Example Device

From AmigaOS Documentation Wiki

Main Page > DeveloperDoc:Main > Devices > Example Device

Code samples on this page are not yet updated to AmigaOS 4.x some of them may be obsolete or incompatible with AmigaOS 4.x.

Contents

Example Device

This appendix contains source code for a sample device. The example code is an excellent starting point for those who want to create a custom device and add it to the Amiga’s system software.

The example is a complete four-unit, static-sized RAM disk that works under the old (standard) filing system, the new Fast Filing System (FFS), and has optional code to bind it to an *AUTOCONFIG* device.

The examples have been assembled under the Metacomco assembler V11.0 and under the CAPE assembler V2.0.

ramdev-mountlist

```
/*
 * Mountlist for manually mounting the sample ramdisk driver.
 *
 * F0: and F1: are set up for the V1.3 fast file system (FFS).
 * S2: and S3: are setup for the old file system (OFS).
 *
 * After mounting, the drives must be formatted.  Be sure to
 * use the FFS flag when formatting the Fast File System
 * ramdrives:
 *
 *  ;make sure "ramdev.device" is in DEVS:
 *
 *  mount f0: from mydev-mountlist
 *  format drive f0: name "Zippy" FFS
 */
```

```
F0:    Device = ramdev.device
       Unit   = 0
       LowCyl = 0 ; HighCyl = 14
       Surfaces  = 1
       Buffers = 1
       BlocksPerTrack = 10
       Flags  = 0
       Reserved = 2
       GlobVec = -1
       BufMemType = 0
       DosType = 0x444F5301
       StackSize = 4000
       FileSystem = l:fastfilesystem
#
F1:    Device = ramdev.device
       Unit   = 1
       LowCyl = 0 ; HighCyl = 14
       Surfaces  = 1
       Buffers = 1
       BlocksPerTrack = 10
       Flags  = 0
       Reserved = 2
       GlobVec = -1
       BufMemType = 0
       DosType = 0x444F5301
       StackSize = 4000
       FileSystem = l:fastfilesystem
#
S2:    Device = ramdev.device
       Unit   = 2
       Flags  = 0
       Surfaces  = 1
       BlocksPerTrack = 10
       Reserved = 1
       Interleave = 0
       LowCyl = 0  ;  HighCyl = 14
       Buffers = 1
       BufMemType = 0
#
S3:    Device = ramdev.device
       Unit   = 3
       Flags  = 0
       Surfaces  = 1
       BlocksPerTrack = 10
       Reserved = 1
       Interleave = 0
       LowCyl = 0  ;  HighCyl = 14
       Buffers = 1
       BufMemType = 0
#
```

ramdev.i

```
******************************************************************
*
*
* Copyright (C) 1986, Amiga Inc.  All rights reserved.
* Permission granted for non-commercial use
*
*****************************************************************
*
* ramdev.i -- external declarations for skeleton ramdisk device
*
*****************************************************************


;--- Assemble-time options
INFO_LEVEL  EQU 0     ; Specify amount of debugging info desired
              ; If > 0 you must link with debug.lib!
              ; You will need to run a terminal program to
              ; set the baud rate.
*INTRRUPT   SET 1     ; Remove "*" to enable fake interrupt code
AUTOMOUNT   EQU 0     ; Work with the "mount" command if 0
              ; Do it automatically if 1
```

```
;--- stack size and priority for the process we will create
MYPROCSTACKSIZE   EQU   $900
MYPROCPRI     EQU   0   ;Devices are often 5, NOT higher

;--- Base constants
NUMBEROFTRACKS EQU  40  ;<<<< Change THIS to change size of ramdisk <<<<
SECTOR         EQU  512 ;# bytes per sector
SECSHIFT       EQU  9   ;Shift count to convert byte # to sector #
SECTORSPER     EQU  10  ;# Sectors per "track"

RAMSIZE       EQU   SECTOR*NUMBEROFTRACKS*SECTORSPER
            ; Use this much RAM per unit
BYTESPERTRACK EQU   SECTORSPER*SECTOR

IAMPULLING    EQU   7     ; "I am pulling the interrupt" bit of INTCRL1
INTENABLE     EQU   4     ; "Interrupt Enable" bit of INTCRL2
INTCTRL1      EQU   $40   ; Interrupt control register offset on board
INTCTRL2      EQU   $42   ; Interrupt control register offset on board
INTACK        EQU   $50   ; My board's interrupt reset address
;-----------------------------------------------------------------------
;
; device command definitions (copied from devices/trackdisk.i)
;
;-----------------------------------------------------------------------
   BITDEF  TD,EXTCOM,15      ; for "extended" commands !!!

   DEVINIT
   DEVCMD   CMD_MOTOR        ; control the disk's motor (NO-OP)
   DEVCMD   CMD_SEEK         ; explicit seek (NO-OP)
   DEVCMD   CMD_FORMAT       ; format disk - equated to WRITE for RAMDISK
   DEVCMD   CMD_REMOVE       ; notify when disk changes (NO-OP)
   DEVCMD   CMD_CHANGENUM    ; number of disk changes (always 0)
   DEVCMD   CMD_CHANGESTATE  ; is there a disk in the drive? (always TRUE)
   DEVCMD   CMD_PROTSTATUS   ; is the disk write protected? (always FALSE)
   DEVCMD   CMD_RAWREAD      ; Not supported
   DEVCMD   CMD_RAWWRITE     ; Not supported
   DEVCMD   CMD_GETDRIVETYPE ; Get drive type
   DEVCMD   CMD_GETNUMTRACKS ; Get number of tracks
   DEVCMD   CMD_ADDCHANGEINT ; Add disk change interrupt (NO-OP)
   DEVCMD   CMD_REMCHANGEINT ; Remove disk change interrupt ( NO-OP)
   DEVCMD   MYDEV_END        ; place marker -- first illegal command #

DRIVE3_5    EQU 1
DRIVE5_25   EQU 2
;-----------------------------------------------------------------------
;
; Layout of parameter packet for MakeDosNode
;
;-----------------------------------------------------------------------

    STRUCTURE MkDosNodePkt,0
   APTR    mdn_dosName   ; Pointer to DOS file handler name
   APTR    mdn_execName   ; Pointer to device driver name
   ULONG   mdn_unit   ; Unit number
   ULONG   mdn_flags   ; OpenDevice flags
   ULONG   mdn_tableSize   ; Environment size
   ULONG   mdn_sizeBlock   ; # longwords in a block
   ULONG   mdn_secOrg   ; sector origin -- unused
   ULONG   mdn_numHeads   ; number of surfaces
   ULONG   mdn_secsPerBlk   ; secs per logical block -- unused
   ULONG   mdn_blkTrack   ; secs per track
   ULONG   mdn_resBlks   ; reserved blocks -- MUST be at least 1!
   ULONG   mdn_prefac   ; unused
   ULONG   mdn_interleave   ; interleave
   ULONG   mdn_lowCyl   ; lower cylinder
   ULONG   mdn_upperCyl   ; upper cylinder
   ULONG   mdn_numBuffers   ; number of buffers
   ULONG   mdn_memBufType   ; Type of memory for AmigaDOS buffers
   STRUCT  mdn_dName,5   ; DOS file handler name "RAM0"
   LABEL   mdn_Sizeof   ; Size of this structure

;-----------------------------------------------------------------------
;
; device data structures
;
;-----------------------------------------------------------------------
; maximum number of units in this device
MD_NUMUNITS   EQU   4
```

```
    STRUCTURE MyDev,LIB_SIZE
   UBYTE   md_Flags
   UBYTE   md_Pad1
   ;now longword aligned
   ULONG   md_SysLib
   ULONG   md_SegList
   ULONG   md_Base  ; Base address of this device's expansion board
   STRUCT  md_Units,MD_NUMUNITS*4
   LABEL   MyDev_Sizeof

   STRUCTURE MyDevUnit,UNIT_SIZE   ;Odd # longwords
   UBYTE    mdu_UnitNum
   UBYTE    mdu_SigBit      ; Signal bit allocated for interrupts
   ;Now longword aligned!
   APTR     mdu_Device
   STRUCT   mdu_stack,MYPROCSTACKSIZE
   STRUCT   mdu_tcb,TC_SIZE ; Task Control Block (TCB) for disk task
   ULONG    mdu_SigMask     ; Signal these bits on interrupt
   IFD   INTRRUPT
    STRUCT   mdu_is,IS_SIZE  ; Interrupt structure
    UWORD    mdu_pad1       ;Longword align
   ENDC
   STRUCT   mdu_RAM,RAMSIZE ; RAM used to simulate disk
   LABEL    MyDevUnit_Sizeof

   ;------ state bit for unit stopped
   BITDEF   MDU,STOPPED,2

MYDEVNAME   MACRO
      DC.B   'ramdev.device',0
      ENDM
```

asmsupp.i

```
*************************************************************************
*
*   Copyright (C) 1985, Amiga Inc.  All rights reserved.
*   Permission granted for non-commercial use
*
* asmsupp.i -- random low level assembly support routines
*          used by the sample Library & Device
*
*************************************************************************
CLEAR   MACRO       ;quick way to clear a D register on 68000
    MOVEQ   #0,\1
    ENDM

;BHS     MACRO
;    BCC.\0  \1 ;\0 is the extension used on the macro (such as ".s")
;    ENDM
;BLO     MACRO
;    BCS.\0  \1
;    ENDM
;EVEN    MACRO       ; word align code stream
;    DS.W    0
;    ENDM

LINKSYS MACRO       ; link to a library without having to see a _LVO
    MOVE.L  A6,-(SP)
    MOVE.L  \2,A6
    JSR _LVO\1(A6)
    MOVE.L  (SP)+,A6
    ENDM

CALLSYS MACRO       ; call a library via A6 without having to see _LVO
    JSR _LVO\1(A6)
    ENDM

XLIB    MACRO       ; define a library reference without the _LVO
    XREF    _LVO\1
    ENDM
;
; Put a message to the serial port at 9600 baud.  Used as so:
```

```
;
;     PUTMSG   30,<'%s/Init: called'>
;
; Parameters can be printed out by pushing them on the stack and
; adding the appropriate C printf-style % formatting commands.
;
        XREF    KPutFmt
PUTMSG:     MACRO   * level,msg

        IFGE    INFO_LEVEL-\1

        PEA subSysName(PC)
        MOVEM.L A0/A1/D0/D1,-(SP)
        LEA msg\@(pc),A0    ;Point to static format string
        LEA 4*4(SP),A1  ;Point to args
        JSR KPutFmt
        MOVEM.L (SP)+,D0/D1/A0/A1
        ADDQ.L  #4,SP
        BRA.S   end\@

msg\@       DC.B    \2
        DC.B    10
        DC.B    0
        DS.W    0
end\@
        ENDC
        ENDM
```

ramdev.device.asm

```
*************************************************************************
*
*   Copyright (C) 1986,1988,1989 Amiga Inc.  All rights reserved.
*   Permission granted for non-commercial use.
*
*************************************************************************
*
* ramdev.asm -- Skeleton device code.
*
* A sample 4 unit ramdisk that can be bound to an expansion slot device,
* or used without.  Works with the Fast File System.
* This code is required reading for device driver writers.  It contains
* information not found elsewhere.  This code is somewhat old; you probably
* don't want to copy it directly.
*
* This example includes a task, though a task is not actually needed for
* a simple ram disk.  Unlike a single set of hardware registers that
* may need to be shared by multiple tasks, ram can be freely shared.
* This example does not show arbitration of hardware resources.
*
* Tested with CAPE and Metacomco
*
*      Based on mydev.asm
*      10/07/86 Modified by Lee Erickson to be a simple disk device
*           using RAM to simulate a disk.
*      02/02/88 Modified by C. Scheppner, renamed ramdev
*      09/28/88 Repaired by Bryce Nesbitt for new release
*      11/02/88 More clarifications
*      02/01/89 Even more clarifications & warnings
*      02/22/89 START/STOP fix from Marco Papa
*
* Bugs: If RTF_AUTOINIT fails, library base still left in memory.
*
*************************************************************************

   SECTION firstsection

   NOLIST
   include "exec/types.i"
   include "exec/devices.i"
   include "exec/initializers.i"
   include "exec/memory.i"
   include "exec/resident.i"
   include "exec/io.i"
   include "exec/ables.i"
   include "exec/errors.i"
   include "exec/tasks.i"
   include "hardware/intbits.i"

   include "asmsupp.i"  ;standard asmsupp.i, same as used for library
   include "ramdev.i"

   IFNE AUTOMOUNT
   include "libraries/expansion.i"
   include "libraries/configvars.i"
   include "libraries/configregs.i"
   ENDC
   LIST


ABSEXECBASE equ 4   ;Absolute location of the pointer to exec.library base


   ;------ These don't have to be external, but it helps some
   ;------ debuggers to have them globally visible
   XDEF   Init
   XDEF   Open
   XDEF   Close
   XDEF   Expunge
   XDEF   Null
   XDEF   myName
   XDEF   BeginIO
   XDEF   AbortIO

   ;Pull these _LVOs in from amiga.lib
   XLIB   AddIntServer
   XLIB   RemIntServer
   XLIB   Debug
   XLIB   InitStruct
   XLIB   OpenLibrary
   XLIB   CloseLibrary
   XLIB   Alert
   XLIB   FreeMem
   XLIB   Remove
   XLIB   AddPort
   XLIB   AllocMem
   XLIB   AddTask
   XLIB   PutMsg
   XLIB   RemTask
   XLIB   ReplyMsg
   XLIB   Signal
   XLIB   GetMsg
   XLIB   Wait
   XLIB   WaitPort
   XLIB   AllocSignal
   XLIB   SetTaskPri
   XLIB   GetCurrentBinding ;Use to get list of boards for this driver
   XLIB   MakeDosNode
   XLIB   AddDosNode
   XLIB   CopyMemQuick  ;Highly optimized copy function from exec.library

   INT_ABLES        ;Macro from exec/ables.i


;-----------------------------------------------------------------------
; The first executable location.  This should return an error
; in case someone tried to run you as a program (instead of
; loading you as a device).

FirstAddress:
        moveq   #-1,d0
        rts

;-----------------------------------------------------------------------
; A romtag structure.  After your driver is brought in from disk, the
; disk image will be scanned for this structure to discover magic constants
; about you (such as where to start running you from...).
;-----------------------------------------------------------------------

   ; Most people will not need a priority and should leave it at zero.
   ; the RT_PRI field is used for configuring the roms.  Use "mods" from
   ; wack to look at the other romtags in the system
MYPRI   EQU   0

```

```
initDDescrip:
                ;STRUCTURE RT,0
     DC.W    RTC_MATCHWORD  ; UWORD RT_MATCHWORD (Magic cookie)
     DC.L    initDDescrip   ; APTR  RT_MATCHTAG  (Back pointer)
     DC.L    EndCode        ; APTR  RT_ENDSKIP   (To end of this hunk)
     DC.B    RTF_AUTOINIT   ; UBYTE RT_FLAGS     (magic-see "Init:")
     DC.B    VERSION        ; UBYTE RT_VERSION
     DC.B    NT_DEVICE      ; UBYTE RT_TYPE      (must be correct)
     DC.B    MYPRI      ; BYTE  RT_PRI
     DC.L    myName     ; APTR  RT_NAME      (exec name)
     DC.L    idString       ; APTR  RT_IDSTRING  (text string)
     DC.L    Init       ; APTR  RT_INIT
           ; LABEL RT_SIZE


   ;This name for debugging use
   IFNE INFO_LEVEL  ;If any debugging enabled at all
subSysName:
    dc.b    "ramdev",0
   ENDC

   ; this is the name that the device will have
myName:      MYDEVNAME

 IFNE  AUTOMOUNT
ExLibName    dc.b 'expansion.library',0   ; Expansion Library Name
 ENDC

   ; a major version number.
VERSION:    EQU   37

   ; A particular revision.  This should uniquely identify the bits in the
   ; device.  I use a script that advances the revision number each time
   ; I recompile.  That way there is never a question of which device
   ; that really is.
REVISION:   EQU   1

   ; this is an identifier tag to help in supporting the device
   ; format is 'name version.revision (d.m.yy)',<cr>,<lf>,<null>
idString:   dc.b   'ramdev 37.1 (28.8.91)',13,10,0

   ; force word alignment
   ds.w   0


   ; The romtag specified that we were "RTF_AUTOINIT".  This means
   ; that the RT_INIT structure member points to one of these
   ; tables below.  If the AUTOINIT bit was not set then RT_INIT
   ; would point to a routine to run.

Init:
   DC.L   MyDev_Sizeof      ; data space size
   DC.L   funcTable     ; pointer to function initializers
   DC.L   dataTable     ; pointer to data initializers
   DC.L   initRoutine       ; routine to run


funcTable:
   ;------ standard system routines
   dc.l   Open
   dc.l   Close
   dc.l   Expunge
   dc.l   Null      ;Reserved for future use!

   ;------ my device definitions
   dc.l   BeginIO
   dc.l   AbortIO

   ;------ custom extended functions
   dc.l   FunctionA
   dc.l   FunctionB

   ;------ function table end marker
   dc.l   -1


   ;The data table initializes static data structures. The format is
   ;specified in exec/InitStruct routine's manual pages.  The
   ;INITBYTE/INITWORD/INITLONG macros are in the file "exec/initializers.i".
   ;The first argument is the offset from the device base for this
   ;byte/word/long. The second argument is the value to put in that cell.
   ;The table is null terminated
   ;
dataTable:
   INITBYTE   LN_TYPE,NT_DEVICE       ;Must be LN_TYPE!
   INITLONG   LN_NAME,myName
   INITBYTE   LIB_FLAGS,LIBF_SUMUSED!LIBF_CHANGED
   INITWORD   LIB_VERSION,VERSION
   INITWORD   LIB_REVISION,REVISION
   INITLONG   LIB_IDSTRING,idString
   DC.W   0   ;terminate list


;-------- initRoutine -------------------------------------------------------
;
; FOR RTF_AUTOINIT:
;   This routine gets called after the device has been allocated.
;   The device pointer is in D0.  The AmigaDOS segment list is in a0.
;   If it returns the device pointer, then the device will be linked
;   into the device list.  If it returns NULL, then the device
;   will be unloaded.
;
; IMPORTANT:
;   If you don't use the "RTF_AUTOINIT" feature, there is an additional
;   caveat.  If you allocate memory in your Open function, remember that
;   allocating memory can cause an Expunge... including an expunge of your
;   device.  This must not be fatal.  The easy solution is don't add your
;   device to the list until after it is ready for action.
;
; This call is single-threaded by exec; please read the description for
; "Open" below.
;
; Register Usage
; ==============
; a3 -- Points to temporary RAM
; a4 -- Expansion library base
; a5 -- device pointer
; a6 -- Exec base
;----------------------------------------------------------------------
initRoutine:
   ;------ get the device pointer into a convenient A register
   PUTMSG   5,<'%s/Init: called'>
   movem.l  d1-d7/a0-a5,-(sp)   ; Preserve ALL modified registers
   move.l   d0,a5

   ;------ save a pointer to exec
   move.l   a6,md_SysLib(a5)    ;faster access than move.l 4,a6

   ;------ save pointer to our loaded code (the SegList)
   move.l   a0,md_SegList(a5)

 IFNE  AUTOMOUNT
*************************************************************************
*
* Here starts the AutoConfig stuff.  If this driver was to be tied to
* an expansion board, you would put this driver in the expansion drawer,
* and be called when BindDrivers finds a board that matches this driver.
* The AmigaOS development team assigned product number of your board must be
* specified in the "PRODUCT=" field in the TOOLTYPES of this driver's icon.
* GetCurrentBinding() returns your (first) board.
*
   lea.l     ExLibName,A1   ; Get expansion lib. name
   moveq.l   #0,D0
   CALLSYS   OpenLibrary    ; Open the expansion library
   tst.l     D0
   beq       Init_Error

   ;------ init_OpSuccess:
   move.l    D0,A4      ;[expansionbase to A4]
   moveq     #0,D3
   lea       md_Base(A5),A0   ; Get the Current Bindings
   moveq     #4,D0        ; Just get address (length = 4 bytes)
   LINKLIB   _LVOGetCurrentBinding,A4
   move.l    md_Base(A5),D0  ; Get start of list
   tst.l     D0        ; If controller not found
   beq       Init_End      ; Exit and unload driver
   PUTMSG    10,<'%s/Init: GetCurrentBinding returned non-zero'>
   move.l    D0,A0     ; Get config structure address
   move.l    cd_BoardAddr(A0),md_Base(A5); Save board base address
   bclr.b    #CDB_CONFIGME,cd_Flags(A0); Mark board as configured

;----------------------------------------------------------------------
;
; Here we build a packet describing the characteristics of our disk to
; pass to AmigaDOS.  This serves the same purpose as a "mount" command
; of this device would.  For disks, it might be useful to actually
; get this information right from the disk itself.  Just as mount,
; it could be for multiple partitions on the single physical device.
; For this example, we will simply hard code the appropriate parameters.
;
; The AddDosNode call adds things to dos's list without needing to
; use mount.  We'll mount all 4 of our units whenever we are
; started.
;
;-----------------------------------------------------------------------

;!!! If your card was successfully configured, you can mount the
;!!! units as DOS nodes

   ;------   Allocate temporary RAM to build MakeDosNode parameter packet
   move.l    #MEMF_CLEAR!MEMF_PUBLIC,d1
   move.l    #mdn_Sizeof,d0   ; Enough room for our parameter packet
   CALLSYS   AllocMem
   move.l    d0,a3      ;:BUG: AllocMem error not checked here.

   ;-----   Use InitStruct to initialize the constant portion of packet
   move.l    d0,a2     ; Point to memory to initialize
   moveq.l   #0,d0     ; Don't need to re-zero it
   lea.l     mdn_Init(pc),A1
   CALLSYS   InitStruct

   lea       mdn_dName(a3),a0     ; Get addr of Device name
   move.l    a0,mdn_dosName(a3)   ;   and save in environment

   moveq     #0,d6        ; Now tell AmigaDOS about all units UNITNUM
Uloop:
   move.b    d6,d0       ; Get unit number
   add.b     #$30,d0         ; Make ASCII, minus 1
   move.b    d0,mdn_dName+2(a3)   ;   and store in name
   move.l    d6,mdn_unit(a3)      ; Store unit # in environment

;
;! Before adding to the dos list, you should really check if you
;! are about to cause a name collision.  This example does not.
;

   move.l    a3,a0
   LINKLIB   _LVOMakeDosNode,a4   ; Build AmigaDOS structures
   ;This can fail, but so what?
   move.l    d0,a0        ; Get deviceNode address
   moveq.l   #0,d0        ; Set device priority to 0
   moveq.l   #0,d1
*  moveq.l   #ADNF_STARTPROC,d1     ; See note below
   ;It's ok to pass a zero in here
   LINKLIB   _LVOAddDosNode,a4


; ADNF_STARTPROC will work, but only if dn_SegList is filled in
; in the SegPtr of the handler task.


   addq     #1,d6     ; Bump unit number
   cmp.b    #MD_NUMUNITS,d6
   bls.s    Uloop     ; Loop until all units installed

   move.l   a3,a1      ; Return RAM to system
   move.l   #mdn_Sizeof,d0
   CALLSYS  FreeMem

Init_End:

   move.l   a4,a1      ; Now close expansion library
   CALLSYS  CloseLibrary
*
*   You would normally set d0 to a NULL if your initialization failed,
*   but I'm not doing that for this demo, since it is unlikely
*   you actually have a board with any particular manufacturer ID
*   installed when running this demo.
************************************************************************
  ENDC

   move.l   a5,d0
Init_Error:
   movem.l  (sp)+,d1-d7/a0-a5
   rts


;----------------------------------------------------------------------
;
; Here begins the system interface commands.  When the user calls
; OpenDevice/CloseDevice/RemDevice, this eventually gets translated
; into a call to the following routines (Open/Close/Expunge).
; Exec has already put our device pointer in a6 for us.
;
; IMPORTANT:
;   These calls are guaranteed to be single-threaded; only one task
;   will execute your Open/Close/Expunge at a time.
;
;   For Kickstart V33/34, the single-threading method involves "Forbid".
;   There is a good chance this will change.  Anything inside your
;   Open/Close/Expunge that causes a direct or indirect Wait() will break
;   the Forbid().  If the Forbid() is broken, some other task might
;   manage to enter your Open/Close/Expunge code at the same time.
;   Take care!
;
; Since exec has turned off task switching while in these routines
; (via Forbid/Permit), we should not take too long in them.
;
;----------------------------------------------------------------------

   ; Open sets the IO_ERROR field on an error.  If it was successfull,
   ; we should also set up the IO_UNIT and LN_TYPE fields.
   ; exec takes care of setting up IO_DEVICE.

Open:      ; ( device:a6, iob:a1, unitnum:d0, flags:d1 )

;** Subtle point: any AllocMem() call can cause a call to this device's
;** expunge vector.  If LIB_OPENCNT is zero, the device might get expunged.
   addq.w   #1,LIB_OPENCNT(a6)  ;Fake an opener for duration of call <|>

   PUTMSG   20,<'%s/Open: called'>
   movem.l  d2/a2/a3/a4,-(sp)

   move.l   a1,a2      ; save the iob

   ;------ see if the unit number is in range   *!* UNIT 0 to 3 *!*
   cmp.l   #MD_NUMUNITS,d0
   bcc.s   Open_Range_Error   ; unit number out of range (BHS)

   ;------ see if the unit is already initialized
   move.l   d0,d2      ; save unit number
   lsl.l    #2,d0
   lea.l    md_Units(a6,d0.l),a4
   move.l   (a4),d0
   bne.s    Open_UnitOK

   ;------ try and conjure up a unit
   bsr      InitUnit    ;scratch:a3 unitnum:d2 devpoint:a6

   ;------ see if it initialized OK
   move.l   (a4),d0
   beq.s    Open_Error

Open_UnitOK:
   move.l   d0,a3      ; unit pointer in a3
   move.l   d0,IO_UNIT(a2)

   ;------ mark us as having another opener
   addq.w   #1,LIB_OPENCNT(a6)
   addq.w   #1,UNIT_OPENCNT(a3)     ;Internal bookkeeping

   ;------ prevent delayed expunges
   bclr     #LIBB_DELEXP,md_Flags(a6)
   CLEAR    d0
   move.b   d0,IO_ERROR(a2)
   move.b   #NT_REPLYMSG,LN_TYPE(a2) ;IMPORTANT: Mark IORequest as "complete"

Open_End:

   subq.w   #1,LIB_OPENCNT(a6) ;** End of expunge protection <|>
   movem.l  (sp)+,d2/a2/a3/a4
   rts

Open_Range_Error:
Open_Error:
   moveq    #IOERR_OPENFAIL,d0
   move.b   d0,IO_ERROR(a2)
   move.l   d0,IO_DEVICE(a2)    ;IMPORTANT: trash IO_DEVICE on open failure
   PUTMSG   2,<'%s/Open: failed'>
   bra.s    Open_End


;----------------------------------------------------------------------------
; There are two different things that might be returned from the Close
; routine.  If the device wishes to be unloaded, then Close must return
; the segment list (as given to Init).  Otherwise close MUST return NULL.

Close:      ; ( device:a6, iob:a1 )
   movem.l  d1/a2-a3,-(sp)
   PUTMSG   20,<'%s/Close: called'>

   move.l   a1,a2

   move.l   IO_UNIT(a2),a3

   ;------ IMPORTANT: make sure the IORequest is not used again
   ;------ with a -1 in IO_DEVICE, any BeginIO() attempt will
   ;------ immediatly halt (which is better than a subtle corruption
   ;------ that will lead to hard-to-trace crashes!!!!!!!!!!!!!!!!!!
   moveq.l  #-1,d0
   move.l   d0,IO_UNIT(a2)  ;We're closed...
   move.l   d0,IO_DEVICE(a2)    ;customers not welcome at this IORequest!!

   ;------ see if the unit is still in use
   subq.w   #1,UNIT_OPENCNT(a3)

;!!!!!! Since this example is a RAM disk (and we don't want the contents to
;!!!!!! disappear between opens, ExpungeUnit will be skipped here.  It would
;!!!!!! be used for drivers of "real" devices
;!!!!!!   bne.s   Close_Device
;!!!!!!   bsr     ExpungeUnit

Close_Device:
   CLEAR   d0
   ;------ mark us as having one fewer openers
   subq.w  #1,LIB_OPENCNT(a6)

   ;------ see if there is anyone left with us open
   bne.s   Close_End

   ;------ see if we have a delayed expunge pending
   btst    #LIBB_DELEXP,md_Flags(a6)
   beq.s   Close_End

   ;------ do the expunge
   bsr     Expunge

Close_End:
   movem.l   (sp)+,d1/a2-a3
   rts              ;MUST return either zero or the SegList!!!


;------- Expunge -----------------------------------------------------------
;
; Expunge is called by the memory allocator when the system is low on
; memory.
;
; There are two different things that might be returned from the Expunge
; routine.  If the device is no longer open then Expunge may return the
; segment list (as given to Init).  Otherwise Expunge may set the
; delayed expunge flag and return NULL.
;
```

```
; One other important note: because Expunge is called from the memory
; allocator, it may NEVER Wait() or otherwise take long time to complete.
;
;    A6      - library base (scratch)
;    D0-D1/A0-A1 - scratch
;
Expunge:   ; ( device: a6 )
   PUTMSG   10,<'%s/Expunge: called'>

   movem.l  d1/d2/a5/a6,-(sp)   ; Save ALL modified registers
   move.l   a6,a5
   move.l   md_SysLib(a5),a6

   ;------ see if anyone has us open
   tst.w   LIB_OPENCNT(a5)
;!!!!!  The following line is commented out for this RAM disk demo, since
;!!!!!  we don't want the RAM to be freed after FORMAT, for example.
;   beq    1$

   ;------ it is still open.  set the delayed expunge flag
   bset    #LIBB_DELEXP,md_Flags(a5)
   CLEAR   d0
   bra.s   Expunge_End

1$:
   ;------ go ahead and get rid of us.  Store our seglist in d2
   move.l   md_SegList(a5),d2

   ;------ unlink from device list
   move.l    a5,a1
   CALLSYS   Remove     ;Remove first (before FreeMem)

   ;
   ; device specific closings here...
   ;

   ;------ free our memory (must calculate from LIB_POSSIZE & LIB_NEGSIZE)
   move.l   a5,a1       ;Devicebase
   CLEAR    d0
   move.w   LIB_NEGSIZE(a5),d0
   suba.l   d0,a1       ;Calculate base of functions
   add.w    LIB_POSSIZE(a5),d0  ;Calculate size of functions + data area
   CALLSYS  FreeMem

   ;------ set up our return value
   move.l   d2,d0

Expunge_End:
   movem.l  (sp)+,d1/d2/a5/a6
   rts


;------- Null ---------------------------------------------------------------
Null:
   PUTMSG  1,<'%s/Null: called'>
   CLEAR   d0
   rts      ;The "Null" function MUST return NULL.


;------- Custom ------------------------------------------------------------
;
;Two "do nothing" device-specific functions
;
FunctionA:
    add.l   d1,d0   ;Add
    rts
FunctionB:
    add.l   d0,d0   ;Double
    rts


****************************************************************************

InitUnit:   ; ( d2:unit number, a3:scratch, a6:devptr )
   PUTMSG   30,<'%s/InitUnit: called'>
   movem.l  d2-d4/a2,-(sp)

   ;------ allocate unit memory
   move.l   #MyDevUnit_Sizeof,d0
   move.l   #MEMF_PUBLIC!MEMF_CLEAR,d1
   LINKSYS  AllocMem,md_SysLib(a6)
   tst.l    d0
   beq      InitUnit_End
   move.l   d0,a3

   moveq.l  #0,d0      ; Don't need to re-zero it
   move.l   a3,a2      ; InitStruct is initializing the UNIT
   lea.l    mdu_Init(pc),A1
   LINKSYS  InitStruct,md_SysLib(a6)

   ;!! IMPORTANT !!
   move.l   #42414400,mdu_RAM(a3)   ;Mark offset zero as ASCII "BAD "
   ;!! IMPORTANT !!

   move.b   d2,mdu_UnitNum(a3)      ;initialize unit number
   move.l   a6,mdu_Device(a3)       ;initialize device pointer

   ;------ start up the unit task.  We do a trick here --
   ;------ we set his message port to PA_IGNORE until the
   ;------ new task has a change to set it up.
   ;------ We cannot go to sleep here: it would be very nasty
   ;------ if someone else tried to open the unit
   ;------ (exec's OpenDevice has done a Forbid() for us --
   ;------ we depend on this to become single threaded).

   ;------ Initialize the stack information
   lea      mdu_stack(a3),a0          ; Low end of stack
   move.l   a0,mdu_tcb+TC_SPLOWER(a3)
   lea      MYPROCSTACKSIZE(a0),a0    ; High end of stack
   move.l   a0,mdu_tcb+TC_SPUPPER(a3)
   move.l   a3,-(A0)              ; argument -- unit ptr (send on stack)
   move.l   a0,mdu_tcb+TC_SPREG(a3)
   lea      mdu_tcb(a3),a0
   move.l   a0,MP_SIGTASK(a3)

   IFGE INFO_LEVEL-30
       move.l   a0,-(SP)
       move.l   a3,-(SP)
       PUTMSG   30,<'%s/InitUnit, unit= %lx, task=%lx'>
       addq.l   #8,sp
   ENDC

   ;------ initialize the unit's message port's list
   lea      MP_MSGLIST(a3),a0
   NEWLIST  a0          ;<- IMPORTANT! Lists MUST! have NEWLIST
                ;work magic on them before use.  (AddPort()
                ;can do this for you)

   IFD   INTRRUPT
   move.l   a3,mdu_is+IS_DATA(a3)   ; Pass unit addr to interrupt server
   ENDC

;   Startup the task
   lea      mdu_tcb(a3),a1
   lea      Task_Begin(PC),a2
   move.l   a3,-(sp)      ; Preserve UNIT pointer
   lea      -1,a3     ; generate address error
              ; if task ever "returns" (we RemTask() it
              ; to get rid of it...)
   CLEAR   d0
   PUTMSG   30,<'%s/About to add task'>
   LINKSYS AddTask,md_SysLib(a6)
   move.l   (sp)+,a3      ; restore UNIT pointer

   ;------ mark us as ready to go
   move.l   d2,d0     ; unit number
   lsl.l    #2,d0
   move.l   a3,md_Units(a6,d0.l)   ; set unit table
   PUTMSG   30,<'%s/InitUnit: ok'>

InitUnit_End:
   movem.l   (sp)+,d2-d4/a2
   rts


;---------------------------------------------------------------------------
FreeUnit:   ; ( a3:unitptr, a6:deviceptr )
   move.l   a3,a1
   move.l   #MyDevUnit_Sizeof,d0
   LINKSYS  FreeMem,md_SysLib(a6)
   rts

;----------------------------------------------------------------------------
ExpungeUnit:   ; ( a3:unitptr, a6:deviceptr )
   PUTMSG   10,<'%s/ExpungeUnit: called'>
   move.l   d2,-(sp)

;
; If you can expunge you unit, and each unit has it's own interrupts,
; you must remember to remove its interrupt server
;

   IFD   INTRRUPT
   lea.l   mdu_is(a3),a1          ; Point to interrupt structure
   moveq   #INTB_PORTS,d0         ; Portia interrupt bit 3
   LINKSYS RemIntServer,md_SysLib(a6) ;Now remove the interrupt server
   ENDC

   ;------ get rid of the unit's task.  We know this is safe
   ;------ because the unit has an open count of zero, so it
   ;------ is 'guaranteed' not in use.
   lea   mdu_tcb(a3),a1
   LINKSYS RemTask,md_SysLib(a6)

   ;------ save the unit number
   CLEAR   d2
   move.b  mdu_UnitNum(a3),d2

   ;------ free the unit structure.
   bsr     FreeUnit

   ;------ clear out the unit vector in the device
   lsl.l   #2,d2
   clr.l   md_Units(a6,d2.l)

   move.l  (sp)+,d2
   rts


*****************************************************************************
;
; here begins the device functions
;
;----------------------------------------------------------------------------
; cmdtable is used to look up the address of a routine that will
; implement the device command.
;
; NOTE: the "extended" commands (ETD_READ/ETD_WRITE) have bit 15 set!
; We deliberately refuse to operate on such commands.  However a driver
; that supports removable media may want to implement this.  One
; open issue is the handling of the "seclabel" area. It is probably
; best to reject any command with a non-null "seclabel" pointer.
;
cmdtable:
   DC.L   Invalid   ;$00000001  ;0  CMD_INVALID
   DC.L   MyReset   ;$00000002  ;1  CMD_RESET
   DC.L   RdWrt     ;$00000004  ;2  CMD_READ    (\/common)
   DC.L   RdWrt     ;$00000008  ;3  CMD_WRITE   (/\common)  ETD_
   DC.L   Update    ;$00000010  ;4  CMD_UPDATE  (NO-OP)     ETD_
   DC.L   Clear     ;$00000020  ;5  CMD_CLEAR   (NO-OP)     ETD_
   DC.L   MyStop    ;$00000040  ;6  CMD_STOP            ETD_
   DC.L   Start     ;$00000080  ;7  CMD_START
   DC.L   Flush     ;$00000100  ;8  CMD_FLUSH
   DC.L   Motor     ;$00000200  ;9  TD_MOTOR    (NO-OP)     ETD_
   DC.L   Seek      ;$00000400  ;A  TD_SEEK     (NO-OP)     ETD_
   DC.L   RdWrt     ;$00000800  ;B  TD_FORMAT   (Same as write)
   DC.L   MyRemove  ;$00001000  ;C  TD_REMOVE   (NO-OP)
   DC.L   ChangeNum ;$00002000  ;D  TD_CHANGENUM    (returns 0)
   DC.L   ChangeState   ;$00004000  ;E  TD_CHANGESTATE  (returns 0)
   DC.L   ProtStatus    ;$00008000  ;F  TD_PROTSTATUS   (returns 0)
   DC.L   RawRead   ;$00010000  ;10 TD_RAWREAD  (INVALID)
   DC.L   RawWrite  ;$00020000  ;11 TD_RAWWRITE (INVALID)
   DC.L   GetDriveType  ;$00040000  ;12 TD_GETDRIVETYPE (Returns 1)
   DC.L   GetNumTracks  ;$00080000  ;13 TD_GETNUMTRACKS (Returns NUMTRKS)
   DC.L   AddChangeInt  ;$00100000  ;14 TD_ADDCHANGEINT (NO-OP)
   DC.L   RemChangeInt  ;$00200000  ;15 TD_REMCHANGEINT (NO-OP)
```

```
cmdtable_end:

; this define is used to tell which commands should be handled
; immediately (on the caller's schedule).
;
; The immediate commands are Invalid, Reset, Stop, Start, Flush
;
; Note that this method limits you to just 32 device specific commands,
; which may not be enough.
;IMMEDIATES   EQU   %00000000000000000000000111000011
;;           --------========--------========
;;           FEDCBA9876543210FEDCBA9876543210

;;An alternate version.  All commands that are trivially short
;;and %100 reentrant are included.  This way you won't get the
;;task switch overhead for these commands.
;;
IMMEDIATES   EQU   %11111111111111111111011111110011
;           --------========--------========
;           FEDCBA9876543210FEDCBA9876543210

    IFD   INTRRUPT   ; if using interrupts,
; These commands can NEVER be done "immediately" if using interrupts,
; since they would "wait" for the interrupt forever!
; Read, Write, Format
NEVERIMMED   EQU   $0000080C
    ENDC


;--------------------------------
; BeginIO starts all incoming io.  The IO is either queued up for the
; unit task or processed immediately.
;
;
; BeginIO often is given the responsibility of making devices single
; threaded... so two tasks sending commands at the same time don't cause
; a problem.  Once this has been done, the command is dispatched via
; PerformIO.
;
; There are many ways to do the threading.  This example uses the
; UNITB_ACTIVE bit.  Be sure this is good enough for your device before
; using!  Any method is ok.  If immediate access can not be obtained, the
; request is queued for later processing.
;
; Some IO requests do not need single threading, these can be performed
; immediatley.
;
; IMPORTANT:
;   The exec WaitIO() function uses the IORequest node type (LN_TYPE)
;   as a flag.  If set to NT_MESSAGE, it assumes the request is
;   still pending and will wait.  If set to NT_REPLYMSG, it assumes the
;   request is finished.  It's the responsibility of the device driver
;   to set the node type to NT_MESSAGE before returning to the user.
;
BeginIO:   ; ( iob: a1, device:a6 )

    IFGE INFO_LEVEL-1
    bchg.b  #1,$bfe001  ;Blink the power LED
    ENDC
    IFGE INFO_LEVEL-3
     clr.l    -(sp)
     move.w   IO_COMMAND(a1),2(sp)  ;Get entire word
     PUTMSG   3,<'%s/BeginIO  -- $%lx'>
     addq.l   #4,sp
    ENDC

    movem.l   d1/a0/a3,-(sp)

    move.b  #NT_MESSAGE,LN_TYPE(a1) ;So WaitIO() is guaranteed to work
    move.l  IO_UNIT(a1),a3      ;bookkeeping -> what unit to play with
    move.w  IO_COMMAND(a1),d0

    ;Do a range check & make sure ETD_XXX type requests are rejected
    cmp.w   #MYDEV_END,d0   ;Compare all 16 bits
    bcc     BeginIO_NoCmd   ;no, reject it.  (bcc=bhs - unsigned)

    ;------ process all immediate commands no matter what
    move.l  #IMMEDIATES,d1
    DISABLE a0          ;<-- Ick, nasty stuff, but needed here.
    btst.l  d0,d1
    bne     BeginIO_Immediate

    IFD   INTRRUPT   ; if using interrupts,
     ;------ queue all NEVERIMMED commands no matter what
     move.w  #NEVERIMMED,d1
     btst    d0,d1
     bne.s   BeginIO_QueueMsg
    ENDC


    ;------ see if the unit is STOPPED.  If so, queue the msg.
    btst    #MDUB_STOPPED,UNIT_FLAGS(a3)
    bne     BeginIO_QueueMsg


    ;------ This is not an immediate command.  See if the device is
    ;------ busy.  If the device is not, do the command on the
    ;------ user schedule.  Else fire up the task.
    ;------ This type of arbitration is not really needed for a ram
    ;------ disk, but is essential for a device to reliably work
    ;------ with shared hardware
    ;------
    ;------ When the lines below are ";" commented out, the task gets
    ;------ a better workout.  When the lines are active, the calling
    ;------ process is usually used for the operation.
    ;------
    ;------ REMEMBER:::: Never Wait() on the user's schedule in BeginIO()!
    ;------ The only exception is when the user has indicated it is ok
    ;------ by setting the "quick" bit.  Since this device copies from
    ;------ ram that never needs to be waited for, this subtlely may not
    ;------ be clear.
    ;------
    bset    #UNITB_ACTIVE,UNIT_FLAGS(a3)   ;<---- comment out these
    beq.s   BeginIO_Immediate          ;<---- lines to test task.


    ;------ we need to queue the device.  mark us as needing
    ;------ task attention.  Clear the quick flag
BeginIO_QueueMsg:
    bset    #UNITB_INTASK,UNIT_FLAGS(a3)
    bclr    #IOB_QUICK,IO_FLAGS(a1)   ;We did NOT complete this quickly
    ENABLE  a0


    IFGE INFO_LEVEL-250
     move.l  a1,-(sp)
     move.l  a3,-(sp)
     PUTMSG  250,<'%s/PutMsg: Port=%lx Message=%lx'>
     addq.l  #8,sp
    ENDC

    move.l  a3,a0
    LINKSYS  PutMsg,md_SysLib(a6)   ;Port=a0, Message=a1
    bra.s   BeginIO_End
    ;----- return to caller before completing


    ;------ Do it on the schedule of the calling process
    ;------
BeginIO_Immediate:
    ENABLE  a0
    bsr.s   PerformIO

BeginIO_End:
    PUTMSG  200,<'%s/BeginIO_End'>
    movem.l (sp)+,d1/a0/a3
    rts

BeginIO_NoCmd:
    move.b  #IOERR_NOCMD,IO_ERROR(a1)
    bra.s   BeginIO_End


;
; PerformIO actually dispatches an io request.  It might be called from
; the task, or directly from BeginIO (thus on the callers's schedule)
;
; It expects a3 to already
; have the unit pointer in it.  a6 has the device pointer (as always).
; a1 has the io request.  Bounds checking has already been done on
; the I/O Request.
;

PerformIO:   ; ( iob:a1, unitptr:a3, devptr:a6 )
    IFGE INFO_LEVEL-150
     clr.l    -(sp)
     move.w   IO_COMMAND(a1),2(sp)  ;Get entire word
     PUTMSG   150,<'%s/PerformIO -- $%lx'>
     addq.l   #4,sp
    ENDC

    moveq   #0,d0
    move.b  d0,IO_ERROR(A1) ; No error so far
    move.b  IO_COMMAND+1(a1),d0 ;Look only at low byte
    lsl.w   #2,d0       ; Multiply by 4 to get table offset
    lea.l   cmdtable(pc),a0
    move.l  0(a0,d0.w),a0

    jmp     (a0)    ;iob:a1  unit:a3  devprt:a6



;
; TermIO sends the IO request back to the user.  It knows not to mark
; the device as inactive if this was an immediate request or if the
; request was started from the server task.
;

TermIO:      ; ( iob:a1, unitptr:a3, devptr:a6 )
    PUTMSG  160,<'%s/TermIO'>
    move.w  IO_COMMAND(a1),d0

    move.w  #IMMEDIATES,d1
    btst    d0,d1
    bne.s   TermIO_Immediate    ;IO was immediate, don't do task stuff...

    ;------ we may need to turn the active bit off.
    btst    #UNITB_INTASK,UNIT_FLAGS(a3)
    bne.s   TermIO_Immediate    ;IO was came from task, don't clear ACTIVE...

    ;------ the task does not have more work to do
    bclr    #UNITB_ACTIVE,UNIT_FLAGS(a3)

TermIO_Immediate:
    ;------ if the quick bit is still set then we don't need to reply
    ;------ msg -- just return to the user.
    btst    #IOB_QUICK,IO_FLAGS(a1)
    bne.s   TermIO_End
    LINKSYS ReplyMsg,md_SysLib(a6)  ;a1-message
    ;(ReplyMsg sets the LN_TYPE to NT_REPLYMSG)

TermIO_End:
    rts


*****************************************************************************
;
; Here begins the functions that implement the device commands
; all functions are called with:
;   a1 -- a pointer to the io request block
;   a3 -- a pointer to the unit
;   a6 -- a pointer to the device
;
; Commands that conflict with 68000 instructions have a "My" prepended
; to them.
;----------------------------------------------------------------------

;We can't AbortIO anything, so don't touch the IORequest!
;
;AbortIO() is a REQUEST to "hurry up" processing of an IORequest.
;If the IORequest was already complete, nothing happens (if an IORequest
;is quick or LN_TYPE=NT_REPLYMSG, the IORequest is complete).
;The message must be replied with ReplyMsg(), as normal.
;
AbortIO:    ; ( iob: a1, device:a6 )
    moveq   #IOERR_NOCMD,d0 ;return "AbortIO() request failed"
    rts
```

```
RawRead:    ; 10 Not supported   (INVALID)
RawWrite:   ; 11 Not supported   (INVALID)
Invalid:
    move.b  #IOERR_NOCMD,IO_ERROR(a1)
    bra.s   TermIO

;
; Update and Clear are internal buffering commands.  Update forces all
; io out to its final resting spot, and does not return until this is
; totally done.  Since this is automatic in a ramdisk, we simply return "Ok".
;
; Clear invalidates all internal buffers.  Since this device
; has no internal buffers, these commands do not apply.
;
Update:
Clear:
MyReset:            ;Do nothing (nothing reasonable to do)
AddChangeInt:           ;Do nothing
RemChangeInt:           ;Do nothing
MyRemove:           ;Do nothing
Seek:               ;Do nothing
Motor:              ;Do nothing
ChangeNum:          ;Return zero (changecount =0)
ChangeState:            ;Zero indicates disk inserted
ProtStatus:         ;Zero indicates unprotected
    clr.l   IO_ACTUAL(a1)
    bra.s   TermIO


GetDriveType:           ;make it look like 3.5" (90mm) drive
    moveq   #DRIVE3_5,d0
    move.l  d0,IO_ACTUAL(a1)
    bra.s   TermIO


GetNumTracks:
    move.l  #RAMSIZE/BYTESPERTRACK,IO_ACTUAL(a1) ;Number of tracks
    bra.s   TermIO

;
; Foo and Bar are two device specific commands that are provided just
; to show you how commands are added.  They currently return that
; no work was done.
;
Foo:
Bar:
    clr.l   IO_ACTUAL(a1)
    bra     TermIO


;---------------------------------------------------------------------------
; This device is designed so that no combination of bad
; inputs can ever cause the device driver to crash.
;---------------------------------------------------------------------------
RdWrt:
    IFGE INFO_LEVEL-200
    move.l  IO_DATA(a1),-(sp)
    move.l  IO_OFFSET(a1),-(sp)
    move.l  IO_LENGTH(a1),-(sp)
    PUTMSG  200,<'%s/RdWrt len %ld offset %ld data $%lx'>
    addq.l  #8,sp
    addq.l  #4,sp
    ENDC

    movem.l a2/a3,-(sp)
    move.l  a1,a2       ;Copy iob
    move.l  IO_UNIT(a2),a3  ;Get unit pointer

*      check operation for legality
    btst.b  #0,IO_DATA+3(a2)    ;check if user's pointer is ODD
    bne.s   IO_LenErr       ;bad...
    ;[D0=offset]

    move.l  IO_OFFSET(a2),d0
    move.l  d0,d1
    and.l   #SECTOR-1,d1    ;Bad sector boundary or alignment?
    bne.s   IO_LenErr       ;bad...
    ;[D0=offset]
```

```
*       check for IO within disc range
    ;[D0=offset]
    add.l   IO_LENGTH(a2),d0    ;Add length to offset
    bcs.s   IO_LenErr       ;overflow... (important test)
    cmp.l   #RAMSIZE,d0     ;Last byte is highest acceptable total
    bhi.s   IO_LenErr       ;bad... (unsigned compare)
    and.l   #SECTOR-1,d0    ;Even sector boundary?
    bne.s   IO_LenErr       ;bad...

*       We've gotten this far, it must be a valid request.

    IFD   INTRRUPT
     move.l   mdu_SigMask(a3),d0  ; Get signals to wait for
     LINKSYS  Wait,md_SysLib(a6)  ; Wait for interrupt before proceeding
    ENDC


    lea.l   mdu_RAM(a3),a0  ; Point to RAMDISK "sector" for I/O
    add.l   IO_OFFSET(a2),a0    ; Add offset to ram base
    move.l  IO_LENGTH(a2),d0
    move.l  d0,IO_ACTUAL(a2)    ; Indicate we've moved all bytes
    beq.s   RdWrt_end       ;---deal with zero length I/O
    move.l  IO_DATA(a2),a1  ; Point to data buffer
;
;A0=ramdisk index
;A1=user buffer
;D0=length
;
    cmp.b   #CMD_READ,IO_COMMAND+1(a2)  ; Decide on direction
    BEQ.S   CopyTheBlock
    EXG     A0,A1       ; For Write and Format, swap source & dest
CopyTheBlock:
    LINKSYS CopyMemQuick,md_SysLib(a6)  ;A0=source A1=dest D0=size
    ;CopyMemQuick is very fast

RdWrt_end:
    move.l  a2,a1
    movem.l (sp)+,a2/a3
    bra     TermIO  ;END



IO_LenErr:
    PUTMSG  10,<'bad length'>
    move.b  #IOERR_BADLENGTH,IO_ERROR(a2)
IO_End:
    clr.l   IO_ACTUAL(a2)   ;Initially, no data moved
    bra.s   RdWrt_end



;
; the Stop command stop all future io requests from being
; processed until a Start command is received.  The Stop
; command is NOT stackable: e.g. no matter how many stops
; have been issued, it only takes one Start to restart
; processing.
;
;Stop is rather silly for a ramdisk
MyStop:
   PUTMSG   30,<'%s/MyStop: called'>
   bset   #MDUB_STOPPED,UNIT_FLAGS(a3)
   bra   TermIO


Start:
    PUTMSG   30,<'%s/Start: called'>
    bsr.s  InternalStart
    bra   TermIO

       ;[A3=unit A6=device]
InternalStart:
    move.l  a1,-(sp)
    ;------ turn processing back on
    bclr   #MDUB_STOPPED,UNIT_FLAGS(a3)
    ;------ kick the task to start it moving
    move.b  MP_SIGBIT(a3),d1
    CLEAR   d0
    bset    d1,d0           ;prepared signal mask
    move.l  MP_SIGTASK(a3),a1       ;:FIXED:marco-task to signal
    LINKSYS Signal,md_SysLib(a6)    ;:FIXED:marco-a6 not a3
    move.l  (sp)+,a1
    rts


;
; Flush pulls all I/O requests off the queue and sends them back.
; We must be careful not to destroy work in progress, and also
; that we do not let some io requests slip by.
;
; Some funny magic goes on with the STOPPED bit in here.  Stop is
; defined as not being reentrant.  We therefore save the old state
; of the bit and then restore it later.  This keeps us from
; needing to DISABLE in flush.  It also fails miserably if someone
; does a start in the middle of a flush. (A semaphore might help...)
;

Flush:
   PUTMSG   30,<'%s/Flush: called'>
   movem.l   d2/a1/a6,-(sp)

   move.l   md_SysLib(a6),a6

   bset   #MDUB_STOPPED,UNIT_FLAGS(a3)
   sne   d2

Flush_Loop:
   move.l   a3,a0
   CALLSYS   GetMsg ;Steal messages from task's port

   tst.l   d0
   beq.s   Flush_End

   move.l   d0,a1
   move.b   #IOERR_ABORTED,IO_ERROR(a1)
   CALLSYS   ReplyMsg

   bra.s   Flush_Loop

Flush_End:
   move.l   d2,d0
   movem.l   (sp)+,d2/a1/a6

   tst.b   d0
   beq.s   1$

   bsr   InternalStart
1$:
   bra     TermIO


*****************************************************************************
;
; Here begins the task related routines
;
; A Task is provided so that queued requests may be processed at
; a later time.  This is not very justifiable for a ram disk, but
; is very useful for "real" hardware devices.  Take care with
; your arbitration of shared hardware with all the multitasking
; programs that might call you at once.
;
; Register Usage
; ==============
; a3 -- unit pointer
; a6 -- syslib pointer
; a5 -- device pointer
; a4 -- task (NOT process) pointer
; d7 -- wait mask
;----------------------------------------------------------------------

; some dos magic, useful for Processes (not us).  A process is started at
; the first  executable address  after a segment list.  We hand craft a
; segment list here.  See the the DOS technical reference if you really
; need to know more about this.
; The next instruction after the segment list is the first executable address

    cnop    0,4     ; long word align

    DC.L    16      ; segment length -- any number will do (this is 4
            ; bytes back from the segment pointer)
myproc_seglist:
    DC.L    0       ; pointer to next segment

Task_Begin:
    PUTMSG  35,<'%s/Task_Begin'>
    move.l  ABSEXECBASE,a6

    ;------ Grab the argument passed down from our parent
    move.l  4(sp),a3           ; Unit pointer
    move.l  mdu_Device(a3),a5  ; Point to device structure

    IFD   INTRRUPT
     ;------ Allocate a signal for "I/O Complete" interrupts
     moveq   #-1,d0     ; -1 is any signal at all
     CALLSYS   AllocSignal
     move.b   d0,mdu_SigBit(A3)   ; Save in unit structure
     moveq   #0,d7     ; Convert bit number signal mask
     bset   d0,d7
     move.l   d7,mdu_SigMask(A3)   ; Save in unit structure
     lea.l   mdu_is(a3),a1  ; Point to interrupt structure
     moveq   #INTB_PORTS,d0 ; Portia interrupt bit 3
     CALLSYS AddIntServer   ; Now install the server
     move.l   md_Base(a5),a0      ; Get board base address
*    bset.b   #INTENABLE,INTCTRL2(a0)   ; Enable interrupts
    ENDC

    ;------ Allocate a signal
    moveq   #-1,d0      ; -1 is any signal at all
    CALLSYS AllocSignal
    move.b  d0,MP_SIGBIT(a3)
    move.b  #PA_SIGNAL,MP_FLAGS(a3) ;Make message port "live"
    ;------ change the bit number into a mask, and save in d7
    moveq   #0,d7   ;Clear D7
    bset    d0,d7

    IFGE INFO_LEVEL-40
     move.l  $114(a6),-(sp)
     move.l  a5,-(sp)
     move.l  a3,-(sp)
     move.l  d0,-(sp)
     PUTMSG  40,<'%s/Signal=%ld, Unit=%lx Device=%lx Task=%lx'>
     add.l   #4*4,sp
    ENDC

    bra.s   Task_StartHere

; OK, kids, we are done with initialization.  We now can start the main loop
; of the driver.  It goes like this.  Because we had the port marked
; PA_IGNORE for a while (in InitUnit) we jump to the getmsg code on entry.
; (The first message will probably be posted BEFORE our task gets a chance
; to run)
;------     wait for a message
;------     lock the device
;------     get a message.  If no message, unlock device and loop
;------     dispatch the message
;------     loop back to get a message

    ;------ no more messages.  back ourselves out.
Task_Unlock:
    and.b   #$ff&(~(UNITF_ACTIVE!UNITF_INTASK)),UNIT_FLAGS(a3)
    ;------ main loop: wait for a new message

Task_MainLoop:
    PUTMSG   75,<'%s/++Sleep'>
    move.l  d7,d0
    CALLSYS Wait
    IFGE INFO_LEVEL-5
    bchg.b  #1,$bfe001  ;Blink the power LED
    ENDC
Task_StartHere:
    PUTMSG   75,<'%s/++Wakeup'>
    ;------ see if we are stopped
    btst    #MDUB_STOPPED,UNIT_FLAGS(a3)
    bne.s   Task_MainLoop   ; device is stopped, ignore messages
    ;------ lock the device
    bset    #UNITB_ACTIVE,UNIT_FLAGS(a3)
    bne     Task_MainLoop   ; device in use (immediate command?)
```

```
   ;------ get the next request
Task_NextMessage:
    move.l  a3,a0
    CALLSYS GetMsg
    PUTMSG  1,<'%s/GotMsg'>
    tst.l   d0
    beq     Task_Unlock ; no message?

    ;------ do this request
    move.l  d0,a1
    exg     a5,a6   ; put device ptr in right place
    bsr     PerformIO
    exg     a5,a6   ; get syslib back in a6

    bra.s   Task_NextMessage

*****************************************************************************
;
; Here is a dummy interrupt handler, with some crucial components commented
; out.  If the IFD INTRRUPT is enabled, this code will cause the device to
; wait for a level two interrupt before it will process each request
; (pressing RETURN on the keyboard will do it).  This code is normally
; disabled, and must fake or omit certain operations since there  isn't
; really any hardware for this driver.  Similar code has been used
; successfully in other, "REAL" device drivers.
;

   IFD   INTRRUPT

;   A1 should be pointing to the unit structure upon entry! (IS_DATA)
myintr:
*      move.l   md_Base(a0),a0      ; point to board base address
*      btst.b   #IAMPULLING,INTCTRL1(a0);See if I'm interrupting
*      beq.s   myexnm         ; if not set, exit, not mine
*      move.b   #0,INTACK(a0)      ; toggle controller's int2 bit

;      ------ signal the task that an interrupt has occurred

    move.l  mdu_Device(a1),a0   ; Get device pointer
    move.l  mdu_SigMask(a1),d0
    lea.l   mdu_tcb(a1),a1
    move.l  md_SysLib(a0),a6   ; Get pointer to system
    CALLSYS Signal

;      now clear the zero condition code so that
;      the interrupt handler doesn't call the next
;      interrupt server.
;
*      moveq   #1,d0         clear zero flag
*      bra.s   myexit         now exit
;
;      this exit point sets the zero condition code
;      so the interrupt handler will try the next server
;      in the interrupt chain
;
myexnm      moveq   #0,d0     set zero condition code
;
myexit      rts
   ENDC


*****************************************************************************

mdu_Init:
;   ------ Initialize the device

    INITBYTE    MP_FLAGS,PA_IGNORE  ;Unit starts with a message port
    INITBYTE    LN_TYPE,NT_MSGPORT  ;
    INITLONG    LN_NAME,myName      ;
    INITLONG    mdu_tcb+LN_NAME,myName
    INITBYTE    mdu_tcb+LN_TYPE,NT_TASK
    INITBYTE    mdu_tcb+LN_PRI,5
    IFD   INTRRUPT
     INITBYTE    mdu_is+LN_PRI,4      ; Int priority 4
     INITLONG    mdu_is+IS_CODE,myintr  ; Interrupt routine addr
     INITLONG    mdu_is+LN_NAME,myName
    ENDC
    DC.W   0

 IFNE  AUTOMOUNT
mdn_Init:
*   ;------ Initialize packet for MakeDosNode

    INITLONG    mdn_execName,myName ; Address of driver name
    INITLONG    mdn_tableSize,12    ; # long words in AmigaDOS env.
    INITLONG    mdn_dName,$524d0000 ; Store 'RM' in name
    INITLONG    mdn_sizeBlock,SECTOR/4  ; # longwords in a block
    INITLONG    mdn_numHeads,1      ; RAM disk has only one "head"
    INITLONG    mdn_secsPerBlk,1    ; secs/logical block, must = "1"
    INITLONG    mdn_blkTrack,SECTORSPER ; secs/track (must be reasonable)
    INITLONG    mdn_resBlks,1       ; reserved blocks, MUST > 0!
    INITLONG    mdn_upperCyl,(RAMSIZE/BYTESPERTRACK)-1 ; upper cylinder
    INITLONG    mdn_numBuffers,1    ; # AmigaDOS buffers to start
    DC.W   0
 ENDC

;----------------------------------------------------------------------
; EndCode is a marker that shows the end of your code.  Make sure it does not
; span hunks, and is not before the rom tag!  It is ok to put it right after
; the rom tag -- that way you are always safe.  I put it here because it
; happens to be the "right" thing to do, and I know that it is safe in this
; case (this program has only a single code hunk).
;----------------------------------------------------------------------
EndCode:    END
```

Retrieved from "http://wiki.amigaos.net/w/index.php?title=Example_Device&oldid=2188"

Categories: Devices | Examples | Need Code review

- This page was last modified on 11 May 2012, at 01:26.