERFACE (UnSafeDOS Class):

checkForSignal: withBitMask
```

1.52 changeMode (UNSAFE):

```
                NAME
    ChangeMode - Change the current mode of a lock or filehandle

SYNOPSIS
    BOOL success = ChangeMode( ULONG type, BPTR object, ULONG newmode );

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

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

AMIGATALK INTERFACE (UnSafeDOS Class):

changeMode: bptrLockOrFH type: type to: newMode
```

1.53 assignPath (UNSAFE):

```
                NAME
    AssignPath -- Creates an assignment to a specified path

SYNOPSIS
    BOOL success = AssignPath( char *name, char *path );

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:xyzzy'' ... )
             , 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

AMIGATALK INTERFACE (UnSafeDOS Class):

addAssignment: assignName toPath: pathName
```

1.54 assignLock (UNSAFE):

```
               NAME
    AssignLock -- Creates an assignment to a locked object

SYNOPSIS
    BOOL success = AssignLock( char *name, BPTR lock );

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

AMIGATALK INTERFACE (UnSafeDOS Class):

addAssignment: assignName toLock: bptrLock
```

1.55 assignLate (UNSAFE):

```
                NAME
    AssignLate -- Creates an assignment to a specified path later

SYNOPSIS
    BOOL success = AssignLate( char *name, char *path );

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:xyzzy"...), 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
              ,

AMIGATALK INTERFACE (UnSafeDOS Class):

addAssignmentLater: assignName to: pathFileName
```

1.56 assignAdd (UNSAFE):

```
                NAME
    AssignAdd -- Adds a lock to an assign for multi-directory assigns

SYNOPSIS
    BOOL success = AssignAdd( char *name, BPTR lock );

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

AMIGATALK INTERFACE (UnSafeDOS Class):

addAssignment: assignName to: bptrLock
```

1.57 addPart (UNSAFE):

```
NAME
    AddPart -- Appends a file/dir to the end of a path

SYNOPSIS
    BOOL success = AddPart( char *dirname, char *filename, ULONG size )

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 occured, 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

AMIGATALK INTERFACE (UnSafeDOS Class):
```

```
addPart: fileName to: dirName ofSize: size
```