# amiga.lib

**COLLABORATORS**

| | *TITLE* :<br><br>amiga.lib | |
|---|---|---|

| *ACTION* | *NAME* | *DATE* | *SIGNATURE* |
|---|---|---|---|
| WRITTEN BY | | March 28, 2025 | |

**REVISION HISTORY**

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

# Contents

# Chapter 1

# amiga.lib

## 1.1 amiga.lib.doc

```
 AddTOF()           DeleteTask()      NewList()
 BeginIO()          FastRand()        printf()
 CreateExtIO()      afp()             RangeRand()
 CreatePort()       arnd()            RemTOF()
 CreateTask()       dbf()             sprintf()
 DeleteExtIO()      fpa()             stdio()
 DeletePort()       fpbcd()
```

## 1.2 amiga.lib/AddTOF

```
NAME
    AddTOF - add a task to the TopOfFrame Interrupt server chain.

SYNOPSIS
    AddTOF(i,p,a);
    void AddTOF(struct Isrvstr *, APTR, APTR);

FUNCTION
    Adds a task to the vertical-blanking interval interrupt server
    chain.  This prevents C programmers from needing to write an
    assembly language stub to do this function.

INPUTS
    i - pointer to structure Isrvstr.
    p - pointer to the C-code routine that this server is to call each
        time TOF happens.
    a - pointer to the first longword in an array of longwords that
        is to be used as the arguments passed to your routine
        pointed to by p.

SEE ALSO
    RemTOF, graphics/graphint.h
```

## 1.3 amiga.lib/BeginIO

```
NAME
    BeginIO -- initiate asynchronous I/O

SYNOPSIS
    BeginIO(iORequest)
    void BeginIO(struct IORequest *);

FUNCTION
    This function takes an IORequest, and passes it directly to the
    BEGINIO vector of the proper device.  This works exactly like
    SendIO, but does not clear the io_Flags field first.

    This function does not wait for the I/O to complete.

INPUTS
    iORequest - Pointer to an initialized, open IORequest structure
                with the io_Flags field set to a reasonable value
                (use zero if you do not require io_Flags).

SEE ALSO
    exec/DoIO, exec/SendIO, exec/WaitIO
```

## 1.4 amiga.lib/CreateExtIO

```
NAME
    CreateExtIO() -- create an IORequest structure

SYNOPSIS
    ioReq = CreateExtIO( ioReplyPort, size );
    struct IORequest *CreateExtIO(struct MsgPort *, ULONG);

FUNCTION
    Allocates memory for and initializes a new IO request block
    of a user-specified number of bytes.  The number of bytes
    MUST be the size of a legal IORequest (or extended IORequest)
    or very nasty things will happen.

INPUTS
    ioReplyPort - a pointer to an already initialized
            message port to be used for this IO request's reply port.
            (usually created by CreatePort()).
    size - the size of the IO request to be created.

RESULT
    Returns a pointer to the new IO Request block, or NULL if
            the request failed.

SEE ALSO
    CreatePort, DeleteExtIO
    CreateIORequest
```

## 1.5 amiga.lib/CreatePort

```
NAME
    CreatePort - Allocate and initialize a new message port

SYNOPSIS
    CreatePort(name,pri)
    struct MsgPort *CreatePort(char *,LONG);

FUNCTION
    Allocates and initializes a new message port.  The message list
    of the new port will be prepared for use (via NewList).  The port
    will be set to signal your task when a message arrives (PA_SIGNAL).

INPUTS
    name - NULL if other tasks will not search for this port
           via the FindPort() call.  If non-null, this must be
           a null-terminated string; the port will be added to
           the system public port list.  The name is not copied.
    pri  - Priority used for insertion into the public port list.

RESULT
    A new MsgPort structure ready for use.

SEE ALSO
    DeletePort, exec/FindPort, exec/ports.h
```

## 1.6 amiga.lib/CreateTask

```
NAME
    CreateTask -- Create task with given name, priority, stacksize

SYNOPSIS
    CreateTask( name, pri, initPC, stackSize)
    task=(struct Task *)CreateTask(char *, LONG, funcEntry, ULONG);

FUNCTION
    This function simplifies program creation of subtasks by
    dynamically allocating and initializing required structures
    and stack space, and adding the task to Exec's task list
    with the given name and priority.  A tc_MemEntry list is provided
    so that all stack and structure memory allocated by CreateTask
    is automatically deallocated when the task is removed.

    An Exec task may not call dos.library functions or any function
    which might cause the loading of a disk-resident library, device,
    or file (since such functions are indirectly calls to dos.library).
    Only AmigaDOS Processes may call AmigaDOS; see the DOS CreateProc()
    call for more information.

    If other tasks or processes will need to find this task by name,
    provide a complex and unique name to avoid conflicts.

    If your compiler provides automatic insertion of stack-checking
```

```
    code, you may need to disable this feature when compiling subtask
    code since the stack for the subtask is at a dynamically allocated
    location.  If your compiler requires 68000 registers to contain
    particular values for base relative addressing, you may need to
    save these registers from your main process, and restore them
    in your initial subtask code.

    The function entry initPC is generally provided as follows:

    In C:
    extern void functionName();
    char *tname = "unique name";
    task = CreateTask(tname, 0L, functionName, 4000L);

    In assembler:
            PEA     startLabel

INPUTS
    name - a null terminated string.
    pri - an Exec task priority between -128 and 127 (commonly 0)
    funcEntry - the address of the first executable instruction
                of the subtask code.
    stackSize - size in bytes of stack for the subtask.  Don't cut it
                too close - system function stack usage may change.

SEE ALSO
    DeleteTask, exec/FindTask
```

## 1.7 amiga.lib/DeleteExtIO

```
NAME
    DeleteExtIO() - return memory allocated for extended IO request

SYNOPSIS
    DeleteExtIO( ioReq );
    void DeleteExtIO(struct IORequest *);

FUNCTION
    Frees up an IO request as allocated by CreateExtIO().  By
    looking at the mn_Length field, it knows how much memory
    to deallocate.

INPUTS
    ioReq - A pointer to the IORequest block to be freed.

SEE ALSO
    CreateExtIO
```

## 1.8 amiga.lib/DeletePort

```
NAME
    DeletePort - Free a message port created by CreatePort
```

```
SYNOPSIS
    DeletePort(msgPort)
    void DeletePort(struct MsgPort *);

FUNCTION
    Frees a message port created by CreatePort.  All messages that
    may have been attached to this port must have already been
    replied to.

INPUTS
    msgPort - A message port

SEE ALSO
    CreatePort
```

## 1.9 amiga.lib/DeleteTask

```
NAME
    DeleteTask -- Delete a task created with CreateTask

SYNOPSIS
    DeleteTask( task )
    void DeleteTask(struct Task *);

FUNCTION
    This function simply calls exec/RemTask, deleting a task from the
    Exec task lists and automatically freeing any stack and
    structure memory allocated for it by CreateTask.

    Before deleting a task, you must first make sure that the task is
    not currently executing any system code which might try to signal
    the task after it is gone.

    This can be accomplished by stopping all sources that might reference
    the doomed task, then causing the subtask execute a Wait(0L).  Another
    option is to have have the task DeleteTask()/RemTask() itself.

INPUTS
    task - pointer to a Task

SEE ALSO
    CreateTask, exec/RemTask
```

## 1.10 amiga.lib/FastRand

```
NAME
    FastRand - quickly generate a somewhat random integer

SYNOPSIS
    number = FastRand(seed);
    ULONG FastRand(ULONG);
```

```
FUNCTION
    C-implementation only.   Seed value is taken from stack, shifted
    left one position, exclusive-or'ed with hex value $1D872B41 and
    returned (D0).

INPUTS
    seed - a 32-bit integer

RESULT
    number - new random seed, a 32-bit value

SEE ALSO
    RangeRand
```

## 1.11 amiga.lib/math/afp

```
NAME
    afp - Convert ASCII string variable into fast floating point

USAGE
    ffp_value = afp(string);

FUNCTION
    Accepts the address of the ASCII string in C format that is
    converted into an FFP floating point number.

    The string is expected in this Format:
    {S}{digits}{'.'}{digits}{'E'}{S}{digits}
    <*******MANTISSA*******><***EXPONENT***>


    Syntax rules:
    Both signs are optional and are '+' or '-'. The mantissa must be
    present. The exponent need not be present. The mantissa may lead
    with a decimal point. The mantissa need not have a decimal point.
    Examples: All of these values represent the number fourty-two.
                 42            .042e3
                 42.          +.042e+03
                +42.          0.000042e6
            0000042.00       420000e-4
                          420000.00e-0004

    Floating point range:
    Fast floating point supports the value zero and non-zero values
    within the following bounds -
                    18                              20
     9.22337177 x 10   > +number >  5.42101070 x 10
                    18                             -20
    -9.22337177 x 10   > -number > -2.71050535 x 10

    Precision:
    This conversion results in a 24 bit precision with guaranteed
    error less than or equal to one-half least significant bit.
```

```
    INPUTS
    string - Pointer to the ASCII string to be converted.


    OUTPUTS
    string - points to the character which terminated the scan
    equ - fast floating point equivalent
```

## 1.12 amiga.lib/math/arnd

```
NAME
    arnd - ASCII round of the provided floating point string

USAGE
    arnd(place, exp, &string[0]);

FUNCTION
    Accepts an ASCII string representing an FFP floating point
    number, the binary representation of the exponent of said
    floating point number and the number of places to round to.
    A rounding process is initiated, either to the left or right
    of the decimal place and the result placed back at the
    input address defined by &string[0].

INPUTS
    place - integer representing number of decimal places to round to
    exp - integer representing exponent value of the ASCII string
    &string[0] - address where rounded ASCII string is to be placed
                 (16 bytes)

RESULT
    &string[0] - rounded ASCII string

BUGS
    None
```

## 1.13 amiga.lib/math/dbf

```
NAME
    dbf - convert FFP dual-binary number to FFP format

USAGE
    fnum = dbf(exp, mant);

FUNCTION
    Accepts a dual-binary format (described below) floating point
    number and converts it to an FFP format floating point number.
    The dual-binary format is defined as:

            exp bit  16     = sign (0=>positive, 1=>negative)
            exp bits 15-0   = binary integer representing the base
                                    ten (10) exponent
```

```
            man             = binary integer mantissa

INPUTS
    exp - binary integer representing sign and exponent
    mant - binary integer representing the mantissa

RESULT
    fnum - converted FFP floating point format number

BUGS
    None
```

## 1.14 amiga.lib/math/fpa

```
NAME
    fpa - convert fast floating point into ASCII string equivalent

USAGE
    exp = fpa(fnum, &string[0]);

FUNCTION
    Accepts an FFP number and the address of the ASCII string where it's
    onverted output is to be stored.  The number is converted to a NULL
    terminated ASCII string in and stored at the address provided.
    Additionally, the base ten (10) exponent in binary form is returned.

INPUTS
    fnum       - Motorola Fast Floating Point number
    &string[0] - address for output of converted ASCII character string
                 (16 bytes)

RESULT
    &string[0] - converted ASCII character string
    exp        - integer exponent value in binary form

BUGS
    None
```

## 1.15 amiga.lib/math/fpbcd

```
NAME
    fpbcd - convert FFP floating point number to BCD format

USAGE
    fpbcd(fnum, &string[0]);

FUNCTION
    Accepts a floating point number and the address where the
    converted BCD data is to be stored.  The FFP number is
    converted and stored at the specified address in an ASCII
    form in accordance with the following format:
```

```
            MMMM S E S B

    Where:  M = Four bytes of BCD, each with two (2) digits of
                the mantissa (8 digits)
            S = Sign of mantissa (0x00 = positive, 0xFF = negative)
            E = BCD byte for two (2) digit exponent
            S = Sign of exponent (0x00 = positive, 0xFF = negative)
            B = One (1) byte binary two's compliment representation
                    of the exponent

INPUTS
    fnum - floating point number
    &string[0] - address where converted BCD data is to be placed

RESULT
    &string[0] - converted BCD data
```

## 1.16 amiga.lib/NewList

```
NAME
    NewList -- prepare a list structure for use

SYNOPSIS
    NewList(list*)
    void NewList(struct List *);

FUNCTION
    Prepare a List structure for use; the list will be empty and
    ready to use.

    This function prepares the lh_Head, lh_Tail and lh_TailPred fields.
    You are responsible for initializing lh_Type.  Assembly programmers
    will want to use the NEWLIST macro instead.

INPUTS
    list - Pointer to a List

SEE ALSO
    exec/lists.h
```

## 1.17 amiga.lib/printf

```
NAME
    printf - print a formatted output line to the standard output.

SYNOPSIS
    printf( formatstring [,value [,values] ] );

FUNCTION
    Format the output in accordance with specifications in the format
    string:
```

```
INPUTS
    formatstring - a pointer to a null-terminated string describing the
                   output data, and locations for parameter substitutions.
    value(s) - numeric variables or addresses of null-terminated strings
               to be added to the format information.

    The function printf can handle the following format conversions, in
    common with the normal C language call to printf:

    %c  - the next long word in the array is to be formatted
          as a character (8-bit) value
    %d  - the next long word in the array is to be formatted
          as a decimal number
    %x  - the next long word in the array is to be formatted
          as a hexadecimal number
    %s  - the next long word is the starting address of a
          null-terminated string of characters

    And "l" (small-L) character must be added between the % and the letter
    if the value is a long (32 bits) or if the compiler in use forces
    passed paramters to 32 bits.

    Floating point output is not supported.

    Following the %, you may also specify:

    o       an optional minus (-) sign that tells the formatter
            to left-justify the formatted item within the field
            width

    o       an optional field-width specifier... that is, how
            many spaces to allot for the full width of this
            item.  If the field width specifier begins with
            a zero (0), it means that leading spaces, ahead of
            the formatted item (usually a number) are to be
            zero-filled instead of blank-filled

    o       an optional period (.) that separates the width
            specifier from a maximum number of characters
            specifier

    o       an optional digit string (for %ls specifications
            only) that specifies the maximum number of characters
            to print from a string.

    See other books on C language programming for examples of the use
    of these formatting options (see "printf" in other books).

NOTE
    The global "_stdout" must be defined, and contain a pointer to
    a legal AmigaDOS file handle.  Using the standard Amiga startup
    module sets this up.  In other cases you will need to define
    stdout, and assign it to some reasonable value (like what the
    AmigaDOS Output() call returns).  This code would set it up:

    ULONG stdout;
```

```
            stdout=Output();
```

## 1.18 amiga.lib/RangeRand

```
NAME
    RangeRand - To obtain a random number within a specific integer range
                of 0 to value.

SYNOPSIS
    number = RangeRand(value);

FUNCTION
    RangeRand accepts a value from 1 to 65535, and returns a value
    within that range. (16-bit integer).  Note:  C-language implementation.

    Value is passed on stack as a 32-bit integer but used as though
    it is only a 16-bit integer.  Variable named RangeSeed is available
    beginning with V1.2 that contains the global seed value passed from
    call to call and thus can be changed by a program by declaring::

           extern ULONG RangeSeed;

INPUTS
    value - integer in the range of 1 to 65535.

RESULT
    number - pseudo random integer in the range of 1 to <value>.

SEE ALSO
    FastRand
```

## 1.19 amiga.lib/RemTOF

```
NAME
    RemTOF - Remove a task from the TopOfFrame interrupt server chain.

SYNOPSIS
    RemTOF(i);
    void RemTOF(struct Isrvstr *);

FUNCTION
    To remove a task from the vertical-blanking interval interrupt server
    chain.

INPUTS
    i - pointer to structure Isrvstr.

SEE ALSO
    AddTOF,graphics/graphinit.h
```

## 1.20 amiga.lib/sprintf

```
NAME
    sprintf - format a C-like string into a string buffer

SYNOPSIS
    sprintf( destination, formatstring [,value [, values] ] );

FUNCTION
    perform string formatting identical to printf, but direct the output
    into a specific destination in memory.  This uses the ROM version
    of printf, so it is very small.

    Assembly programmers can call this by placing values on the
    stack, followed by a pointer to the formatstring, followed
    by a pointer to the destination string.

INPUTS
    destination - the address of an area in memory into which the
                  formatted output is to be placed.
    formatstring - pointer to a null terminated string describing the
                   desired output formatting.
    value(s) - numeric information to be formatted into the output
               stream.

SEE ALSO
    printf, exec/RawDoFmt
```

## 1.21 amiga.lib/stdio

```
NAMES
    fclose  - close file
    fgetc   - get a character from a file
    fprintf - format data to file (see exec.library/RawDoFmt)
    fputc   - put character to file
    fputs   - write string to file
    getchar - get a character from stdin
    printf        - put format data to stdout (see exec.library/RawDoFmt)
    putchar - put character to stdout
    puts    - put string to stdout, followed by newline
    sprintf      - format data into string (see exec.library/RawDoFmt)

FUNCTION
    These functions work much like the standard C functions of the same
    names.  The file I/O functions all use non-buffered AmigaDOS
    filehandles, and must not be mixed with the file I/O of any C
    compiler.  The names of these function match those found in many
    standard C libraries, when a name conflict occurs, the function is
    generally taken from the FIRST library that was specified on the
    linker's command line.  Thus to use these functions, specify
    the amiga.lib library first.

    To get a suitable AmigaDOS filehandle, the AmigaDOS Open()
    function must be used.
```

```
    All of the functions that write to stdout expect an appropriate
    filehandle to have been set up ahead of time.  Depending on
    your C compiler and options, this may have been done by the
    startup code.  Or it can be done manually:

FROM C:
    extern ULONG stdout;
    /* Remove the extern if startup code did not define stdout */
            stdout=Output();

FROM ASSEMBLY:
            XDEF    _stdout
            DC.L    _stdout ;<- Place result of dos.library Output() here.
```