ADosSafe

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

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

1 ADosSafe **1**

1.1 AmigaTalk to AmigaDOS Help: . . . 1
1.2 waitForChar (SAFE): . . . 3
1.3 vPrintf (SAFE): . . . 4
1.4 vFPrintf (SAFE): . . . 5
1.5 unGetC (SAFE): . . . 5
1.6 strToLong (SAFE): . . . 6
1.7 strToDate (SAFE): . . . 7
1.8 splitName (SAFE): . . . 8
1.9 setProtection (SAFE): . . . 9
1.10 setPrompt (SAFE): . . . 10
1.11 setIoErr (SAFE): . . . 11
1.12 setFileDate (SAFE): . . . 11
1.13 setComment (SAFE): . . . 12
1.14 sameLock (SAFE): . . . 12
1.15 sameDevice (SAFE): . . . 13
1.16 readLink (SAFE): . . . 14
1.17 readItem (SAFE): . . . 14
1.18 readArgs (SAFE): . . . 15
1.19 readFile (SAFE): . . . 18
1.20 putStr (SAFE): . . . 18
1.21 printFault (SAFE): . . . 19
1.22 pathPart (SAFE): . . . 20
1.23 parentOfFH (SAFE): . . . 20
1.24 parentDir (SAFE): . . . 21
1.25 maxCli (SAFE): . . . 22
1.26 matchNext (SAFE): . . . 22
1.27 matchFirst (SAFE): . . . 23
1.28 matchEnd (SAFE): . . . 25
1.29 isInteractive (SAFE): . . . 25

1.30 isFileSystem (SAFE): . . . 26
1.31 ioErr (SAFE): . . . 26
1.32 getVar (SAFE): . . . 27
1.33 getPrompt (SAFE): . . . 28
1.34 getProgramName (SAFE): . . . 29
1.35 getProgramDir (SAFE): . . . 29
1.36 getFileSysTask (SAFE): . . . 30
1.37 getDeviceProc (SAFE): . . . 30
1.38 getCurrentDirName (SAFE): . . . 31
1.39 getConsoleTask (SAFE): . . . 32
1.40 getArgStr (SAFE): . . . 32
1.41 fPutS (SAFE): . . . 33
1.42 fPutC (SAFE): . . . 34
1.43 findVar (SAFE): . . . 34
1.44 findCliProc (SAFE): . . . 35
1.45 filePart (SAFE): . . . 35
1.46 fGetS (SAFE): . . . 36
1.47 fGetC (SAFE): . . . 37
1.48 fault (SAFE): . . . 38
1.49 ErrorReport (SAFE): . . . 39
1.50 endNotify (SAFE): . . . 40
1.51 delay (SAFE): . . . 40
1.52 dateToStr (SAFE): . . . 41
1.53 currentDir (SAFE): . . . 42
1.54 compareDates (SAFE): . . . 43
1.55 cliPointer (SAFE): . . . 43
1.56 addBuffers (SAFE): . . . 44
1.57 AbortPacket (SAFE): . . . 44

Chapter 1

ADosSafe

1.1 AmigaTalk to AmigaDOS Help:

The functions listed here & used by AmigaTalk have been deemed the least harmful of the AmigaDOS functions. This is based on my judgement only, but here is how I arrived at this: The functions determined to be safe are mainly for gathering information from AmigaDOS. Those that actually change things are easily corrected by the User (for example: If you use setComment: commentString onFile: fileName & you used the wrong commentString, you can always re-do the Method with the correct comment, or correct it using a file utility, such as DirOpus or DiskMaster II). 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!).

SAFE AmigaDOS Functions/AmigaTalk Methods:

waitForChar

vPrintf

vFPrintf

unGetC

strToLong

strToDate

splitName

setProtection

setPrompt

setIoErr

setFileDate

setComment

sameLock

sameDevice

readLink

readItem

readArgs

readFile

putStr

printFault

pathPart

parentOfFH

parentDir

maxCli

matchNext

matchFirst

matchEnd

isInteractive

isFileSystem

ioErr

getVar

getPrompt

getProgramName

getProgramDir

getFileSysTask

getDeviceProc

getCurrentDirName

getConsoleTask

getArgStr

fPutS

fPutC

findVar

findCliProc

filePart

fGetS

fGetC

fault

endNotify

errReport

delay

dateToStr

currentDir

compareDates

cliPointer

addBuffers

abortPacket

1.2 waitForChar (SAFE):

```
                NAME
    WaitForChar -- Determine if chars arrive within a time limit

SYNOPSIS
    BOOL status = WaitForChar( BPTR file, LONG timeout );


FUNCTION
    If a character is available to be read from 'file' within 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

SEE ALSO

                Read
               ,
                FGetC

AMIGATALK INTERFACE (SafeDOS Class):

waitForCharAt: bptrFileHandle for: timeout
```

1.3 vPrintf (SAFE):

```
                 NAME
    VPrintf -- format and print string (buffered)

SYNOPSIS
    LONG count = VPrintf( char *fmt, LONG *argv );

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 (example:  %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 ,

                FPutC
               , RawDoFmt()
```

```
AMIGATALK INTERFACE (SafeDOS Class):

vPrintf: formatString withArgs: argv
```

1.4 vFPrintf (SAFE):

```
                NAME
    VFPrintf -- format and print a string to a file (buffered)

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

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 ,

              FPutC
             , RawDoFmt()

AMIGATALK INTERFACE (SafeDOS Class):

vFPrintfTo: bptrFileHandle format: fmtString withArgs: argv
```

1.5 unGetC (SAFE):

```
                NAME
    UnGetC -- Makes a char available for reading again. (buffered)

SYNOPSIS
    LONG value = UnGetC( BPTR fh, LONG character )

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

AMIGATALK INTERFACE (SafeDOS Class):

unGetC: chr to: bptrFileHandle
```

1.6 strToLong (SAFE):

```
NAME
    strToLong -- string to long value (decimal)

SYNOPSIS
    LONG value = strToLong: aString
```

```
FUNCTION
    Converts decimal string into LONG value.
    Skips over leading spaces & tabs.  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.

RESULT
    result - the value the string was converted to.

AMIGATALK INTERFACE (SafeDOS Class):

strToLong: aString
```

1.7 strToDate (SAFE):

```
                NAME
    StrToDate -- Converts a string to a DateStamp

SYNOPSIS
    BOOL success = StrToDate( struct DateTime *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_StrDat.  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:   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 bythis 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>

AMIGATALK INTERFACE (SafeDOS Class):

strToDate: dateTimeObject
```

1.8 splitName (SAFE):

```
                NAME
    SplitName -- splits out a component of a pathname into a buffer

SYNOPSIS
    WORD newpos = SplitName( char *name, UBYTE separator,
                             char *buf, WORD oldpos, LONG size );

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 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 charactor 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.

BUGS
    In V36 and V37, path portions greater than or equal to 'size' caused
    the last character of the portion to be lost when followed by a
separator.  Fixed for V39 dos.  For V36 and V37, the suggested work-
around is to call SplitName() with a buffer one larger than normal
(for example, 32 bytes), and then set buf[size - 2] to '0' (for example,
buf[30] = '0';).

SEE ALSO

             FilePart
            ,
             PathPart
            ,
     AddPart

AMIGATALK INTERFACE (SafeDOS Class):

splitName: name by: sep into: aBuffer ofSize: size at: oldpos
```

1.9 setProtection (SAFE):

```
                NAME
    SetProtection -- Set protection for a file or directory

SYNOPSIS
    BOOL success = SetProtection( char *name, LONG mask );

FUNCTION
    SetProtection() sets the protection attributes on a file or
    directory.  See <dos/dos.h> for a listing of protection bits.
```

```
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

SEE ALSO

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

AMIGATALK INTERFACE (SafeDOS Class):

setProtectionOf: filename to: protectionMask   " Tested. "
```

1.10 setPrompt (SAFE):

```
                NAME
    SetPrompt -- Sets the CLI/shell prompt for the current process

SYNOPSIS
    BOOL success = SetPrompt( char *name );

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.

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

SEE ALSO

               GetPrompt
```

```
AMIGATALK INTERFACE (SafeDOS Class):

setPromptTo: newPromptString
```

1.11 setIoErr (SAFE):

```
                NAME
    SetIoErr -- Sets the value returned by
              IoErr
                SYNOPSIS
    LONG oldcode = SetIoErr( LONG code );

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

AMIGATALK INTERFACE (SafeDOS Class):

setIoErrTo: errorCode
```

1.12 setFileDate (SAFE):

```
NAME
    SetFileDate -- Sets the modification date for a file or dir

SYNOPSIS
    BOOL success = SetFileDate( char *name, struct DateStamp *date );


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


SEE ALSO
     DateStamp ,  Examine ,
     ExNext ,  ExAll

AMIGATALK INTERFACE (SafeDOS Class):

setFileDateOf: fileOrDirName to: dateStampObject
```

1.13 setComment (SAFE):

```
                NAME
    SetComment -- Change a files' comment string

SYNOPSIS
    BOOL success = SetComment( char *name, char *comment );


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


SEE ALSO
     Examine ,  ExNext ,

               SetProtection

AMIGATALK INTERFACE (SafeDOS Class):

setCommentFieldOf: fileOrDirName to: comment   " Tested "
```

1.14 sameLock (SAFE):

```
NAME
    SameLock -- returns whether two locks are on the same object

SYNOPSIS
    LONG value = SameLock( BPTR lock1, BPTR lock2 );
```

```
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>

AMIGATALK INTERFACE (SafeDOS Class):

areSameLock: bptrLock1 and: bptrLock2
```

1.15 sameDevice (SAFE):

```
NAME
    SameDevice -- Are two locks on the same partition of
                  the device? (V37)

SYNOPSIS
    BOOL same = SameDevice( BPTR lock1, BPTR lock2 );

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

AMIGATALK INTERFACE (SafeDOS Class):
```

```
areSameDevice: bptrLock1 and: bptrLock2
```

1.16 readLink (SAFE):

```
                NAME
    ReadLink -- Reads the path for a soft filesystem link

SYNOPSIS
    BOOL success = ReadLink( struct MsgPort *port, BPTR lock,
                             char *path, char *buffer,
                             ULONG size );

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.


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

SEE ALSO
     MakeLink ,  Open ,
     Lock ,
              GetDeviceProc

AMIGATALK INTERFACE (SafeDOS Class):

readLinkInto: aBuffer ofSize: length onPort: msgPort
       using: bptrLock   and: pathName
```

1.17 readItem (SAFE):

```
                NAME
    ReadItem - reads a single argument/name from command line

SYNOPSIS
```

```
    LONG value = ReadItem( char *buffer, LONG maxchars,
                           struct CSource *input );

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 unreads 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.

BUGS
    Doesn't actually unread the terminator.

SEE ALSO

              ReadArgs
             ,  FindArg ,

              UnGetC
             ,
              FGetC
             ,
      Input ,  FreeArgs ,
    <dos/dos.h>, <dos/rdargs.h>

AMIGATALK INTERFACE (SafeDOS Class):

readItemInto: aBuffer ofSize: maxChars with: csourceInput
```

1.18 readArgs (SAFE):

```
                NAME
    ReadArgs - Parse the command line input

SYNOPSIS
    struct RDArgs *result = ReadArgs( char          *template,
                                      LONG          *array,
                                      struct RDArgs *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 RDArgs if it succeeds.  This serves as an
"anchor" to allow  FreeArgs()  to free the associated memory.  You can
also pass in a struct RDArgs 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 RDArgs, you must
still call FreeArgs() to release storage that gets attached to it,
but you are responsible for freeing the RDArgs yourself.

If you pass in a RDArgs structure, you MUST reset (clear or set)
RDA_Buffer for each new call to RDArgs.  The exact behavior if you
don't do this varies from release to release and case to case; don't
count on the behavior!

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 RDArgs 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,xyzzy/A" would require
 "xyzzy=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

AMIGATALK INTERFACE (SafeDOS Class):

readArgs: template into: stringPointerArray auxRDArgs: rdArgs
```

1.19 readFile (SAFE):

```
                NAME
    Read -- Read bytes of data from a file

SYNOPSIS
    LONG actualLength = Read( BPTR file, char *buffer, LONG length );


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

The value returned is the length of the information actually read.
So, when 'actualLength' is greater than zero, the value of
'actualLength' is the the number of characters read. Usually Read
will try to fill up your buffer before returning. A value of zero
means that end-of-file has been reached. Errors are indicated by a
value of -1.

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

RESULT
    actualLength - integer

SEE ALSO
     Open ,  Close ,
     Write ,  Seek ,

              FGetC

AMIGATALK INTERFACE (SafeDOS Class):

read: bptrFileHandle into: aBuffer ofSize: length
```

1.20 putStr (SAFE):

```
                NAME
    PutStr -- Writes a string the the default output (buffered)

SYNOPSIS
    LONG error = PutStr( char *str );

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

AMIGATALK INTERFACE (SafeDOS Class):

putStr: aString
```

1.21 printFault (SAFE):

```
                NAME
    PrintFault -- Returns the text associated with a DOS error code

SYNOPSIS
    BOOL success = PrintFault( LONG code, char *header );

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

SEE ALSO

               IoErr
              ,
               Fault
              ,

               SetIoErr
```

```
              ,  Output ,

               FPuts

AMIGATALK INTERFACE (SafeDOS Class):

printFault: header code: c
```

1.22 pathPart (SAFE):

```
                NAME
    PathPart -- Returns a pointer to the end of the next-to-last
                component of a path.
SYNOPSIS
    char *fileptr = PathPart( char *path );

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

AMIGATALK INTERFACE (SafeDOS Class):

getPathPart: pathAndFile   " Tested "
```

1.23 parentOfFH (SAFE):

```
                NAME
    ParentOfFH -- returns a lock on the parent directory of a file

SYNOPSIS
    BPTR lock = ParentOfFH( BPTR fh );
```

```
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

AMIGATALK INTERFACE (SafeDOS Class):

getParentLockFromFH: fromBPTRFileHandle
```

1.24 parentDir (SAFE):

```
                NAME
    ParentDir -- Obtain the parent of a directory or file

SYNOPSIS
    BPTR newlock = ParentDir( BPTR lock )


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

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

INPUTS
    lock - BCPL pointer to a lock

RESULT
    newlock - BCPL pointer to a lock

SEE ALSO
     Lock ,  DupLock ,
     UnLock ,
               ParentOfFH
             ,
     DupLockFromFH

AMIGATALK INTERFACE (SafeDOS Class):
```

```
getParentDirLock: fromBPTRLock
```

1.25 maxCli (SAFE):

```
                NAME
    MaxCli -- returns the highest CLI process number possibly in use

SYNOPSIS
    LONG number = 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

AMIGATALK INTERFACE (SafeDOS Class):

getMaxCli
```

1.26 matchNext (SAFE):

```
                NAME
    MatchNext - Finds the next file or directory that matches pattern

SYNOPSIS
    LONG error = MatchNext( struct AnchorPath *ap );

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
             ,

              MatchEnd
             ,  ExNext ,
    <dos/dosasl.h>

AMIGATALK INTERFACE (SafeDOS Class):

matchNext: anchorPath
```

1.27 matchFirst (SAFE):

```
                NAME
    MatchFirst -- Finds file that matches pattern

SYNOPSIS
    LONG error = MatchFirst( char *pat, struct AnchorPath *ap );

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 (particularily 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
              ,

                MatchEnd
              ,  ExNext ,
      <dos/dosasl.h>

  AMIGATALK INTERFACE (SafeDOS Class):

  matchFirst: pattern fromAnchor: anchorPath
```

1.28 matchEnd (SAFE):

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

  SYNOPSIS
      void MatchEnd( struct AnchorPath *ap );

  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
              ,

                MatchNext
              ,  ExNext ,
      <dos/dosasl.h>

  AMIGATALK INTERFACE (SafeDOS Class):

  matchEnd: anchorPath
```

1.29 isInteractive (SAFE):

```
NAME
    IsInteractive -- Discover whether a file is "interactive"

SYNOPSIS
    BOOL status = IsInteractive( BPTR file )

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

INPUTS
    file - BCPL pointer to a file handle

AMIGATALK INTERFACE (SafeDOS Class):

isInteractive: bptrFileHandle
```

1.30 isFileSystem (SAFE):

```
NAME
    IsFileSystem -- returns whether a Dos handler is a filesystem

SYNOPSIS
    BOOL result = IsFileSystem( char *name )

FUNCTION
    Returns whether the device is a filesystem or not.  A filesystem
    supports seperate 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

AMIGATALK INTERFACE (SafeDOS Class):

ifFileSystem: name
```

1.31 ioErr (SAFE):

```
                NAME
    IoErr -- Return extra information from the system

SYNOPSIS
```

```
    LONG error = 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 specified by the routine.

RESULT
    error - integer

SEE ALSO

             Fault
            ,
             PrintFault
            ,

             SetIoErr

AMIGATALK INTERFACE (SafeDOS Class):

getIoErr
```

1.32 getVar (SAFE):

```
                NAME
    GetVar -- Returns the value of a local or global variable

SYNOPSIS
    LONG len = GetVar( char *name, char *buffer, LONG size, LONG flags );

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 newline is hit, unless
GVF_BINARY_VAR is specified.  (The newline 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),
             & 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 newline
```

```
          GVF_DONT_NULL_TERM - no null termination (only valid
                               for binary variables). (V37)

    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, unless GVF_DONT_NULL_TERM is in effect).  If it
    succeeds, len is the number of characters put in the buffer
    (not including null termination), and IoErr() will return the
    the size of the variable (regardless of buffer size).

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().
GVF_DONT_NULL_TERM only works for local variables under V37.  For
V39, it also works for globals.

SEE ALSO
     SetVar ,  DeleteVar ,

               FindVar
              , <dos/var.h>

AMIGATALK INTERFACE (SafeDOS Class):

getVarNamed: name into: aBuffer ofSize: size flags: flags
```

1.33 getPrompt (SAFE):

```
                NAME
    GetPrompt -- Returns the prompt for the current process

SYNOPSIS
    BOOL success = GetPrompt( char *buf, LONG len );

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

SEE ALSO

              SetPrompt

AMIGATALK INTERFACE (SafeDOS Class):

getPromptInto: aBuffer ofSize: length
```

1.34 getProgramName (SAFE):

```
                NAME
    GetProgramName -- Returns the current program name

SYNOPSIS
    BOOL success = GetProgramName( char *buf, LONG len )

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.
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

SEE ALSO
     SetProgramName

AMIGATALK INTERFACE (SafeDOS Class):

getProgramNameInto: aBuffer ofSize: length
```

1.35 getProgramDir (SAFE):

```
NAME
    GetProgramDir -- Returns a lock on the directory the program was loaded
                     from
SYNOPSIS
```

```
    BPTR lock = 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

AMIGATALK INTERFACE (SafeDOS Class):

getProgramDir
```

1.36 getFileSysTask (SAFE):

```
NAME
    GetFileSysTask -- Returns the default filesystem for the process

SYNOPSIS
    struct MsgPort *port = 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

AMIGATALK INTERFACE (SafeDOS Class):

getFileSysTask
```

1.37 getDeviceProc (SAFE):

```
NAME
    GetDeviceProc -- Finds a handler to send a message to

SYNOPSIS
```

```
    struct DevProc *devproc = GetDeviceProc( char *name, struct DevProc * ↩
       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->dvp_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

AMIGATALK INTERFACE (SafeDOS Class):

getDeviceProc: name auxDevProc: devProc
```

1.38 getCurrentDirName (SAFE):

```
                NAME
    GetCurrentDirName -- returns the current directory name

SYNOPSIS
    BOOL success = GetCurrentDirName( char *buf, LONG len );
```

```
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

AMIGATALK INTERFACE (SafeDOS Class):

getCurrentDirNameInto: aBuffer ofSize: length
```

1.39 getConsoleTask (SAFE):

```
NAME
    GetConsoleTask -- Returns the default console for the process

SYNOPSIS
    struct MsgPort *port = 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

AMIGATALK INTERFACE (SafeDOS Class):

getConsoleTask
```

1.40 getArgStr (SAFE):

```
NAME
    GetArgStr -- Returns the arguments for the process

SYNOPSIS
    char *ptr = 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

AMIGATALK INTERFACE (SafeDOS Class):

getArgStr
```

1.41 fPutS (SAFE):

```
                NAME
    FPuts -- Writes a string the the specified output (buffered)

SYNOPSIS
    LONG error = FPuts( BPTR fh, char *str );

FUNCTION
    This routine writes an unformatted string to the filehandle.  No
    newline is appended to the string.  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

AMIGATALK INTERFACE (SafeDOS Class):

fPutS: aString to: bptrFileHandle
```

1.42 fPutC (SAFE):

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

SYNOPSIS
    LONG char = FPutC( BPTR fh, LONG chr );


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.

BUGS
    Older autodocs indicated that you should pass a UBYTE.  The
    correct usage is to pass a LONG in the range 0-255.

SEE ALSO

              FGetC
             ,
              UnGetC
             ,
     Flush

AMIGATALK INTERFACE (SafeDOS Class):

fPutC: theChar to: bptrFileHandle
```

1.43 findVar (SAFE):

```
                NAME
    FindVar -- Finds a local variable

SYNOPSIS
    struct LocalVar *var = FindVar( char *name, ULONG type );

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>

AMIGATALK INTERFACE (SafeDOS Class):

findVar: varName ofType: type
```

1.44 findCliProc (SAFE):

```
                NAME
    FindCliProc -- returns a pointer to the requested CLI process

SYNOPSIS
    struct Process *proc = FindCliProc( ULONG num );

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 (range 1-N)

RESULT
    proc - Pointer to given CLI process

SEE ALSO

               Cli
             , Forbid,
               MaxCli

AMIGATALK INTERFACE (SafeDOS Class):

findCliProc: numbered
```

1.45 filePart (SAFE):

```
                NAME
    FilePart -- Returns the last component of a path
```

```
SYNOPSIS
    char *fileptr = FilePart( char *path );

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

SEE ALSO

              PathPart
             ,  AddPart

AMIGATALK INTERFACE (SafeDOS Class):

getFilePart: pathAndFile   " Tested "
```

1.46 fGetS (SAFE):

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

SYNOPSIS
    char *buffer = FGets( BPTR fh, char *buf, ULONG len );

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 returnd for an EOF,

              IoErr()
              will return 0.

BUGS
    In V36 and V37, it copies one more byte than it should if it doesn't
 hit an EOF or newline.  In the example above, it would copy 50 bytes
 and put a null in the 51st.  This is fixed in dos V39.  Workaround
 for V36/V37: pass in buffersize-1.

SEE ALSO
     FRead ,
              FPuts
             ,
              FGetC

AMIGATALK INTERFACE (SafeDOS Class):

fGets: fromBPTRFileHandle into: aBuffer ofSize: length using: flag

   If flag is 0, then a newline will be left on the end of the
   returned String, a value of 1 will replace the last newline
   with a value of 0.
```

1.47 fGetC (SAFE):

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

SYNOPSIS
    LONG char = FGetC( BPTR fh );

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

AMIGATALK INTERFACE (SafeDOS Class):

fGetC: fromBPTRFileHandle
```

1.48 fault (SAFE):

```
                NAME
    Fault -- Returns the text associated with a DOS error code

SYNOPSIS
    LONG len = Fault( LONG code, char *header, char *buffer, LONG len );


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>".

The number of characters put into the buffer is returned, which will
be 0 if the code passed in was 0.

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

RESULT
    len    - number of characters put into buffer (may be 0)

SEE ALSO

              IoErr
             ,
              SetIoErr
             ,

              PrintFault
                BUGS
    In older documentation, the return was shown as BOOL success.
    This was incorrect, it has always returned the length.


AMIGATALK INTERFACE (SafeDOS Class):
```

```
fault: header code: c into: aBuffer ofSize: length
```

1.49 ErrorReport (SAFE):

```
                NAME
    ErrorReport -- Displays a Retry/Cancel requester for an error

SYNOPSIS
    BOOL status = ErrorReport( LONG code, LONG type,
                               ULONG arg1, struct MsgPort *device );

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

AMIGATALK INTERFACE (SafeDOS Class):

errorReport: code type: t arg1: a1 fromDevicePort: msgPort
```

1.50 endNotify (SAFE):

```
NAME
    EndNotify -- Ends a notification request

SYNOPSIS
    void EndNotify( struct NotifyRequest *notifystructure );

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>

AMIGATALK INTERFACE (SafeDOS Class):

endNotify: notifyRequest
```

1.51 delay (SAFE):

```
NAME
    Delay -- Delay a process for a specified time

SYNOPSIS
    void Delay( ULONG ticks );

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.


AMIGATALK INTERFACE (SafeDOS Class):

delay: ticks
```

1.52 dateToStr (SAFE):

```
                NAME
    DateToStr -- Converts a DateStamp to a string

SYNOPSIS
    BOOL success = DateToStr( struct DateTime *datetime );

FUNCTION
    DateToStr 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>

AMIGATALK INTERFACE (SafeDOS Class):

dateToStr: dateTime
```

1.53 currentDir (SAFE):

```
NAME
    CurrentDir -- Make a directory lock the current directory

SYNOPSIS
    BPTR oldLock = CurrentDir( BPTR lock );

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

A value of zero is a valid result here, this 0 lock represents the
root of file system that you booted from.

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

RESULT
    oldLock - BCPL pointer to a lock

SEE ALSO
     Lock ,  UnLock ,
     Open ,  DupLock
```

```
AMIGATALK INTERFACE (SafeDOS Class):

currentDir: fromBPTRLock
```

1.54 compareDates (SAFE):

```
                NAME
    CompareDates -- Compares two datestamps

SYNOPSIS
    LONG result = CompareDates( struct DateStamp *date1,
                                struct DateStamp *date2 );

FUNCTION
    Compares two times for relative magnitide.  < 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

AMIGATALK INTERFACE (SafeDOS Class):

compareDates: dateStamp1 and: dateStamp2
```

1.55 cliPointer (SAFE):

```
NAME
    Cli -- Returns a pointer to the CLI structure of the process

SYNOPSIS
    struct CommandLineInterface *cli_ptr = 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.

AMIGATALK INTERFACE (SafeDOS Class):
```

```
getCLIObject
```

1.56 addBuffers (SAFE):

```
                NAME
    AddBuffers -- Changes the number of buffers for a filesystem

SYNOPSIS
    BOOL success = AddBuffers( char *filesystem, LONG number );

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

AMIGATALK INTERFACE (SafeDOS Class):

addBuffers: howMany toFileDevice: diskDrive
```

1.57 AbortPacket (SAFE):

```
NAME
    AbortPkt -- Aborts an asynchronous packet, if possible.
```

```
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

AMIGATALK INTERFACE (SafeDOS Class):

abortPacket: dosPacket onMsgPort: msgPort
```