

**amiga\_lib**

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

	<i>TITLE :</i> amiga_lib		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY		July 19, 2024	

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

NUMBER	DATE	DESCRIPTION	NAME

# Contents

<b>1</b>	<b>amiga_lib</b>	<b>1</b>
1.1	amiga_lib.doc	1
1.2	amiga.lib/ACrypt	2
1.3	amiga.lib/AddTOF	3
1.4	amiga.lib/afp	3
1.5	amiga.lib/ArgArrayDone	4
1.6	amiga.lib/ArgArrayInit	5
1.7	amiga.lib/ArgInt	6
1.8	amiga.lib/ArgString	7
1.9	amiga.lib/arnd	7
1.10	amiga.lib/BeginIO	8
1.11	amiga.lib/CallHook	8
1.12	amiga.lib/CallHookA	9
1.13	amiga.lib/CheckRexxMsg	10
1.14	amiga.lib/CoerceMethod	11
1.15	amiga.lib/CoerceMethodA	11
1.16	amiga.lib/CreateExtIO	12
1.17	amiga.lib/CreatePort	12
1.18	amiga.lib/CreateStdIO	13
1.19	amiga.lib/CreateTask	14
1.20	amiga.lib/CxCustom	15
1.21	amiga.lib/CxDebug	16
1.22	amiga.lib/CxFilter	16
1.23	amiga.lib/CxSender	17
1.24	amiga.lib/CxSignal	18
1.25	amiga.lib/CxTranslate	19
1.26	amiga.lib/dbf	19
1.27	amiga.lib/DeleteExtIO	20
1.28	amiga.lib/DeletePort	20
1.29	amiga.lib/DeleteStdIO	21

---

1.30	amiga.lib/DeleteTask . . . . .	21
1.31	amiga.lib/DoMethod . . . . .	22
1.32	amiga.lib/DoMethodA . . . . .	22
1.33	amiga.lib/DoSuperMethod . . . . .	23
1.34	amiga.lib/DoSuperMethodA . . . . .	24
1.35	amiga.lib/FastRand . . . . .	24
1.36	amiga.lib/fpa . . . . .	25
1.37	amiga.lib/FreeIEvents . . . . .	25
1.38	amiga.lib/GetRexxVar . . . . .	26
1.39	amiga.lib/HookEntry . . . . .	27
1.40	amiga.lib/HotKey . . . . .	28
1.41	amiga.lib/InvertString . . . . .	29
1.42	amiga.lib/NewList . . . . .	29
1.43	amiga.lib/printf . . . . .	30
1.44	amiga.lib/RangeRand . . . . .	31
1.45	amiga.lib/RemTOF . . . . .	31
1.46	amiga.lib/SetRexxVar . . . . .	32
1.47	amiga.lib/SetSuperAttrs . . . . .	33
1.48	amiga.lib/sprintf . . . . .	34
1.49	amiga.lib/stdio . . . . .	34
1.50	amiga.lib/TimeDelay . . . . .	35
1.51	pools.lib/LibAllocPooled . . . . .	36
1.52	pools.lib/LibCreatePool . . . . .	37
1.53	pools.lib/LibDeletePool . . . . .	38
1.54	pools.lib/LibFreePooled . . . . .	39

# Chapter 1

## amiga\_lib

### 1.1 amiga\_lib.doc

```
ACrypt ()
AddTOF ()
afp ()
ArgArrayDone ()
ArgArrayInit ()
ArgInt ()
ArgString ()
arnd ()
BeginIO ()
CallHook ()
CallHookA ()
CheckRexxMsg ()
CoerceMethod ()
CoerceMethodA ()
CreateExtIO ()
CreatePort ()
CreateStdIO ()
CreateTask ()
CxCustom ()
CxDebug ()
CxFilter ()
CxSender ()
CxSignal ()
CxTranslate ()
dbf ()
DeleteExtIO ()
DeletePort ()
DeleteStdIO ()
DeleteTask ()
DoMethod ()
DoMethodA ()
DoSuperMethod ()
DoSuperMethodA ()
FastRand ()
fpa ()
FreeIEvents ()
GetRexxVar ()
HookEntry ()
```

---

```

HotKey()
InvertString()
NewList()
printf()
RangeRand()
RemTOF()
SetRexxVar()
SetSuperAttrs()
sprintf()
stdio()
TimeDelay()
LibAllocPooled()
LibCreatePool()
LibDeletePool()
LibFreePooled()

```

## 1.2 amiga.lib/ACrypt

### NAME

ACrypt -- Encrypt a password (V37)

### SYNOPSIS

```
newpass = ACrypt( buffer, password, username )
```

```
STRPTR ACrypt( STRPTR, STRPTR, STRPTR);
```

### FUNCTION

This function takes a buffer of at least 12 characters in length, an unencrypted password and the user's name (as known to the host system) and returns an encrypted password in the passed buffer. This is a one-way encryption. Normally, the user's encrypted password is stored in a file for future password comparison.

### INPUTS

```

buffer      - a pointer to a buffer at least 12 bytes in length.
password    - a pointer to an unencrypted password string.
username    - a pointer to the user's name.

```

### RESULT

```

newpass     - a pointer to the passed buffer if successful, NULL
              upon failure. The encrypted password placed in the
              buffer will be be eleven (11) characters in length
              and will be NULL-terminated.

```

### EXAMPLE

```

UBYTE *pw, *getpassword() ;
UBYTE *user = "alf"
UBYTE *newpass ;
UBYTE buffer[16] ;          /* size >= 12 */

pw = getpassword() ;        /* your own function */

if((newpass = ACrypt(buffer, pw, user)) != NULL)
{

```

```

    printf("pw = %s\n", newpass) ; /* newpass = &buffer[0] */
}
else
{
    printf("ACrypt failed\n") ;
}

```

#### NOTES

This function first appeared in later V39 versions of amiga.lib, but works under V37 and up.

## 1.3 amiga.lib/AddTOF

#### NAME

AddTOF - add a task to the VBlank 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 an initialized Isrvstr structure  
 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>

## 1.4 amiga.lib/afp

#### NAME

afp - Convert ASCII string variable into fast floating point

#### SYNOPSIS

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 forty-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

## 1.5 amiga.lib/ArgArrayDone

**NAME**

ArgArrayDone -- release the memory allocated by a previous call to ArgArrayInit(). (V36)

**SYNOPSIS**

```
ArgArrayDone();
```

```
VOID ArgArrayDone(VOID);
```

**FUNCTION**

This function frees memory and does cleanup required after a call to ArgArrayInit(). Don't call this until you are done using the ToolTypes argument strings.

**SEE ALSO**

ArgArrayInit()



## 1.6 amiga.lib/ArgArrayInit

### NAME

ArgArrayInit -- allocate and initialize a tooltype array. (V36)

### SYNOPSIS

```
ttypes = ArgArrayInit(argc,argv);
```

```
UBYTE **ArgArrayInit(LONG,UBYTE **);
```

### FUNCTION

This function returns a null-terminated array of strings suitable for sending to icon.library/FindToolType(). This array will be the ToolTypes array of the program's icon, if it was started from Workbench. It will just be 'argv' if the program was started from a shell.

Pass ArgArrayInit() your startup arguments received by main().

ArgArrayInit() requires that icon.library be open (even if the caller was started from a shell, so that the function FindToolType() can be used) and may call GetDiskObject(), so clean up is necessary when the strings are no longer needed. The function ArgArrayDone() does just that.

### INPUTS

argc - the number of arguments in argv, 0 when started from Workbench  
 argv - an array of pointers to the program's arguments, or the Workbench startup message when started from WB.

### RESULTS

ttypes - the initialized argument array or NULL if it could not be allocated

### EXAMPLE

Use of these routines facilitates the use of ToolTypes or command-line arguments to control end-user parameters in Commodities applications. For example, a filter used to trap a keystroke for popping up a window might be created by something like this:

```
char    *ttypes  = ArgArrayInit(argc, argv);
CxObj    *filter = UserFilter(ttypes, "POPWINDOW", "alt f1");

    ... with ...

    CxObj *UserFilter(char **tt, char *action_name,
        char *default_descr)
{
    char *desc;

    desc = FindToolType(tt,action_name);

    return(CxFilter((ULONG) (desc? desc: default_descr)));
}
```

In this way the user can assign "alt f2" to the action by

entering a tooltype in the program's icon of the form:

```
POPWINDOW=alt f2
```

or by starting the program from the CLI like so:

```
myprogram "POPWINDOW=alt f2"
```

#### NOTE

Your program must open `icon.library` and set up `IconBase` before calling this routine. In addition `IconBase` must remain valid until after `ArgArrayDone()` has been called!

#### SEE ALSO

`ArgArrayDone()`, `ArgString()`, `ArgInt()`, `icon.library/FindToolType()`

## 1.7 amiga.lib/ArgInt

#### NAME

`ArgInt` -- return an integer value from a ToolTypes array. (V36)

#### SYNOPSIS

```
value = ArgInt(tt,entry,defaultval)
```

```
LONG ArgInt(UBYTE **,STRPTR, LONG);
```

#### FUNCTION

This function looks in the ToolTypes array 'tt' returned by `ArgArrayInit()` for 'entry' and returns the value associated with it. 'tt' is in standard ToolTypes format such as:

```
ENTRY=Value
```

The Value string is passed to `atoi()` and the result is returned by this function.

If 'entry' is not found, the integer 'defaultval' is returned.

#### INPUTS

tt - a ToolTypes array as returned by `ArgArrayInit()`

entry - the entry in the ToolTypes array to search for

defaultval - the value to return in case 'entry' is not found within the ToolTypes array

#### RESULTS

value - the value associated with 'entry', or defaultval if 'entry' is not in the ToolTypes array

#### NOTES

This function requires that `dos.library` V36 or higher be opened.

#### SEE ALSO

`ArgArrayInit()`

---

## 1.8 amiga.lib/ArgString

### NAME

ArgString -- return a string pointer from a ToolTypes array. (V36)

### SYNOPSIS

```
string = ArgString(tt,entry,defaultstring)
```

```
STRPTR ArgString(UBYTE **,STRPTR,STRPTR);
```

### FUNCTION

This function looks in the ToolTypes array 'tt' returned by ArgArrayInit() for 'entry' and returns the value associated with it. 'tt' is in standard ToolTypes format such as:

```
ENTRY=Value
```

This function returns a pointer to the Value string.

If 'entry' is not found, 'defaultstring' is returned.

### INPUTS

tt - a ToolTypes array as returned by ArgArrayInit()

entry - the entry in the ToolTypes array to search for

defaultstring - the value to return in case 'entry' is not found within the ToolTypes array

### RESULTS

value - the value associated with 'entry', or defaultstring if 'entry' is not in the ToolTypes array

### SEE ALSO

ArgArrayInit()

## 1.9 amiga.lib/arnd

### NAME

arnd - ASCII round of the provided floating point string

### SYNOPSIS

```
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

## 1.10 amiga.lib/BeginIO

NAME  
BeginIO -- initiate asynchronous device I/O

SYNOPSIS  
BeginIO(ioReq)

```
VOID BeginIO(struct IORequest *);
```

FUNCTION  
This function takes an IORequest, and passes it directly to the "BeginIO" vector of the proper device. This is equivalent to SendIO(), except that io\_Flags is not cleared. A good understanding of Exec device I/O is required to properly use this function.

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

INPUTS  
ioReq - an initialized and opened IORequest structure with the io\_Flags field set to a reasonable value (set to 0 if you do not require io\_Flags).

SEE ALSO  
exec.library/DoIO(), exec.library/SendIO(), exec.library/WaitIO()

## 1.11 amiga.lib/CallHook

NAME  
CallHook -- Invoke a hook given a message on the stack.

SYNOPSIS  
result = CallHook( hookPtr, obj, ... )

```
ULONG CallHook( struct Hook *, Object *, ... );
```

FUNCTION  
Like CallHookA(), CallHook() invoke a hook on the supplied hook-specific data (an "object") and a parameter packet ("message"). However, CallHook() allows you to build the message on your stack.

INPUTS  
hookPtr - A system-standard hook  
obj - hook-specific data object

---

... - The hook-specific message you wish to send. The hook is expecting a pointer to the message, so a pointer into your stack will be sent.

#### RESULT

result - a hook-specific result.

#### NOTES

This function first appeared in the V37 release of amiga.lib. However, it does not depend on any particular version of the OS, and works fine even in V34.

#### EXAMPLE

If your hook's message was

```
struct myMessage
{
    ULONG mm_FirstGuy;
    ULONG mm_SecondGuy;
    ULONG mm_ThirdGuy;
};
```

You could write:

```
result = CallHook( hook, obj, firstguy, secondguy, thirdguy );
```

as a shorthand for:

```
struct myMessage msg;

msg.mm_FirstGuy = firstguy;
msg.mm_SecondGuy = secondguy;
msg.mm_ThirdGuy = thirdguy;

result = CallHookA( hook, obj, &msg );
```

#### SEE ALSO

CallHookA(), utility.library/CallHookPkt(), <utility/hooks.h>

## 1.12 amiga.lib/CallHookA

#### NAME

CallHookA -- Invoke a hook given a pointer to a message.

#### SYNOPSIS

```
result = CallHookA( hookPtr, obj, message )
```

```
ULONG CallHook( struct Hook *, Object *, APTR );
```

#### FUNCTION

Invoke a hook on the supplied hook-specific data (an "object") and a parameter packet ("message"). This function is equivalent to utility.library/CallHookPkt().

#### INPUTS

---

hookPtr - A system-standard hook  
obj - hook-specific data object  
message - The hook-specific message you wish to send

RESULT  
result - a hook-specific result.

NOTES  
This function first appeared in the V37 release of amiga.lib.  
However, it does not depend on any particular version of the OS,  
and works fine even in V34.

SEE ALSO  
CallHook(), utility.library/CallHookPkt(), <utility/hooks.h>

## 1.13 amiga.lib/CheckRexxMsg

NAME  
CheckRexxMsg - Check if a RexxMsg is from ARexx

SYNOPSIS  
result = CheckRexxMsg(message)  
D0                                   A0

BOOL CheckRexxMsg(struct RexxMsg \*);

FUNCTION  
This function checks to make sure that the message is from ARexx directly. It is required when using the Rexx Variable Interface routines (RVI) that the message be from ARexx.

While this function is new in the V37 amiga.lib, it is safe to call it in all versions of the operating system. It is also PURE code, thus usable in resident/pure executables.

NOTE  
This is a stub in amiga.lib. It is only available via amiga.lib. The stub has two labels. One, \_CheckRexxMsg, takes the arguments from the stack. The other, CheckRexxMsg, takes the arguments in registers.

EXAMPLE  
if (CheckRexxMsg(rxmsg))  
{  
    /\* Message is one from ARexx \*/  
}

INPUTS  
message    A pointer to the RexxMsg in question

RESULTS  
result     A boolean - TRUE if message is from ARexx.

SEE ALSO

---

GetRexxVar(), SetRexxVar()

## 1.14 amiga.lib/CoerceMethod

### NAME

CoerceMethod -- Perform method on coerced object.

### SYNOPSIS

```
result = CoerceMethod( cl, obj, MethodID, ... )
```

```
ULONG CoerceMethod( struct IClass *, Object *, ULONG, ... );
```

### FUNCTION

Boopsi support function that invokes the supplied message on the specified object, as though it were the specified class. Equivalent to CoerceMethodA(), but allows you to build the message on the stack.

### INPUTS

cl - pointer to boopsi class to receive the message  
obj - pointer to boopsi object  
... - method-specific message built on the stack

### RESULT

result - class and message-specific result.

### NOTES

This function first appears in the V37 release of amiga.lib. While it intrinsically does not require any particular release of the system software to operate, it is designed to work with the boopsi subsystem of Intuition, which was only introduced in V36.

### SEE ALSO

CoerceMethodA(), DoMethodA(), DoSuperMethodA(), <intuition/classusr.h>  
ROM Kernel Manual boopsi section

## 1.15 amiga.lib/CoerceMethodA

### NAME

CoerceMethodA -- Perform method on coerced object.

### SYNOPSIS

```
result = CoerceMethodA( cl, obj, msg )
```

```
ULONG CoerceMethodA( struct IClass *, Object *, Msg );
```

### FUNCTION

Boopsi support function that invokes the supplied message on the specified object, as though it were the specified class.

---

**INPUTS**

cl - pointer to boopsi class to receive the message  
 obj - pointer to boopsi object  
 msg - pointer to method-specific message to send

**RESULT**

result - class and message-specific result.

**NOTES**

This function first appears in the V37 release of amiga.lib. While it intrinsically does not require any particular release of the system software to operate, it is designed to work with the boopsi subsystem of Intuition, which was only introduced in V36. Some early example code may refer to this function as CM().

**SEE ALSO**

CoerceMethod(), DoMethodA(), DoSuperMethodA(), <intuition/classusr.h>  
 ROM Kernel Manual boopsi section

## 1.16 amiga.lib/CreateExtIO

**NAME**

CreateExtIO -- create an IORequest structure

**SYNOPSIS**

```
ioReq = CreateExtIO(port,ioSize);
```

```
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**

port - an already initialized message port to be used for this IO request's reply port. If this is NULL this function fails.  
 ioSize - the size of the IO request to be created.

**RESULT**

ioReq - a new IO Request block, or NULL if there was not enough memory

**EXAMPLE**

```
if (ioReq = CreateExtIO(CreatePort(NULL,0),sizeof(struct IOExtTD)))
```

**SEE ALSO**

DeleteExtIO(), CreatePort(), exec.library/CreateMsgPort()

## 1.17 amiga.lib/CreatePort



## NAME

CreatePort - Allocate and initialize a new message port

## SYNOPSIS

```
port = CreatePort(name,pri)
```

```
struct MsgPort *CreatePort (STRPTR, LONG);
```

## FUNCTION

Allocates and initializes a new message port. The message list of the new port will be prepared for use (via NewList). A signal bit will be allocated, and the port will be set to signal your task when a message arrives (PA\_SIGNAL).

You *must* use DeletePort() to delete ports created with CreatePort()!

## INPUTS

name - public name of the port, or NULL if the port is not named.  
The name string is not copied. Most ports do not need names, see notes below on this.

pri - Priority used for insertion into the public port list, normally 0.

## RESULT

port - a new MsgPort structure ready for use, or NULL if the port could not be created due to not enough memory or no available signal bit.

## NOTE

In most cases, ports should not be named. Named ports are used for rendez-vous between tasks. Everytime a named port needs to be located, the list of all named ports must be traversed. The more named ports there are, the longer this list traversal takes. Thus, unless you really need to, do not name your ports, which will keep them off of the named port list and improve system performance.

## BUGS

With versions of amiga.lib prior to V37.14, this function would not fail even though it couldn't allocate a signal bit. The port would be returned with no signal allocated.

## SEE ALSO

DeletePort(), exec.library/FindPort(), <exec/ports.h>, exec.library/CreateMsgPort()

## 1.18 amiga.lib/CreateStdIO

## NAME

CreateStdIO -- create an IOStdReq structure

## SYNOPSIS

```
ioReq = CreateStdIO(port);
```

```
struct IOStdReq *CreateStdIO(struct MsgPort *)
```

#### FUNCTION

Allocates memory for and initializes a new IOStdReq structure.

#### INPUTS

port - an already initialized message port to be used for this IO request's reply port. If this is NULL this function fails.

#### RESULT

ioReq - a new IOStdReq structure, or NULL if there was not enough memory

#### SEE ALSO

DeleteStdIO(), CreateExtIO(), exec.library/CreateIORequest()

## 1.19 amiga.lib/CreateTask

#### NAME

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

#### SYNOPSIS

```
task = CreateTask(name,pri,initPC,stackSize)
```

```
struct Task *CreateTask(STRPTR, LONG, funcEntry, ULONG);
```

#### FUNCTION

This function simplifies program creation of sub-tasks 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.library/CreateProc() or the dos.library/CreateNewProc() functions 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 sub-task 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 name string  
pri - an Exec task priority between -128 and 127, normally 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.

RESULT  
task - a pointer to the newly created task, or NULL if there was not  
enough memory.

BUGS  
Under exec.library V37 or beyond, the AddTask() function used  
internally by CreateTask() can fail whereas it couldn't fail in  
previous versions of Exec. Prior to amiga.lib V37.14, this function  
did not check for failure of AddTask() and thus might return a  
pointer to a task structure even though the task was not actually  
added to the system.

SEE ALSO  
DeleteTask(), exec/FindTask()

## 1.20 amiga.lib/CxCustom

NAME  
CxCustom -- create a custom commodity object. (V36)

SYNOPSIS  
customObj = CxCustom(action,id);

CxObj \*CxCustom(LONG(\*)(),LONG);

FUNCTION  
This function creates a custom commodity object. The action  
of this object on receiving a commodity message is to call a  
function of the application programmer's choice.

The function provided ('action') will be passed a pointer to  
the actual commodities message (in commodities private data  
space), and will actually execute as part of the input handler  
system task. Among other things, the value of 'id' can be  
recovered from the message by using the function CxMsgID().

The purpose of this function is two-fold. First, it allows  
programmers to create Commodities Exchange objects with  
functionality that was not imagined or chosen for inclusion  
by the designers. Secondly, this is the only way to act

synchronously with Commodities.

This function is a C-language macro for `CreateCxObj()`, defined in `<libraries/commodities.h>`.

#### INPUTS

action - a function to call whenever a message reaches the object  
id - a message id to assign to the object

#### RESULTS

customObj - a pointer to the new custom object, or NULL if it could not be created.

#### SEE ALSO

`commodities.library/CreateCxObj()`, `commodities.library/CxMsgID()`

## 1.21 amiga.lib/CxDebug

#### NAME

`CxDebug` -- create a commodity debug object. (V36)

#### SYNOPSIS

```
debugObj = CxDebug(id);
```

```
CxObj *CxDebug(LONG);
```

#### FUNCTION

This function creates a Commodities debug object. The action of this object on receiving a Commodities message is to print out information about the Commodities message through the serial port (using the `kprintf()` routine). The value of 'id' will also be displayed.

Note that this is a synchronous occurrence (the printing is done by the input device task). If screen or file output is desired, using a sender object instead of a debug object is necessary, since such output is best done by your application process.

This function is a C-language macro for `CreateCxObj()`, defined in `<libraries/commodities.h>`.

#### INPUTS

id - the id to assign to the debug object, this value is output whenever the debug object sends data to the serial port.

#### RESULTS

debugObj - a pointer to the debug object, or NULL if it could not be created.

#### SEE ALSO

`commodities.library/CreateCxObj()`, `CxSender()`, `debug.lib/kprintf()`

## 1.22 amiga.lib/CxFilter

---

## NAME

CxFilter -- create a commodity filter object. (V36)

## SYNOPSIS

```
filterObj = CxFilter(description);
```

```
CxObj *CxFilter(STRPTR)
```

## FUNCTION

Creates an input event filter object that matches the 'description' string. If 'description' is NULL, the filter will not match any messages.

A filter may be modified by the functions SetFilter(), using a description string, and SetFilterIX(), which takes a binary Input Expression as a parameter.

This function is a C-language macro for CreateCxObj(), defined in <libraries/commodities.h>.

## INPUTS

description - the description string in the same format as strings expected by commodities.library/SetFilter()

## RESULTS

filterObj - a pointer to the filter object, or NULL if there was not enough memory. If there is a problem in the description string, the internal error code of the filter object will be set to so indicate. This error code may be interrogated using the function CxObjError().

## SEE ALSO

commodities.library/CreateCxObj(), commodities.library/SetFilter(), commodities.library/SetFilterIX(), commodities.library/CxObjError()

## 1.23 amiga.lib/CxSender

## NAME

CxSender -- create a commodity sender object. (V36)

## SYNOPSIS

```
senderObj = CxSender(port,id)
```

```
CxObj *CxSender(struct MsgPort *,LONG);
```

## FUNCTION

This function creates a Commodities sender object. The action of this object on receiving a Commodities message is to copy the Commodities message into a standard Exec Message, to put the value 'id' in the message as well, and to send the message off to the message port 'port'.

The value 'id' is used so that an application can monitor messages from several senders at a single port. It can be retrieved

---

from the Exec message by using the function `CxMsgID()`. The value can be a simple integer ID, or a pointer to some application data structure.

Note that Exec messages sent by sender objects arrive asynchronously at the destination port. Do not assume anything about the status of the Commodities message which was copied into the Exec message you received.

All Exec messages sent to your ports must be replied. Messages may be replied after the sender object has been deleted.

This function is a C-language macro for `CreateCxObj()`, defined in `<libraries/commodities.h>`.

#### INPUTS

port - the port for the sender to send messages to  
id - the id of the messages sent by the sender

#### RESULTS

senderObj - a pointer to the sender object, or NULL if it could not be created.

#### SEE ALSO

`commodities.library/CreateCxObj()`, `commodities.library/CxMsgID()`, `exec.library/PutMsg()`, `exec.library/ReplyMsg()`

## 1.24 amiga.lib/CxSignal

#### NAME

`CxSignal` -- create a commodity signaller object. (V36)

#### SYNOPSIS

```
signalerObj = CxSignal(task,signal);
```

```
CxObj *CxSignal(struct Task *,LONG);
```

#### FUNCTION

This function creates a Commodities signal object. The action of this object on receiving a Commodities message is to send the 'signal' to the 'task'. The caller is responsible for allocating the signal and determining the proper task ID.

Note that 'signal' is the signal value as returned by `AllocSignal()`, not the mask made from that value.

This function is a C-language macro for `CreateCxObj()`, defined in `<libraries/commodities.h>`.

#### INPUTS

task - the task for the signaller to signal  
signal - the signal bit number for the signaller to send

#### RESULTS

signallerObj - a pointer to the signaller object, or NULL if it could

---

not be created.

SEE ALSO

commodities.library/CreateCxObj(), exec.library/FindTask()  
exec.library/Signal(), exec.library/AllocSignal(),

## 1.25 amiga.lib/CxTranslate

NAME

CxTranslate -- create a commodity translator object. (V36)

SYNOPSIS

translatorObj = CxTranslate(ie);

CxObj \*CxTranslate(struct InputEvent \*);

FUNCTION

This function creates a Commodities 'translator' object. The action of this object on receiving a Commodities message is to replace that message in the commodities network with a chain of Commodities input messages.

There is one new Commodities input message generated for each input event in the linked list starting at 'ie' (and NULL terminated). The routing information of the new input messages is copied from the input message they replace.

The linked list of input events associated with a translator object can be changed using the SetTranslate() function.

If 'ie' is NULL, the null translation occurs: that is, the original commodities input message is disposed, and no others are created to take its place.

This function is a C-language macro for CreateCxObj(), defined in <libraries/commodities.h>.

INPUTS

ie - the input event list used as replacement by the translator

RESULTS

translatorObj - a pointer to the translator object, or NULL if it could not be created.

SEE ALSO

commodities.library/CreateCxObj(), commodities.library/SetTranslate()

## 1.26 amiga.lib/dbf

NAME

dbf - convert FFP dual-binary number to FFP format

---

## SYNOPSIS

```
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

## 1.27 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().

## INPUTS

ioReq - the IORequest block to be freed, or NULL.

## SEE ALSO

CreateExtIO()

## 1.28 amiga.lib/DeletePort

## NAME

DeletePort - free a message port created by CreatePort()

## SYNOPSIS

```
DeletePort(port)
```

```
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 before this function is called.

**INPUTS**

port - message port to delete

**SEE ALSO**

`CreatePort()`

## 1.29 amiga.lib/DeleteStdIO

**NAME**

`DeleteStdIO` - return memory allocated for `IOStdReq`

**SYNOPSIS**

```
DeleteStdIO(ioReq);
```

```
VOID DeleteStdIO(struct IOStdReq *);
```

**FUNCTION**

Frees up an `IOStdReq` as allocated by `CreateStdIO()`.

**INPUTS**

ioReq - the `IORequest` block to be freed, or `NULL`.

**SEE ALSO**

`CreateStdIO()`, `DeleteExtIO()`, `exec.library/CreateIORequest()`

## 1.30 amiga.lib/DeleteTask

**NAME**

`DeleteTask` -- delete a task created with `CreateTask()`

**SYNOPSIS**

```
DeleteTask(task)
```

```
VOID DeleteTask(struct Task *);
```

**FUNCTION**

This function simply calls `exec.library/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 to execute a `Wait(0L)`. Another option is to have the task call `DeleteTask()/RemTask()` on

---

itself.

#### INPUTS

task - task to remove from the system

#### NOTE

This function simply calls `exec.library/RemTask()`, so you can call `RemTask()` directly instead of calling this function.

#### SEE ALSO

`CreateTask()`, `exec.library/RemTask()`

## 1.31 amiga.lib/DoMethod

#### NAME

`DoMethod` -- Perform method on object.

#### SYNOPSIS

`result = DoMethod( obj, MethodID, ... )`

`ULONG DoMethod( Object *, ULONG, ... );`

#### FUNCTION

Boopsi support function that invokes the supplied message on the specified object. The message is invoked on the object's true class. Equivalent to `DoMethodA()`, but allows you to build the message on the stack.

#### INPUTS

obj - pointer to boopsi object

MethodID - which method to send (see `<intuition/classusr.h>`)

... - method-specific message built on the stack

#### RESULT

result - specific to the message and the object's class.

#### NOTES

This function first appears in the V37 release of `amiga.lib`. While it intrinsically does not require any particular release of the system software to operate, it is designed to work with the boopsi subsystem of Intuition, which was only introduced in V36.

#### SEE ALSO

`DoMethodA()`, `CoerceMethodA()`, `DoSuperMethodA()`, `<intuition/classusr.h>`  
ROM Kernel Manual boopsi section

## 1.32 amiga.lib/DoMethodA

#### NAME

`DoMethodA` -- Perform method on object.

---

## SYNOPSIS

```
result = DoMethodA( obj, msg )
```

```
ULONG DoMethodA( Object *, Msg );
```

## FUNCTION

Boopsi support function that invokes the supplied message on the specified object. The message is invoked on the object's true class.

## INPUTS

obj - pointer to boopsi object

msg - pointer to method-specific message to send

## RESULT

result - specific to the message and the object's class.

## NOTES

This function first appears in the V37 release of amiga.lib. While it intrinsically does not require any particular release of the system software to operate, it is designed to work with the boopsi subsystem of Intuition, which was only introduced in V36.

Some early example code may refer to this function as DM().

## SEE ALSO

DoMethod(), CoerceMethodA(), DoSuperMethodA(), <intuition/classusr.h>  
ROM Kernel Manual boopsi section

## 1.33 amiga.lib/DoSuperMethod

## NAME

DoSuperMethod -- Perform method on object coerced to superclass.

## SYNOPSIS

```
result = DoSuperMethod( cl, obj, MethodID, ... )
```

```
ULONG DoSuperMethod( struct IClass *, Object *, ULONG, ... );
```

## FUNCTION

Boopsi support function that invokes the supplied message on the specified object, as though it were the superclass of the specified class. Equivalent to DoSuperMethodA(), but allows you to build the message on the stack.

## INPUTS

cl - pointer to boopsi class whose superclass is to receive the message

obj - pointer to boopsi object

... - method-specific message built on the stack

## RESULT

result - class and message-specific result.

## NOTES

This function first appears in the V37 release of `amiga.lib`. While it intrinsically does not require any particular release of the system software to operate, it is designed to work with the boopsi subsystem of Intuition, which was only introduced in V36.

SEE ALSO

`CoerceMethodA()`, `DoMethodA()`, `DoSuperMethodA()`, `<intuition/classusr.h>`  
ROM Kernel Manual boopsi section

## 1.34 `amiga.lib/DoSuperMethodA`

NAME

`DoSuperMethodA` -- Perform method on object coerced to superclass.

SYNOPSIS

```
result = DoSuperMethodA( cl, obj, msg )
```

```
ULONG DoSuperMethodA( struct IClass *, Object *, Msg );
```

FUNCTION

Boopsi support function that invokes the supplied message on the specified object, as though it were the superclass of the specified class.

INPUTS

`cl` - pointer to boopsi class whose superclass is to receive the message  
`obj` - pointer to boopsi object  
`msg` - pointer to method-specific message to send

RESULT

`result` - class and message-specific result.

NOTES

This function first appears in the V37 release of `amiga.lib`. While it intrinsically does not require any particular release of the system software to operate, it is designed to work with the boopsi subsystem of Intuition, which was only introduced in V36.

Some early example code may refer to this function as `DSM()`.

SEE ALSO

`CoerceMethodA()`, `DoMethodA()`, `DoSuperMethod()`, `<intuition/classusr.h>`  
ROM Kernel Manual boopsi section

## 1.35 `amiga.lib/FastRand`

NAME

`FastRand` - quickly generate a somewhat random integer

SYNOPSIS

---

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

```
ULONG FastRand(ULONG);
```

#### FUNCTION

Seed value is taken from stack, shifted left one position, exclusive-or'ed with hex value \$1D872B41 and returned.

#### INPUTS

seed - a 32-bit integer

#### RESULT

number - new random seed, a 32-bit value

#### SEE ALSO

RangeRand()

## 1.36 amiga.lib/fpa

#### NAME

fpa - convert fast floating point into ASCII string equivalent

#### SYNOPSIS

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

#### FUNCTION

Accepts an FFP number and the address of the ASCII string where it's converted 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

## 1.37 amiga.lib/FreeEvents

#### NAME

FreeEvents -- free a chain of input events allocated by  
InvertString(). (V36)

#### SYNOPSIS

```
FreeEvents(events)
```

---

```
VOID FreeIEvents(struct InputEvent *);
```

#### FUNCTION

This function frees a linked list of input events as obtained from InvertString().

#### INPUTS

events - the list of input events to free, may be NULL.

#### SEE ALSO

InvertString()

## 1.38 amiga.lib/GetRexxVar

#### NAME

GetRexxVar - Gets the value of a variable from a running ARexx program

#### SYNOPSIS

```
error = GetRexxVar(message, varname, bufpointer)
D0,A1          A0      A1      (C-only)
```

```
LONG GetRexxVar(struct RexxMsg *,char *,char **);
```

#### FUNCTION

This function will attempt to extract the value of the symbol varname from the ARexx script that sent the message. When called from C, a pointer to the extracted value will be placed in the pointer pointed to by bufpointer. (\*bufpointer will be the pointer to the value)

When called from assembly, the pointer will be returned in A1.

The value string returned *MUST* *NOT* be modified.

While this function is new in the V37 amiga.lib, it is safe to call it in all versions of the operating system. It is also PURE code, thus usable in resident/pure executables.

#### NOTE

This is a stub in amiga.lib. It is only available via amiga.lib. The stub has two labels. One, \_GetRexxVar, takes the arguments from the stack. The other, GetRexxVar, takes the arguments in registers.

This routine does a CheckRexxMsg() on the message.

#### EXAMPLE

```
char *value;
```

```
/* Message is one from ARexx */
if (!GetRexxVar(rxmsg, "TheVar", &value))
{
    /* The value was gotten and now is pointed to by value */
    printf("Value of TheVar is %s\n", value);
}
```

```
}
```

#### INPUTS

message    A message gotten from an ARexx script  
varname    The name of the variable to extract  
bufpointer    (For C only) A pointer to a string pointer.

#### RESULTS

error    0 for success, otherwise an error code.  
          (Other codes may exist, these are documented)  
3    == Insufficient Storage  
9    == String too long  
10 == invalid message

A1    (Assembly only)    Pointer to the string.

#### SEE ALSO

SetRexxVar(), CheckRexxMsg()

## 1.39 amiga.lib/HookEntry

#### NAME

HookEntry -- Assembler to HLL conversion stub for hook entry.

#### SYNOPSIS

```
result = HookEntry( struct Hook *, Object *, APTR )
D0                            A0                            A2                            A1
```

#### FUNCTION

By definition, a standard hook entry-point must receive the hook in A0, the object in A2, and the message in A1. If your hook entry-point is written in a high-level language and is expecting its parameters on the stack, then HookEntry() will put the three parameters on the stack and invoke the function stored in the hook h\_SubEntry field.

This function is only useful to hook implementers, and is never called from C.

#### INPUTS

hook - pointer to hook being invoked  
object - pointer to hook-specific data  
msg - pointer to hook-specific message

#### RESULT

result - a hook-specific result.

#### NOTES

This function first appeared in the V37 release of amiga.lib. However, it does not depend on any particular version of the OS, and works fine even in V34.

#### EXAMPLE

If your hook dispatcher is this:

```
dispatch( struct Hook *hookPtr, Object *obj, APTR msg )
{
    ...
}
```

Then when you initialize your hook, you would say:

```
myhook.h_Entry = HookEntry; /* amiga.lib stub */
myhook.h_SubEntry = dispatch; /* HLL entry */
```

SEE ALSO  
 CallHook(), CallHookA(), <utility/hooks.h>

## 1.40 amiga.lib/HotKey

NAME  
 HotKey -- create a commodity triad. (V36)

SYNOPSIS  
 filterObj = Hotkey(description,port,id);

CxObj \*HotKey(STRPTR,struct MsgPort \*,LONG);

FUNCTION  
 This function creates a triad of commodity objects to accomplish a high-level function.

The three objects are a filter, which is created to match by the call CxFilter(description), a sender created by the call CxSender(port,id), and a translator which is created by CxTranslate(NULL), so that it swallows any commodity input event messages that are passed down by the filter.

This is the simple way to get a message sent to your program when the user performs a particular input action.

It is strongly recommended that the ToolTypes environment be used to allow the user to specify the input descriptions for your application's hotkeys.

INPUTS  
 description - the description string to use for the filter in the same format as accepted by commodities.library/SetFilter()  
 port - port for the sender to send messages to.  
 id - id of the messages sent by the sender

RESULTS  
 filterObj - a pointer to a filter object, or NULL if it could not be created.

SEE ALSO  
 CxFilter(), CxSender(), CxTranslate(),  
 commodities.library/CxObjError(), commodities.library/SetFilter()



## 1.41 amiga.lib/InvertString

### NAME

InvertString -- produce input events that would generate the given string. (V36)

### SYNOPSIS

```
events = InvertString(str, km)
```

```
struct InputEvent *InvertString(STRPTR, struct KeyMap *);
```

### FUNCTION

This function returns a linked list of input events which would translate into the string using the supplied keymap (or the system default keymap if 'km' is NULL).

'str' is null-terminated and may contain:

- ANSI character codes
- backslash escaped characters:
  - \n - CR
  - \r - CR
  - \t - TAB
  - \0 - illegal, do not use!
  - \ - backslash
- a text description of an input event as used by ParseIX(), enclosed in angle brackets.

An example is:

```
abc<alt f1>\nhi there.
```

### INPUTS

str - null-terminated string to convert to input events

km - keymap to use for the conversion, or NULL to use the default keymap.

### RESULTS

events - a chain of input events, or NULL if there was a problem. The most likely cause of failure is an illegal description enclosed in angled brackets.

This chain should eventually be freed using FreeIEvents().

### SEE ALSO

commodities.library/ParseIX(), FreeIEvents()

## 1.42 amiga.lib/NewList

### NAME

NewList -- prepare a list structure for use

### SYNOPSIS

```
NewList(list)
```

```
VOID NewList(struct List *);
```

---

```
VOID NewList(struct MinList *);
```

#### FUNCTION

Perform the magic needed to prepare a List header structure for use; the list will be empty and ready to use. (If the list is the full featured type, you may need to initialize lh\_Type afterwards)

Assembly programmers may want to use the NEWLIST macro instead.

#### INPUTS

list - pointer to a List or MinList.

#### SEE ALSO

<exec/lists.h>

## 1.43 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 C-language-like NULL-terminated format string, with the following supported % options:

```
%[flags][width][.limit][length]type
```

\$	- must follow the arg_pos value, if specified
flags	- only one allowed. '-' specifies left justification.
width	- field width. If the first character is a '0', the field is padded with leading 0s.
.	- must precede the field width value, if specified
limit	- maximum number of characters to output from a string. (only valid for %s or %b).
length	- size of input data defaults to word (16-bit) for types c, d, u and x, 'l' changes this to long (32-bit).
type	- supported types are:
	b - BSTR, data is 32-bit BPTR to byte count followed by a byte string. A NULL BPTR is treated as an empty string. (V36)
	d - signed decimal
	u - unsigned decimal
	x - hexadecimal with hex digits in uppercase
	X - hexadecimal with hex digits in lowercase
	s - string, a 32-bit pointer to a NULL-terminated byte string. A NULL pointer is treated as an empty string.
	c - character

value(s) - numeric variables or addresses of null-terminated strings to be added to the format information.

#### 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 `dos.library/Output()` call returns). This code would set it up:

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

#### BUGS

This function will crash if the resulting stream after parameter substitution is longer than 140 bytes.

## 1.44 amiga.lib/RangeRand

#### NAME

RangeRand - generate a random number within a specific integer range

#### SYNOPSIS

```
number = RangeRand(maxValue);
```

```
UWORD RangeRand(UWORD);
```

#### FUNCTION

RangeRand() accepts a value from 0 to 65535, and returns a value within that range.

maxValue 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 V33 that contains the global seed value passed from call to call and thus can be changed in a program by declaring:

```
extern ULONG RangeSeed;
```

#### INPUTS

maxValue - the returned random number will be in the range [0..maxValue-1]

#### RESULT

number - pseudo random number in the range of [0..maxValue-1].

#### SEE ALSO

FastRand()

## 1.45 amiga.lib/RemTOF

#### NAME

RemTOF - remove a task from the VBlank interrupt server chain.

---

## SYNOPSIS

```
RemTOF(i);
```

```
VOID RemTOF(struct Isrvstr *);
```

## FUNCTION

Removes a task from the vertical-blanking interval interrupt server chain.

## INPUTS

i - pointer to an Isrvstr structure

## SEE ALSO

AddTOF(), <graphics/graphint.h>

## 1.46 amiga.lib/SetRexxVar

## NAME

SetRexxVar - Sets the value of a variable of a running ARexx program

## SYNOPSIS

```
error = SetRexxVar(message, varname, value, length)
```

```
D0                A0      A1      D0      D1
```

```
LONG SetRexxVar(struct RexxMsg *, char *, char *, ULONG);
```

## FUNCTION

This function will attempt to the the value of the symbol varname in the ARexx script that sent the message.

While this function is new in the V37 amiga.lib, it is safe to call it in all versions of the operating system. It is also PURE code, thus usable in resident/pure executables.

## NOTE

This is a stub in amiga.lib. It is only available via amiga.lib. The stub has two labels. One, \_SetRexxVar, takes the arguments from the stack. The other, SetRexxVar, takes the arguments in registers.

This routine does a CheckRexxMsg() on the message.

## EXAMPLE

```
char *value;
```

```
/* Message is one from ARexx */
```

```
if (!SetRexxVar(rxmsg, "TheVar", "25 Dollars", 10))
```

```
{
```

```
    /* The value of TheVar will now be "25 Dollars" */
```

```
}
```

## INPUTS

message A message gotten from an ARexx script

varname    The name of the variable to set  
 value     A string that will be the new value of the variable  
 length    The length of the value string

#### RESULTS

error    0 for success, otherwise an error code.  
          (Other codes may exist, these are documented)  
       3 == Insufficient Storage  
       9 == String too long  
      10 == invalid message

#### SEE ALSO

SetRexxVar(), CheckRexxMsg()

## 1.47 amiga.lib/SetSuperAttrs

#### NAME

SetSuperAttrs -- Invoke OM\_SET method on superclass with varargs.

#### SYNOPSIS

```
result = SetSuperAttrs( cl, obj, tag, ... )
```

```
ULONG SetSuperAttrs( struct IClass *, Object *, ULONG, ... );
```

#### FUNCTION

Boopsi support function which invokes the OM\_SET method on the superclass of the supplied class for the supplied object. Allows the ops\_AttrList to be supplied on the stack (i.e. in a varargs way). The equivalent non-varargs function would simply be

```
DoSuperMethod( cl, obj, OM_SET, taglist, NULL );
```

#### INPUTS

cl - pointer to boopsi class whose superclass is to receive the OM\_SET message  
 obj - pointer to boopsi object  
 tag - list of tag-attribute pairs, ending in TAG\_DONE

#### RESULT

result - class and message-specific result.

#### NOTES

This function first appears in the V37 release of amiga.lib. While it intrinsically does not require any particular release of the system software to operate, it is designed to work with the boopsi subsystem of Intuition, which was only introduced in V36.

#### SEE ALSO

CoerceMethodA(), DoMethodA(), DoSuperMethodA(), <intuition/classusr.h>  
 ROM Kernel Manual boopsi section

## 1.48 amiga.lib/sprintf

### NAME

sprintf - format a C-like string into a string buffer.

### SYNOPSIS

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

### FUNCTION

Performs string formatting identical to printf, but directs the output into a specific destination in memory. This uses the ROM version of printf (exec.library/RawDoFmt()), 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 (see printf() for a description of this string).

value(s) - numeric information to be formatted into the output stream.

### SEE ALSO

printf(), exec.library/RawDoFmt()

## 1.49 amiga.lib/stdio

### NAMES

fclose - close a file

fgetc - get a character from a file

fprintf - format data to file (see printf())

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

### FUNCTION

These functions work much like the standard C functions of the same names. The file I/O functions all use non-buffered AmigaDOS files, and must not be mixed with the file I/O of any C compiler. The names of these functions 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 dos.library/Open() or dos.library/Output() functions 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.
```

## 1.50 amiga.lib/TimeDelay

NAME

TimeDelay -- Return after a period of time has elapsed.

SYNOPSIS

```
Error = TimeDelay( Unit, Seconds, MicroSeconds )
D0                      D0      D1      D2
```

```
LONG TimeDelay( LONG, ULONG, ULONG );
```

FUNCTION

Waits for the period of time specified before returning to the the caller.

INPUTS

Unit -- timer.device unit to open for this command.

Seconds -- The seconds field of a timerequest is filled with this value. Check the documentation for what a particular timer.device unit expects there.

MicroSeconds -- The microseconds field of a timerequest is filled with this value. Check the documentation for what a particular timer.device units expects there.

RESULTS

Error -- will be zero if all went well; otherwise, non-zero.

NOTES

Two likely reasons for failures are invalid unit numbers or no more free signal bits for this task.

While this function first appears in V37 amiga.lib, it works on Kickstart V33 and higher.

SEE ALSO

```
timer.device/TR_ADDREQUEST,
timer.device/TR_WAITUNTIL,
timer.device/WaitUnitl()
```

BUGS

## 1.51 pools.lib/LibAllocPooled

NAME

LibAllocPooled -- Allocate memory with the pool manager (V33)

SYNOPSIS

```
memory=LibAllocPooled(poolHeader,memSize)
d0                      a0          d0
```

```
void *LibAllocPooled(void *,ULONG);
```

FUNCTION

This function is a copy of the pool functions in V39 and up of EXEC. In fact, if you are running in V39, this function will notice and call the EXEC function. This function works in V33 and up (1.2) Amiga system.

The C code interface is `_LibAllocPooled()` and takes its arguments from the stack just like the C code interface for `AllocPooled()` in `amiga.lib`. The assembly code interface is with the symbol `_AsmAllocPooled:` and takes the parameters in registers with the additional parameter of `ExecBase` being in `a6` which can be used from SAS/C 6 by a prototype of:

```
void * __asm AsmAllocPooled(register __a0 void *,
                           register __d0 ULONG,
                           register __a6 struct ExecBase *);
```

Allocate `memSize` bytes of memory, and return a pointer. NULL is returned if the allocation fails.

Doing a `LibDeletePool()` on the pool will free all of the puddles and thus all of the allocations done with `LibAllocPooled()` in that pool. (No need to `LibFreePooled()` each allocation)

INPUTS

`memSize` - the number of bytes to allocate  
`poolHeader` - a specific private pool header.

RESULT

A pointer to the memory, or NULL.  
 The memory block returned is long word aligned.

NOTES

The pool function do not protect an individual pool from multiple accesses. The reason is that in most cases the pools will be used by a single task. If your pool is going to be used by more than one task you must Semaphore protect the pool from having more than one task trying to allocate within the same pool at the same time. Warning: `Forbid()` protection *\*will not work\** in the future. *\*Do NOT\** assume that we will be able to make it work in the future. `LibAllocPooled()` may well break a `Forbid()` and as such can only be protected



by a semaphore.

To track sizes yourself, the following code can be used:

\*Assumes a6=ExecBase\*

```

;
; Function to do AllocVecPooled(Pool,memSize)
;
AllocVecPooled: addq.l  #4,d0    ; Get space for tracking
                move.l  d0,-(sp) ; Save the size
                jsr LibAllocPooled ; Call pool...
                move.l  (sp)+,d1 ; Get size back...
                tst.l  d0      ; Check for error
                beq.s  avp_fail ; If NULL, failed!
                move.l  d0,a0   ; Get pointer...
                move.l  d1,(a0)+ ; Store size
                move.l  a0,d0   ; Get result
avp_fail: rts      ; return

;
; Function to do LibFreeVecPooled(pool,memory)
;
FreeVecPooled: move.l  -(a1),d0 ; Get size / adjust pointer
                jmp LibFreePooled

```

SEE ALSO

FreePooled(), CreatePool(), DeletePool(),  
LibFreePooled(), LibCreatePool(), LibDeletePool()

## 1.52 pools.lib/LibCreatePool

NAME

LibCreatePool -- Generate a private memory pool header (V33)

SYNOPSIS

```
newPool=LibCreatePool(memFlags,puddleSize,threshSize)
a0                      d0          d1          d2
```

```
void *LibCreatePool(ULONG,ULONG,ULONG);
```

FUNCTION

This function is a copy of the pool functions in V39 and up of EXEC. In fact, if you are running in V39, this function will notice and call the EXEC function. This function works in V33 and up (1.2) Amiga system.

The C code interface is `_LibCreatePool()` and takes its arguments from the stack just like the C code interface for `CreatePool()` in `amiga.lib`. The assembly code interface is with the symbol `_AsmCreatePool:` and takes the parameters in registers with the additional parameter of `ExecBase` being in `a6` which can be used from SAS/C 6 by a prototype of:

```
void * __asm AsmCreatePool(register __d0 ULONG,
                          register __d1 ULONG,
```

```
register __d2 ULONG,
register __a6 struct ExecBase *);
```

Allocate and prepare a new memory pool header. Each pool is a separate tracking system for memory of a specific type. Any number of pools may exist in the system.

Pools automatically expand and shrink based on demand. Fixed sized "puddles" are allocated by the pool manager when more total memory is needed. Many small allocations can fit in a single puddle. Allocations larger than the threshSize are allocation in their own puddles.

At any time individual allocations may be freed. Or, the entire pool may be removed in a single step.

#### INPUTS

memFlags - a memory flags specifier, as taken by AllocMem.  
 puddleSize - the size of Puddles...  
 threshSize - the largest allocation that goes into normal puddles  
               This \*MUST\* be less than or equal to puddleSize  
               (LibCreatePool() will fail if it is not)

#### RESULT

The address of a new pool header, or NULL for error.

#### SEE ALSO

DeletePool(), AllocPooled(), FreePooled(), exec/memory.i,  
 LibDeletePool(), LibAllocPooled(), LibFreePooled()

## 1.53 pools.lib/LibDeletePool

#### NAME

LibDeletePool -- Drain an entire memory pool (V33)

#### SYNOPSIS

```
LibDeletePool(poolHeader)
               a0
```

```
void LibDeletePool(void *);
```

#### FUNCTION

This function is a copy of the pool functions in V39 and up of EXEC. In fact, if you are running in V39, this function will notice and call the EXEC function. This function works in V33 and up (1.2) Amiga system.

The C code interface is \_LibDeletePool() and takes its arguments from the stack just like the C code interface for DeletePool() in amiga.lib. The assembly code interface is with the symbol \_AsmDeletePool: and takes the parameters in registers with the additional parameter of ExecBase being in a6 which can be used from SAS/C 6 by a prototype of:

```
void __asm AsmDeletePool(register __a0 void *,
```

---

```
register __a6 struct ExecBase *);
```

Frees all memory in all puddles of the specified pool header, then deletes the pool header. Individual free calls are not needed.

#### INPUTS

poolHeader - as returned by LibCreatePool().

#### SEE ALSO

CreatePool(), AllocPooled(), FreePooled(),  
LibCreatePool(), LibAllocPooled(), LibFreePooled()

## 1.54 pools.lib/LibFreePooled

#### NAME

LibFreePooled -- Free pooled memory (V33)

#### SYNOPSIS

```
LibFreePooled(poolHeader,memory,memSize)
               a0          a1          d0
```

```
void LibFreePooled(void *,void *,ULONG);
```

#### FUNCTION

This function is a copy of the pool functions in V39 and up of EXEC. In fact, if you are running in V39, this function will notice and call the EXEC function. This function works in V33 and up (1.2) Amiga system.

The C code interface is \_LibFreePooled() and takes its arguments from the stack just like the C code interface for FreePooled() in amiga.lib. The assembly code interface is with the symbol \_AsmFreePooled: and takes the parameters in registers with the additional parameter of ExecBase being in a6 which can be used from SAS/C 6 by a prototype of:

```
void __asm AsmFreePooled(register __a0 void *,
                          register __a1 void *,
                          register __d0 ULONG,
                          register __a6 struct ExecBase *);
```

Deallocates memory allocated by LibAllocPooled(). The size of the allocation *MUST* match the size given to LibAllocPooled(). The reason the pool functions do not track individual allocation sizes is because many of the uses of pools have small allocation sizes and the tracking of the size would be a large overhead.

Only memory allocated by LibAllocPooled() may be freed with this function!

Doing a LibDeletePool() on the pool will free all of the puddles and thus all of the allocations done with LibAllocPooled() in that pool. (No need to LibFreePooled() each allocation)

#### INPUTS

memory - pointer to memory allocated by AllocPooled.  
 poolHeader - a specific private pool header.

#### NOTES

The pool function do not protect an individual pool from multiple accesses. The reason is that in most cases the pools will be used by a single task. If your pool is going to be used by more than one task you must Semaphore protect the pool from having more than one task trying to allocate within the same pool at the same time. Warning: Forbid() protection \*will not work\* in the future. \*Do NOT\* assume that we will be able to make it work in the future. LibFreePooled() may well break a Forbid() and as such can only be protected by a semaphore.

To track sizes yourself, the following code can be used:  
 \*Assumes a6=ExecBase\*

```
;
; Function to do AllocVecPooled(Pool,memSize)
;
AllocVecPooled: addq.l  #4,d0    ; Get space for tracking
                move.l  d0,-(sp) ; Save the size
                jsr LibAllocPooled ; Call pool...
                move.l  (sp)+,d1 ; Get size back...
                tst.l  d0      ; Check for error
                beq.s  avp_fail ; If NULL, failed!
                move.l  d0,a0   ; Get pointer...
                move.l  d1,(a0)+ ; Store size
                move.l  a0,d0   ; Get result
avp_fail: rts      ; return

;
; Function to do LibFreeVecPooled(pool,memory)
;
FreeVecPooled: move.l  -(a1),d0 ; Get size / ajust pointer
                jmp LibFreePooled
```

#### SEE ALSO

AllocPooled(), CreatePool(), DeletePool(),  
 LibAllocPooled(), LibCreatePool(), LibDeletePool()