ros

COLLABORATORS			
	TITLE : ros		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		February 12, 2023	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

ros

1.1 ros.doc

ros.library

Overview

History

AddTimerInt()

ROSAddTimerInt()

ROSAllocAudio()

ROSAllocMem()

ROSAwakeSystem()

ROSChipsetCheck()

ROSCPUCheck()

ROSCPUClock()

ROSDeleteFile()

ROSDisableReq()

ROSEnableReq()

ROSFreeAudio()

ROSFreeMem()

ROSGetDMA()

ROSGetInt()

ROSGetKey()

ROSKillSystem()

ROSQueryKeys()

ROSReadExe()

ROSReadFile()

ROSRemTimerInt()

ROSRequester()

ROSScreenToBack()

ROSScreenToFront()

ROSSetCache()

ROSSetCopper()

ROSSetDMA()

ROSSetExitHandler()

ROSSetInt()

ROSSetIntVec()

ROSSetTimerVec()

ROSStartTimer()

ROSStopTimer()

ROSSysCall()

ROSSysCallEnd()

ROSWaitVBlank()

ROSWriteFile()

1.2 ros.library/Overview (information)

This section describes the way how to use the RETIRE Operating System within your own applications:

The ros.library is an enhanced lowlevel.library. It's compatible with currently all available machines and CPU's (upto '060) and OS1.3 compatible too!
The ROS provides additional functions to take over the entire system so it should be useful for demo & game coders.
Another advantage is the ability to call important system functions (memory allocation, read, write) from a killed system.

ROS offers a way to run your hardware bashing application within multitasking.
And finally a ROS prefs file is provided to support custom settings.
On startup ROS tries to load the ROS.prefs from your ENV: directory. You may edit this file to configure the library for your needs.

But remember: It's impossible to open the ros.library from two applications at the same time. The second application will fail to open the library!

Programming guidelines:

The ros.library is a standard Amiga library so normal restrictions applies to function calls:
- a6 loaded with _ROSBase
- d0/d1/a0/a1 scratch registers (destroyed by libcall)
- functions can't be called from within interrupts

Some ROS functions are special in these cases; they need no scratch registers or they may be called from within interrupts. Look at the appropriate 'note' comment for further information.

Do not call any ROS function from supervisor mode. The machine will crash!!!
I think there is no need to switch to supervisor mode. Think of the '060 CPU; its supervisor model differs from the '040 CPU and a few basic commands will be emulated by software. Executing such a command from supervisor mode may crash the machine!!!

Call ROSKillSystem(ROSKSF_DEATHMODE) to take over the entire system. But do not modify DMA & interrupt stuff by yourself! Use the appropriate ROS routines!!!!!
ROSKillSystem() opens a PAL-LowRes screen so all 'special' AGA registers are flushed (FMODE may not!?!) and it forces graphics boards to switch to the Amiga custom chip display.

After taking over the system you may use the blitter. You should finish your blits before calling ROS functions which may invoke system calls. After the function returns you may bash the blitter again.
Remember: Use a WaitBlit before freeing some chip memory!!!

In a multitasking environment your vblank code may be called after the display is started depending on the speed of higher prioritized vblank interrupts.
If you want to synchronize custom chip modifications with the display you should use the copper.
And remember: The vblank frequency may vary (not always 50 Hz) if ROS screen behind other screens!

The registers you may trash:
- the display custom chip registers after taking over the system, in multi-tasking mode only if the ROS screen is the frontmost
- the audio custom chip registers (if channels allocated)
- the blitter registers (only if taken over the system)
- the CIA registers (if successfully allocated)

You should never touch:
- DMACON and INTENA registers
- CIA ICR registers
- CPU interrupt vectors
- serial registers
- disk related registers

Please read the following autodocs carefully!

```
If you have questions, problems, idea's or bug reports feel free to contact me:

TIK/RETIRE via email:  Nico.Schmidt@Student.Uni-Magdeburg.de

or write to:           Nico Schmidt
                       Möhlauer Str. 9
                       06773 Jüdenberg
                       GERMANY

Thanks to:  TODI/RETIRE for hints and useful idea's
            PABLO/RETIRE for beta testing
```

1.3 ros.library/History (information)

```
                                ROS HISTORY

******************************************************************************
RELEASE 1.0, 29.12.95 (Library version 1.0)

    The first release.

******************************************************************************
RELEASE 1.1, 05.01.96 (Library version 1.1)

    Fixed two minor bugs:
      - ROSCPUClock() shouldn't display a requester if no free CIA available,
        but it does
      - user vblank code was called from vblank server with priority 9, this
        may confuse the timer.device (priority 0) and thus all CIA interrupts
        (e.g. level6 p61) if it steals to much rastertime;
        now user vblank code is executed from a priority 0 vblank server

******************************************************************************
RELEASE 1.2, 08.01.96 (Library version 1.2)

    Library name changed from 'ROS.library' to 'ros.library' because a library
    name should always be a lower-case string (thanks to Thomas Kessler)

******************************************************************************
```

1.4 ros.library/ROSAddTimerInt()

```
                NAME
      ROSAddTimerInt -- add an interrupt that is executed at regular
                        intervals

   SYNOPSIS
      intHandle = ROSAddTimerInt(Flags, intRoutine);
      D0                          D0     A0

      ULONG AddTimerInt(ULONG, APTR);
```

```
FUNCTION
     Calling this routine causes the system to allocate a CIA timer
     depending on 'Flags' and set up 'intRoutine' to service any interrupts
     caused by the timer.
     Although the timer is allocated it is neither running, nor enabled.

             ROSStartTimer()
              must be called to establish the time interval and
     start the timer.

     The routine is called from within an interrupt, so normal
     restrictions apply. On entry A5 holds the custom base ($dff000) and
     A6 holds _ROSBase. Your code may trash all registers.

     The CIA timer used by this routine is not guaranteed to always be
     the same. This routine utilizes the CIA resource and uses an
     unallocated CIA timer.

     You should remove the timer with
             ROSRemTimerInt()
             .
     Closing the ros.library removes this timer interrupt automatically.

INPUTS
     Flags - specify the timer to allocate:
                     TIMF_ANY  - try to allocate any timer
                     TIMF_LEV2 - try to allocate level 2 timers (CIA A)
                     TIMF_LEV6 - try to allocate level 6 timers (CIA B)
     intRoutine - the routine to invoke upon timer interrupts.

RESULT
     intHandle - a handle used to manipulate the interrupt, or NULL
                 if it was not possible to attach the routine.

NOTE
     If allocation failed a Retry/Cancel-Requester appears to inform the
     user about the problem (if not suppressed and if not in Death-Mode).

     This function interrupts the Death-Mode state with
             ROSSysCall()
             !

SEE ALSO

             ROSRemTimerInt()
             ,
             ROSStartTimer()
             ,
             ROSStopTimer()
             ,
             ROSSetTimerVec()
```

1.5 ros.library/ROSAllocAudio()

```
                NAME
     ROSAllocAudio -- allocate audio channels

SYNOPSIS
     success = ROSAllocAudio()
     D0

     BOOL ROSAllocAudio(VOID);

FUNCTION
     This function allocates all audio channels. Audio DMA is updated
     so that matches with current DMA settings (former
             ROSSetDMA()
              calls).

     You should free the audio channels with
             ROSFreeAudio()
             .
     Closing the ros.library frees audio channels automatically.

RESULT
     success - boolean

NOTE
     If allocation failed a Retry/Cancel-Requester appears to inform the
     user about the problem (if not suppressed and if not in Death-Mode).

     This function interrupts the Death-Mode state with
             ROSSysCall()
             !

SEE ALSO

             ROSFreeAudio()
```

1.6 ros.library/ROSAllocMem()

```
                NAME
     ROSAllocMem -- allocate memory and keep track of the size

SYNOPSIS
     memoryBlock = ROSAllocMem(byteSize, attributes, alignmask)
     D0                        D0        D1          D2

     VOID *ROSAllocMem(ULONG, ULONG, ULONG);

FUNCTION
     This function works identically to Exec's AllocMem(), but tracks the
     size of the allocation and provides additional memory alignment.

     See the AllocMem() documentation for details.

     You should free the memory with
```

```
            ROSFreeMem()
            .
     Do _not_ use exec.library functions!!!!!!!
     Closing the ros.library frees allocated memory automatically.

INPUTS
     byteSize - # of bytes to allocate
     attributes - memory requirements
     alignmask - used to allocate memory at special boundaries, if NULL
                 Exec's standard alignment applies

                 --> newaddr = memaddr AND NOT(alignmask)
                 e.g. mem align: even by 8 bytes -> alignmask = $00000007

RESULT
     memoryBlock - a pointer to the newly allocated memory block or NULL

NOTE
     This function interrupts the Death-Mode state with
            ROSSysCall()
            !

SEE ALSO

            ROSFreeMem()
            , exec.lib/AllocMem()
```

1.7 ros.library/ROSAwakeSystem()

```
                NAME
     ROSAwakeSystem -- close ROS screen and awake from Death-Mode

SYNOPSIS
     ROSAwakeSystem()

     VOID ROSAwakeSystem(VOID);

FUNCTION
     This function deallocates all things allocated with
            ROSKillSystem()
            .
     It closes the opened screen and sets the display related DMA to the
     appropriate system values.
     Calling this function without a matching
            ROSKillSystem()
             is harmless.

     Closing the ros.library calls automatically ROSAwakeSystem().

SEE ALSO

            ROSKillSystem()
```

1.8 ros.library/ROSChipsetCheck()

```
NAME
     ROSChipsetCheck -- check if desired custom chipset available

SYNOPSIS
     result = ROSChipsetCheck(chipsetbit)
     D0                       D0

     UWORD ROSChipsetCheck(UWORD);

FUNCTION
     Use this routine to obtain the available chipset.
     As input you may specify the chipset requirement (bitnumber) and if
     _not_ found a requester appears to inform the user about his lack of
     hardware (if not suppressed and if not in Death-Mode).

INPUTS
     chipsetbit - the required chipset; currently defined are
                      ROSCSB_ECS for at least HR-AGNUS rev3 and 8373 DENISE
                      ROSCSB_AGA for at least ALICE rev2 and LISA,
                  or -1 for only obtaining chipset (no requester appears)

RESULT
     result - chipset flags (ROSCF_xxx, AGA includes the ECS flag) or NULL
              if required chipset not found
```

1.9 ros.library/ROSCPUCheck()

```
NAME
     ROSCPUCheck -- check if desired CPU/FPU available

SYNOPSIS
     result = ROSCPUCheck(cpubit)
     D0                   D0

     UWORD ROSCPUCheck(UWORD);

FUNCTION
     Use this routine to obtain the available CPU/FPU.
     As input you may specify the processor requirement (bitnumber) and if
     _not_ found a requester appears to inform the user about his hardware
     problem (if not suppressed and if not in Death-Mode).
     Call this routine twice if you need a CPU (AFB_680x0) check and a FPU
     (AFB_6888x/FPU40) check.

INPUTS
     cpubit - the required CPU/FPU -> AttnFlags AFB_xxxxx
              or -1 for only obtaining CPU/FPU (no requester appears)

RESULT
     result - AttnFlags or NULL if required chipset not found
```

```
NOTE
     This code handles the FPU40 flag correct: it's only valid if AFB_68040
     set. Keep this in mind if you examine the RESULT (AttnFlags)!!!
```

1.10 ros.library/ROSCPUClock()

```
                NAME
     ROSCPUClock -- CPU/FPU clock calculation

SYNOPSIS
     CPU/FPUclock = ROSCPUClock()
     D0  D1

     ULONG/ULONG ROSCPUClock(VOID);

FUNCTION
     Calculate CPU/FPU clock. The values returned should be accurate at
     least 1/10 MHz, so it's enough for everyone.
     This routine works with all (currently) available processors and it
     takes only a few milliseconds!!!
     In some cases the returned clock is NULL because of the lack of
     free CIA timers (or other problems, see BUGS) so it's best to call
     this routine first (e.g. before any timer function).

RESULT
     CPUclock - the CPU clock in kHz or NULL on error (see also BUGS)
     FPUclock - the FPU clock in kHz or NULL on error

NOTE
     This function interrupts the Death-Mode state with
             ROSSysCall()
             !

BUGS
     In some rare cases the function returns NULL:
      - on 68000/10 with only chipmem
      - on _really_ slow machines (e.g. a 10MHz 68040?!?)
      - on 68040/60 systems where it's impossible to turn on instruction
        cache (e.g. A4000 with only chipmem and 68040.library loaded)

     The FPU clock is _always_ NULL for 68881/2 systems because I'm
     (currently) unable to find a reliable algorithm (blame on me).
     It's hard to deal with the asynchronous data transfer between CPU
     and FPU.
     On 68040/60 systems with FPU the returned value is the same like
     CPU clock.
```

1.11 ros.library/ROSDeleteFile()

```
                NAME
     ROSDeleteFile -- Delete a file or directory

SYNOPSIS
     success = ROSDeleteFile(flags, name)
     D0                      D0     A0

     ULONG ROSDeleteFile(ULONG, STRPTR);

FUNCTION
     This attempts to delete the file or directory specified by 'name'.
     Note that all the files within a directory must be deleted before the
     directory itself can be deleted.

     If ROSRWF_DISKWAIT flag set the routine assumes that you want to delete
     'name' from a floppy. This means:
      - waiting for the disk to be inserted (can't be cancelled!!!)
        (implementation: DOS error codes DEVICE_NOT_MOUNTED, NOT_A_DOS_DISK,
        NO_DISK force retry until any other error occurs)
      - waiting until disk drive stopped
        (implementation: polling disk.resource until drives are ready)

     Since OS2.0 it's possible to delete files from your program path
     instead of the current directory using PROGDIR:xxx.
     On OS1.3 the "PROGDIR:" string is automatically removed by ROS so you
     don't have to worry about it. The only thing the user have to do is a
     "CD" to the program directory before starting (only on OS1.3). ;-)

     WARNING: It may be dangerous to use this routine within Death-Mode!
              Because of the AMIGA's multitasking facilities it isn't
              possible (isn't it???) to wait till the final end of a delete
              operation. That's why I use a simple delay of 2 seconds after
              any delete operation in Death-Mode. If your filesystem is to
              slow to write all the data within this time your partition
              will be _not_ validated!!!
              You've been warned!!!

INPUTS
     flags - use ROSRWF_DISKWAIT if you want to delete from a floppy
     name - pointer to a null-terminated string

RESULT
     success - DOS error code on failure or NULL for success

NOTE
     This function interrupts the Death-Mode state with
             ROSSysCall()
             !

SEE ALSO
     DOS.lib/DeleteFile()
```

1.12 ros.library/ROSDisableReq()

```
                NAME
     ROSDisableReq -- suspend requesters for our task

SYNOPSIS
     ROSDisableReq()

     VOID ROSDisableReq(VOID);

FUNCTION
     Suspend all upcoming requesters for our task (ROS- and DOS-Requesters).
     By default requesters are enabled.

SEE ALSO

             ROSEnableReq()
             ,
             ROSRequester()
```

1.13 ros.library/ROSEnableReq()

```
                NAME
     ROSEnableReq -- permit requesters for our task

SYNOPSIS
     ROSEnableReq()

     VOID ROSEnableReq(VOID);

FUNCTION
     Permit all upcoming requesters for our task (ROS- and DOS-Requesters).
     By default requesters are enabled.

SEE ALSO

             ROSDisableReq()
             ,
             ROSRequester()
```

1.14 ros.library/ROSFreeAudio()

```
                NAME
     ROSFreeAudio -- free audio channels

SYNOPSIS
     ROSFreeAudio()

     VOID ROSFreeAudio(VOID);
```

```
FUNCTION
     This function removes probably installed audio interrupt handlers and
     resets the audio DMA (former system values). Then channels are freed.

     Freeing channels without allocation is harmless.

     Closing the ros.library frees audio channels automatically.

NOTE
     This function interrupts the Death-Mode state with
             ROSSysCall()
             !

SEE ALSO

             ROSAllocAudio()
```

1.15 ros.library/ROSFreeMem()

```
                NAME
     ROSFreeMem -- return allocated memory to the system

SYNOPSIS
     ROSFreeMem(memoryBlock)
                A0

     VOID ROSFreeMem(VOID *);

FUNCTION
     Free an allocation made by the
             ROSAllocMem()
              call. The memory will be
     returned to the system pool from which it came.

     WARNING: Before freeing some chip memory be sure that the blitter has
              finished its work within this memory block!!!
              The same applies to closing the ros.library.

     Closing the ros.library frees allocated memory automatically.

NOTE
     Passing a wrong pointer is harmless ;-)

     This function interrupts the Death-Mode state with
             ROSSysCall()
             !

INPUTS
     memoryBlock - pointer to the memory block to free, or NULL

SEE ALSO

             ROSAllocMem()
```

1.16 ros.library/ROSGetDMA()

```
                NAME
     ROSGetDMA -- obtain DMA settings

SYNOPSIS
     DMACONR = ROSGetDMA()
     D0

     UWORD ROSGetDMA(VOID);

FUNCTION
     This function returns current DMA settings (set via
             ROSSetDMA()
             ).

     Do _not_ use "move.w $dff002,d0" !!!!!!!
     This may return wrong bits if you are in Non-Death-Mode (e.g. copper
     DMA -> the OS may change the bit during screen switching).
     But ROSGetDMA() returns the right value.

     Closing the ros.library reset former system DMA.

RESULT
     DMACONR - DMA settings set via
             ROSSetDMA()
              with additional BBUSY/BZERO
               bits (unsupported (disk) & unallocated (audio, ...) DMA bits
               are always 0)

NOTE
     This call is guaranteed to preserve all other registers and is save
     to call from interrupts.

SEE ALSO

             ROSSetDMA()
```

1.17 ros.library/ROSGetInt()

```
                NAME
     ROSGetInt -- obtain interrupt settings

SYNOPSIS
     INTENAR = ROSGetInt()
     D0

     UWORD ROSGetInt(VOID);
```

```
FUNCTION
     This function returns interrupt enable bits (set via
             ROSSetDMA()
             ).
     Additional bits are defined in "ros.i".

     Do _not_ use "move.w $dff01c,d0" !!!!!!! This may return wrong bits.

     Closing the ros.library reset former system interrupt bits.

RESULT
     INTENAR - Interrupt bits set via
             ROSSetInt()
                                Note: level 1/2/5/6 bits (disk, serial, timer)  ↩
                                   are not
                     supported!!!
                     Instead additonal bits are defined:  - KEYB
                                                          - CIAATIMA/B
                                                          - CIABTIMA/B

NOTE
     This call is guaranteed to preserve all other registers and is save
     to call from interrupts.

SEE ALSO

             ROSSetInt()
             , libraries/ros.i
```

1.18 ros.library/ROSGetKey()

```
NAME
     ROSGetKey -- returns the currently pressed rawkey code and qualifiers

SYNOPSIS
     key = ROSGetKey()
     D0

     ULONG ROSGetKey(VOID);

FUNCTION
     This function returns the currently pressed non-qualifier key and
     all pressed qualifiers.

     The values returned by this function are not modified by which
     window/screen currently has input focus.

RESULT
     key - key code for the last non-qualifier key pressed in the low
           order word ($0..$67). If no key is pressed this word will be $ff.
           The upper order word contains the qualifiers which can be found
           within the long word as follows (lowlevel.library like):
                     Qualifier               Key
```

```
                    LLKB_LSHIFT             Left Shift
                    LLKB_RSHIFT             Rigt Shift
                    LLKB_CAPSLOCK           Caps Lock
                    LLKB_CONTROL            Control
                    LLKB_LALT               Left Alt
                    LLKB_RALT               Right Alt
                    LLKB_LAMIGA             Left Amiga
                    LLKB_RAMIGA             Right Amiga

NOTE
     This call is guaranteed to preserve all other registers and is save
     to call from interrupts.

SEE ALSO
     lowlevel.lib/GetKey(), libraries/lowlevel.i
```

1.19 ros.library/ROSKillSystem()

```
               NAME
     ROSKillSystem -- kill the system and open a PAL-LowRes screen

SYNOPSIS
     success = ROSKillSystem(mode)
     D0                      D0

     BOOL ROSKillSystem(ULONG);

FUNCTION
     This function opens a PAL-LowRes screen, installs your copper list and
     sets additional DMA (copper, bitplane, sprite).
     By default all DMA will be cleared.
     Whenever this screen is the frontmost you may bash all display related
     custom chip registers (preferably via copper list).
     If the ROS is unable to open the screen or if it's not PAL-LowRes
     (promoted screen) a Retry/Cancel-Requester appears (if not suppressed).

     If you specify the flag KILLF_DEATHMODE the multitasking will be turned
     off. Now ROS has taken over the entire system. But interrupts and DMA
     should only be set with ROS functions!!!
     The blitter is owned via OwnBlitter() so you may use it. But before
     using you should do a WaitBlit because it may still working.
     You may interrupt the Death-Mode via
             ROSSysCall()
             . At this time the
     blitter will be disowned so you do _not_ use it during a SysCall!!!

     Call ROSKillSystem() with KILLF_SYSMODE to run your application within
     multitasking. But be polite and bash the hardware as less as possible.
     The blitter is under system control so you should use OwnBlitter()/
     DisownBlitter() if you need it. But allocate the blitter only for a
     short time because the system uses it too!

     Also check out the ROS_KillSysFlags (the current ROS state).
     You may call ROSKillSystem() multiple times to switch between Death-
     and SysMode.
```

```
     A switch from Sys- into Death-Mode automatically makes the ROS screen
     the frontmost so you don't have to call
             ROSScreenToFront()
             .

     You should return to the system with
             ROSAwakeSystem()
             .
     Closing the ros.library calls
             ROSAwakeSystem()
              automatically.

INPUTS
     mode - KILLF_SYSMODE for a multitasking environment
            KILLF_DEATHMODE to take over the entire system

RESULT
     success - FALSE if ROS was unable to open the screen

SEE ALSO

             ROSAwakeSystem()
             ,
             ROSSysCall()
             ,
             ROSSysCallEnd()
```

1.20 ros.library/ROSQueryKeys()

```
NAME
     ROSQueryKeys -- return the states for a set of keys

SYNOPSIS
     ROSQueryKeys(queryArray, arraySize)
                  A0          D0

     VOID ROSQueryKeys(struct KeyQuery *, UBYTE);

FUNCTION
     Scans the keyboard to determine which of the rawkey codes
     listed in the 'queryArray' are currently pressed. The state for each
     key is returned in the array.

     The values returned by this function are not modified by which
     window/screen currently has input focus.

INPUTS
     queryArray - an array of KeyQuery structures. The kq_KeyCode fields
                  of these structures should be filled with the rawkey
                  codes you wish to query about. Upon return from this
                  function, the kq_Pressed field of these structures
                  will be set to TRUE if the associated key is down,
                  and FALSE if not.
```

```
     arraySize - number of key code entries in 'queryArray'

NOTE
     This function is save to call from interrupts.

SEE ALSO
     lowlevel.lib/QueryKeys(), libraries/lowlevel.i
```

1.21 ros.library/ROSReadExe()

```
               NAME
     ROSReadExe -- scatterload a loadable file

SYNOPSIS
     seglist/errorcode = ROSReadExe(flags, name)
     D0      D1                     D0     A0

     BPTR/ULONG ROSReadExe(ULONG, STRPTR);

FUNCTION
     This function tries to scatterload the file 'name' using DOS.library's
     LoadSeg().

     If ROSRWF_DISKWAIT flag set the routine assumes that you want to load
     'name' from a floppy. This means:
      - waiting for the disk to be inserted (can't be cancelled!!!)
        (implementation: DOS error codes DEVICE_NOT_MOUNTED, NOT_A_DOS_DISK,
        NO_DISK force retry until any other error occurs)
      - waiting until disk drive stopped
        (implementation: polling disk.resource until drives are ready)

     Since OS2.0 it's possible to load files from your program path instead
     of the current directory using PROGDIR:xxx.
     On OS1.3 the "PROGDIR:" string is automatically removed by ROS so you
     don't have to worry about it. The only thing the user have to do is a
     "CD" to the program directory before starting (only on OS1.3). ;-)

     You should unload the file with
             ROSFreeMem
             (seglist)!
     Do _not_ use DOS.library's UnLoadSeg()!!!!!!!
     Closing the ros.library unloads file automatically.

INPUTS
     flags - use ROSRWF_DISKWAIT if you want to load from a floppy
     name - pointer to a null-terminated string

RESULT
     seglist - BCPL pointer to a seglist or NULL if error
     errorcode - DOS error code on failure (may be NULL!!!)

NOTE
     This function interrupts the Death-Mode state with
             ROSSysCall()
             !
```

```
SEE ALSO

             ROSFreeMem()
             , DOS.lib/LoadSeg()
```

1.22 ros.library/ROSReadFile()

```
                NAME
     ROSReadFile -- read a number bytes from a file

SYNOPSIS
     actLength/errorcode/actBuffer = ROSReadFile(flags, length, offset,
     D0        D1        A0                      D0     D1      D2
                                              memtype, memalign, name, buffer)
                                              D3       D4        A0    A1

     ULONG/ULONG/VOID *  ROSReadFile(ULONG, ULONG, ULONG,
                                       ULONG, ULONG, STRPTR, VOID *);

FUNCTION
     This function tries to load 'length' bytes from the file 'name' using
     DOS.library's Read(). Set 'length` = 0 to load the whole file.
     A1 points to a buffer where the data should be placed, or set A1=0
     to force an automatic memory allocation. In this case you must specify
     'memtype' and 'memalign' (see also
             ROSAllocMem()
             ).

     If ROSRWF_DISKWAIT flag set the routine assumes that you want to load
     'name' from a floppy. This means:
      - waiting for the disk to be inserted (can't be cancelled!!!)
        (implementation: DOS error codes DEVICE_NOT_MOUNTED, NOT_A_DOS_DISK,
        NO_DISK force retry until any other error occurs)
      - waiting until disk drive stopped
        (implementation: polling disk.resource until drives are ready)

     Since OS2.0 it's possible to load files from your program path instead
     of the current directory using PROGDIR:xxx.
     On OS1.3 the "PROGDIR:" string is automatically removed by ROS so you
     don't have to worry about it. The only thing the user have to do is a
     "CD" to the program directory before starting (only on OS1.3). ;-)

     You should free the automatically allocated memory with
             ROSFreeMem()
             !
     Closing the ros.library unloads file automatically (if automatic
     memory allocation).

INPUTS
     flags - use ROSRWF_DISKWAIT if you want to load from a floppy
     length - number of bytes to load or NULL for the whole file
     offset - number of bytes to skip before reading from file
     memtype - see
```

```
             ROSAllocMem()
             , only neccessary if 'buffer' == NULL
     memalign - see
             ROSAllocMem()
             , only neccessary if 'buffer' == NULL
     name - pointer to a null-terminated string
     buffer - pointer to buffer or NULL to force an automitic memory
              allocation

RESULT
     actLength - number of bytes read from file or -1 if error
     errorcode - DOS error code on failure (may be NULL!!!)
     actBuffer - points to loaded data, only valid on success (D0 <> -1)!!!

NOTE
     This function interrupts the Death-Mode state with
             ROSSysCall()
             !

SEE ALSO

             ROSAllocMem()
             ,
             ROSFreeMem()
             , DOS.lib/Read()
```

1.23 ros.library/ROSRemTimerInt()

```
                NAME
     ROSRemTimerInt -- remove a previously installed timer interrupt

SYNOPSIS
     ROSRemTimerInt(intHandle);
                    A1

     VOID
             AddTimerInt
             (ULONG);

FUNCTION
     Removes a timer interrupt routine previously installed with

             ROSAddTimerInt()
             .

     Closing the ros.library removes this timer interrupt automatically.

INPUTS
     intHandle - handle obtained from
             ROSAddTimerInt()
             , if NULL nothing
                 happens

NOTE
```

```
    This function interrupts the Death-Mode state with
            ROSSysCall()
            !

SEE ALSO

            ROSAddTimerInt()
            ,
            ROSStartTimer()
            ,
            ROSStopTimer()
            ,
            ROSSetTimerVec()
```

1.24 ros.library/ROSRequester()

```
                NAME
    ROSRequester -- Display a requester

SYNOPSIS
    result = ROSRequester(TextFormat, ArgList, GadgetFormat)
    D0                    A0          A1       A2

    LONG ROSRequester(STRPTR, APTR, STRPTR)

FUNCTION
    This procedure automatically builds a requester for you using
    intuition's EasyRequestArgs() (AutoRequest() on OS1.3) and then
    waits for a response from the user.

    The requester appears only if it is not disabled via
            ROSDisableReq()
                    or your tasks pr_WindowPtr = -1.
    Do not set this pointer directly. Use instead ROSDisable()/ROSEnable().
    By default requesters are enabled.

    In Death-Mode requesters are always disabled.
    If requesters are disabled the 'result' will be always FALSE (NULL).

INPUTS
    TextFormat - Format string, a la RawDoFmt(), for message in
        requester body. Lines are separated by the newline character ($0A).
        Formatting '%' functions are supported exactly as in RawDoFmt().
        Note: on OS1.3 max 5 textlines
    GadgetFormat - Format string for gadgets.  Text for separate
        gadgets is separated by '|'. You MUST specify at least one gadget.
        Note: max 2 gadgets on OS1.3
              gadget format functions only supported on OS2.0+.
    argList - Arguments for format commands. Arguments for gadgets
              follow arguments for bodytext

RESULT
    0, 1, ..., N - Successive GadgetID values, for the gadgets
                   you specify for the requester.  NOTE: The numbering
```

```
                    from left to right is actually: 1, 2, ..., N, 0.
                    This is for compatibility with AutoRequest(), which has
                    FALSE for the rightmost gadget.

BUGS
    Do not rely on the 'result' of an one-button requester!
    Due a bug in intuitions AutoRequest()/EasyRequestArgs() the 'result'
    may be TRUE (1) or FALSE (0) because it's possible to satisfy an one-
    button requester with both short-cuts: AMIGA-B and AMIGA-V.

SEE ALSO
    intuition.lib/AutoRequest(), intuition.lib/EasyRequestArgs(),
    exec.lib/RawDoFmt(),
            ROSDisableReq()
            ,
            ROSEnableReq()
```

1.25 ros.library/ROSScreenToBack()

```
                NAME
    ROSScreenToBack -- Send ROS screen to the back of the display

SYNOPSIS
    ROSScreenToBack()

    VOID ROSScreenToBack(VOID);

FUNCTION
    Sends the ROS screen to the back of the display if it is open and
    if _not_ in Death-Mode.

SEE ALSO

            ROSScreenToFront()
```

1.26 ros.library/ROSScreenToFront()

```
                NAME
    ROSScreenToFront -- Make ROS screen the frontmost

SYNOPSIS
    ROSScreenToFront()

    VOID ROSScreenToFront(VOID);

FUNCTION
    Brings the ROS screen to the front of the display if it is open and
    if _not_ in Death-Mode.

SEE ALSO
```

```
            ROSScreenToBack()
```

1.27 ros.library/ROSSetCache()

```
                NAME
     ROSSetCache - Instruction & data cache control

SYNOPSIS
     oldBits = ROSSetCache(cacheBits, cacheMask)
     D0                    D0         D1

     ULONG ROSSetCache(ULONG, ULONG);

FUNCTION
     This function provides global control of any instruction or data
     caches that may be connected to the system.  All settings are
     global (until ros.library closed) -- per task control is not provided.

     Closing the ros.library reset former system cache settings.

INPUTS
     cacheBits - new values for the bits specified in cacheMask.
                 use -1 to reset to system defaults (ignores cacheMask)
     cacheMask - a mask with ones for all bits to be changed

     Note: cacheBits/cacheMask may be overridden by values specified in
           ROS.prefs (to support individual cache settings)

RESULT
     oldBits - the complete prior values for all settings.

NOTE
     As a side effect, this function clears all caches.
     This function interrupts the Death-Mode state with
             ROSSysCall()
             !

SEE ALSO
     exec/execbase.i, exec.lib/CacheControl()
```

1.28 ros.library/ROSSetCopper()

```
                NAME
     ROSSetCopper -- install copperlist for ROS screen

SYNOPSIS
     ROSSetCopper(flags, coplist)
                  D0     A0
```

```
     VOID ROSSetCopper(ULONG, APTR);

FUNCTION
     This function installs a new copper list for our ROS screen.
     Whenever this screen is the frontmost this copper list becomes the
     active.

     Do _not_ use "move.l #mycoplist,$dff080" or "move.w #0,$dff088" !!!!!!!
     This may crash the display if you are in Non-Death-Mode and the ROS
     screen is not yet available or behind others.
     ROSSetCopper() takes this into account and displays this copper list as
     soon as possible.

     A screen switch forces always copper1 starting in a long frame!

INPUTS
     flags - COPF_COPPER1/2: modify copper1 or copper2
                 Remember: copper1 is always started during vertical blank
             COPF_STROBE: force an immediate copper start, move.w #0,$dff088
                 set 'coplist' to NULL for only strobe
             COPF_LOF/SHF: forces a long/short frame while starting copper
                 (to display interlaced screens correctly)
     coplist - copper list pointer, if NULL the old copper pointer remains

NOTE
     This call is guaranteed to preserve all registers (except D0!!!) and is
     save to call from interrupts.

SEE ALSO
     libraries/ros.i,
             ROSKillSystem()
```

1.29 ros.library/ROSSetDMA()

```
                NAME
     ROSSetDMA -- modify DMA bits

SYNOPSIS
     ROSSetDMA(DMACON)
               D0

     ROSSetDMA(UWORD);

FUNCTION
     This function acts like a write to the DMACON register.

     Do _not_ use "move.w #xxx,$dff096" !!!!!!!
     This may interfere with the system you are in Non-Death-Mode, but
     ROSSetDMA() handles all exceptions correctly.

     DMA bit modification:
       - DMAF_BLTPRI: only in Death-Mode, but during a
             ROSSysCall()
              the
```

```
                    system DMA bit will be restored
      - DMAF_DMAEN: same like BLTPRI
      - DMAF_BPLEN: only if ROS screen in front
      - DMAF_COPEN: same like BPLEN
      - DMAF_BLTEN: same like BLTPRI
      - DMAF_SPREN: same like BPLEN
      - DMAF_DSKEN: never modified!!!
      - DMAF_AUDxEN: only if audio channels allocated

    A currently forbidden DMA modification is automatically updated as
    soon as possible (e.g. bitplane DMA if ROS screen behind others).

    Closing the ros.library reset former system DMA.

INPUTS
    DMACON - DMA bits

NOTE
    This call is guaranteed to preserve all registers (except D0!!!) and is
    save to call from interrupts.

SEE ALSO

            ROSGetDMA()
            , hardware/dmabits.i
```

1.30 ros.library/ROSSetExitHandler()

```
NAME
    ROSSetExitHandler -- add handler that is executed at exit key condition

SYNOPSIS
    ROSSetExitHandler(handler);
                      A0

    ROSSetExitHandler(APTR);

FUNCTION
    Calling this routine causes the ROS to install an exit key conditon
    handler. Whenever this special key is pressed your handler is invoked
    (should be only once). Now it's time to finish your application.

    The routine is called from within an interrupt, so normal
    restrictions apply. On entry A5 holds the custom base ($dff000) and
    A6 holds _ROSBase. Your code may trash all registers.

    The default exit key code is SHIFT ESC but may be changed in ROS.prefs.

    You should remove this handler with ROSSetExitHandler(NULL).
    Closing the ros.library removes this timer interrupt automatically.

INPUTS
    handler - the routine to invoke upon exit key condition or NULL to
```

```
               detach the handler

NOTE
     This call is guaranteed to preserve all registers (except A0!!!) and is
     save to call from interrupts.

     The handler may be called more than once if your application isn't
     fast enough to exit!!!

SEE ALSO
```

1.31 ros.library/ROSSetInt()

```
                NAME
     ROSSetInt -- modify interrupt settings

SYNOPSIS
     ROSSetInt(INTENA)
               D0

     ROSSetInt(UWORD);

FUNCTION
     This function acts like a write to the INTENA register.

     Do _not_ use "move.w #xxx,$dff09a" !!!!!!!
     This may interfere with the system you are in Non-Death-Mode, but
     ROSSetInt() handles all exceptions correctly.

     Only those bits are affected which belongs to previous
             ROSSetIntVec()
                     calls. Look at
             ROSSetIntVec()
              for further information.

     In Non-Death-Mode the master interrupt bit (INTB_INTEN) is emulated to
     support enable/disable of all ROS interrupts at once (system interrupts
     are still active!!!)

     Closing the ros.library reset former system interrupt bits.

INPUTS
     INTENA - Int bits
              Note: level 1/2/5/6 bits (disk, serial, timer) are not
                    supported!!!
                    Instead additonal bits are defined:  - KEYB
                                                         - CIAATIMA/B
                                                         - CIABTIMA/B
              Look at
             ROSSetIntVec()
              for further details
NOTE
     This call is guaranteed to preserve all registers (except D0!!!).

     Do _not_ call this routine from interrupts if you want to modify the
```

```
     CIA interrupts (CIAATIMA/B, CIABTIMA/B)!!!!

SEE ALSO

             ROSGetInt()
             ,
             ROSSetIntVec()
             , libraries/ros.i
```

1.32 ros.library/ROSSetIntVec()

```
                NAME
     ROSSetIntVec -- set a new handler for an interrupt vector

SYNOPSIS
     success = ROSSetIntVec(intNum, handler)
     D0                     D0      A0

     BOOL ROSSetIntVec(UWORD, APTR);

FUNCTION
     This function attaches a new handler to the interrupt specified with
     'intNum'. Detach a handler with A0 = NULL.
     The routine is called from within an interrupt, so normal
     restrictions apply. On entry A5 holds the custom base ($dff000) and
     A6 holds _ROSBase. Your code may trash all registers.
     Your handler won't be invoked until you enable it (and the master int)
     via
             ROSSetInt()
             . After enabling your handler it's always invoked
     whether you taken over the systen or not.

     Interrupts are executed as follows:

     » INTB_BLIT
         - only in Death-Mode, during a
             ROSSysCall()
              the system routine
           will be restored so you do _not_ use the blitter while the system
           is (partially) active
         - returncode is always TRUE
     » INTB_VERTB
         - always invoked, also without a call to
             ROSKillSystem()
                         - in Non-Death-Mode your vblank code may be called  ↩
                             after display
           is started depending on the speed of higher prioritized vblank
           interrupts. If you want to synchronize custom chip modifications
           with the display you should use the copper.
         - but remember: in Non-Death-Mode the vblank frequency may vary
           (not always 50Hz) if ROS screen behind others!
         - returncode is always TRUE
     » INTB_COPER
         - only if ROS screen in front
```

```
        - returncode is always TRUE
    » INTB_NMI (bit 15)
        - always invoked
        - returncode is always TRUE
    » INTB_AUD0..3
        - only if channels allocated, else the returncode will be FALSE

    Further 5 new int definitions are supported:

    » INTB_CIAATIMA/CIAATIMB/CIABTIMA/CIABTIMB
        - used to attach a handler to the specified CIA timer
        - returncode may be FALSE if CIA allocation failed, in this case a
          Retry/Cancel-Requester appears (if not suppressed)
        - if returncode TRUE the specified timer bits are set to 0 (timer
          stopped) and now you may modify the contents of the CIA control
          register (CRA/CRB)
        - never read or write the CIA interrupt control register (ICR),
          use instead
            ROSSetInt()
             to enable/disble CIA interrupts!!!!!!

    » INTB_KEYB
        - used to track keyboard events (see also
            ROSGetKey()
            )
        - only invoked if ROS screen in front!!!
        - your handler is called with additional register D0 setup:
            * the lower word contains the keycode ($0..$67) with the
              additional key_up/down flag (bit7 set for key-up condition)
            * the upper word contains the qualifier bits (shift, alt, ...)
        - in Non-Death-Mode it's up to you passing this key to other system
          components (like intuition.lib, commodities.lib, ...)
            * set the ZERO-flag at the end of your handler to pass the key
              to other system components, e.g.  moveq #0,d0
            * else clear the ZERO-flag, e.g.  moveq #1,d0
            * but remember: if you catch all key events screen switching
              and other system hotkeys dosn't work!!!
            * by default all keys without left-amiga or without control
              qualifiers are catched to prevent background menu activities
              and other undesirable things
        - returncode is always TRUE

    All other interrupt bits are _not_ supported!!!

    Closing the ros.library detaches interrupt handlers automatically.

INPUTS
    intnum - the Paula interrupt bit number (0 through 14). Processor
             level seven interrupts (NMI) are encoded as intNum 15.
             Note: level 1/2/5/6 bits (disk, serial, timer) are not
                   supported!!!
                   Instead additonal bits are defined:  - KEYB
                                                        - CIAATIMA/B
                                                        - CIABTIMA/B
    handler - the routine to invoke or NULL to detach the handler

RESULT
```

```
     success - TRUE if handler successfully attached

NOTE
     This call is guaranteed to preserve all other registers.

     Do _not_ call this routine from interrupts if you want to add a CIA
     handler for the first time!!! After a successful call you may change
     this vector with ROSSetIntVec() also from interrupts because the
     desired CIA is already allocated.

SEE ALSO

             ROSSetInt()
             ,
             ROSGetInt()
             , libraries/ros.i
```

1.33 ros.library/ROSSetTimerVec()

```
                NAME
     ROSSetTimerVec -- change the timer interrupt entry point

SYNOPSIS
     ROSSetTimerVec(intRoutine, intHandle);
                    A0          A1

     VOID ROSSetTimerVec(APTR, ULONG);

FUNCTION
     This routine changes the interrupt entry point that is assocatied with
     a timer interrupt created by
             ROSAddTimerInt()
             .
     No timer modification is done (no restart, no stop). The next interrupt
     will execute your new 'intRoutine'.

INPUTS
     intRoutine - the routine to invoke upon timer interrupts.
     intHandle - handle obtained from
             ROSAddTimerInt()
             .

NOTE
     This call is guaranteed to preserve all registers (except A1!!!) and is
     save to call from interrupts.

SEE ALSO

             ROSAddTimerInt()
             ,
             ROSRemTimerInt()
             ,
             ROSStartTimer()
             ,
```

```
            ROSStopTimer()
```

1.34 ros.library/ROSStartTimer()

```
                NAME
     ROSStartTimer -- start the timer associated with the timer interrupt

SYNOPSIS
     ROSStartTimer(timeInterval, continuous, intHandle);
                   D0            D1          A1

     VOID ROSStartTimer(ULONG, BOOL, ULONG);

FUNCTION
     This routine starts a stopped timer that is assocatied with a
     timer interrupt created by
             ROSAddTimerInt()
             .

INPUTS
     timeInterval - number of micoseconds between interrupts. The
                    maximum value allowed is 90,000. If higher values
                    are passed there will be unexpected results.
     continuous - FALSE for a one shot interrupt. TRUE for multiple
                  interrupts.
     intHandle - handle obtained from
             ROSAddTimerInt()
             .

NOTE
     This call is guaranteed to preserve register A0 and is save to call
     from interrupts.

SEE ALSO

             ROSAddTimerInt()
             ,
             ROSRemTimerInt()
             ,
             ROSSetTimerVec()
             ,
             ROSStopTimer()
```

1.35 ros.library/ROSStopTimer()

```
                NAME
     ROSStopTimer -- stop the timer associated with the timer interrupt

SYNOPSIS
     ROSStopTimer(intHandle);
```

```
                  A1

     VOID ROSStopTimer(ULONG);

FUNCTION
     Stops the timer associated with the timer interrupt handle passed.
     This is used to stop a continuous timer started by

             ROSStartTimer()
             .

INPUTS
     intHandle - handle obtained from
             ROSAddTimerInt()
                NOTE
     This call is guaranteed to preserve all registers (except A1!!!) and is
     save to call from interrupts.

SEE ALSO

             ROSAddTimerInt()
             ,
             ROSRemTimerInt()
             ,
             ROSSetTimerVec()
             ,
             ROSStartTimer()
```

1.36 ros.library/ROSSysCall()

```
                NAME
     ROSSysCall -- interrupts Death-Mode to enable system calls

SYNOPSIS
     ROSSysCall()

     VOID ROSSysCall(VOID)

FUNCTION
     This function provides a method to call system functions from within
     Death-Mode.

     YOU SHOULD NEVER NEED THIS FUNCTION! IT SHOULD BE A PRIVATE ROS CALL.
     DO NOT USE THIS CALL WITHOUT GOOD JUSTIFICATION.

     In order to restore normal Death-Mode, the programmer must execute
     exactly one call to
             ROSSysCallEnd()
              for every call to ROSSysCall().

     During a SysCall all ROS interrupts continue executing (look at

             ROSSetIntVec()
              for details).
```

```
     But remember: system interrupts are also active and additional system
     DMA will be set!
     The blitter will be disowned so you do _not_ bash the blitter until

             ROSSysCallEnd()
              is called.
     Do _not_ call system functions during a SysCall which may cause any
     visual modifications (like OpenScreen(), ScreenToBack(), ...).
     These functions will never return as long as the Death-Mode is the
     active one!!!

     Calling ROSSysCall() without a Death-Mode killed system is harmless.

RESULT
     A partially active system.

NOTE
     This call is guaranteed to preserve all registers.

SEE ALSO

             ROSKillSystem()
             ,
             ROSSetDMA()
             ,
             ROSSetIntVec()
             ,
             ROSSysCallEnd()
```

1.37 ros.library/ROSSysCallEnd()

```
                NAME
     ROSSysCallEnd -- return from a SysCall

SYNOPSIS
     ROSSysCallEnd()

     VOID ROSSysCallEnd(VOID)

FUNCTION
     Call this function to return from a SysCall after a matching

             ROSSysCall()
              has been executed.

RESULT
     A Death-Mode killed system.

NOTE
     This call is guaranteed to preserve all registers.

SEE ALSO

             ROSSysCall()
```

1.38 ros.library/ROSWaitVBlank()

```
NAME
    ROSWaitVBlank -- Wait for the vertical blank to occur

SYNOPSIS
    ROSWaitVBlank()

    VOID ROSWaitVBlank(VOID);

FUNCTION
    Waits for vertical blank to occur and than returns to the caller.

NOTE
     This call is guaranteed to preserve all registers and is save to call
     from interrupts.
     This call does not require A6 to be loaded with the library base.

BUGS
    Currently this routine polls the VPOSR register so you should only use
    this call if the ROS screen is the frontmost.
    It doesn't work while displaying a 'foreign' screen on a graphics board.
```

1.39 ros.library/ROSWriteFile()

```
                NAME
     ROSWriteFile -- Write a number bytes to a file

SYNOPSIS
     actLength/errorcode = ROSWriteFile(flags, length, offset, name, buffer)
     D0        D1                       D0     D1      D2      A0    A1

     ULONG/ULONG ROSWriteFile(ULONG, ULONG, ULONG, STRPTR, VOID *);

FUNCTION
     This function writes 'length' bytes to the file 'name' using
     DOS.library's Write().
     Set 'offset' to NULL to overwrite an existing file (file opened with
     MODE_NEWFILE). In the other case the data is written to the file 'name'
     with an 'offset' from the beginning (file opened with MODE_READWRITE).
     This can be used to append/replace data to/in an existing file. If the
     file doesn't exists or if filelength shorter than 'offset' a seek error
     occurs.
     A1 points to a buffer from where the data is taken from.

     Remember: On error a corrupt or a 0-byte-file may still exists!!!
               It's up to you to delete this.

     If ROSRWF_DISKWAIT flag set the routine assumes that you want to write
```

```
     'name' to a floppy. This means:
      - waiting for the disk to be inserted (can't be cancelled!!!)
        (implementation: DOS error codes DEVICE_NOT_MOUNTED, NOT_A_DOS_DISK,
        NO_DISK force retry until any other error occurs)
      - waiting until disk drive stopped
        (implementation: polling disk.resource until drives are ready)

     Since OS2.0 it's possible to write files to your program path instead
     of the current directory using PROGDIR:xxx.
     On OS1.3 the "PROGDIR:" string is automatically removed by ROS so you
     don't have to worry about it. The only thing the user have to do is a
     "CD" to the program directory before starting (only on OS1.3). ;-)

     WARNING: It may be dangerous to use this routine within Death-Mode!
              Because of the AMIGA's multitasking facilities it isn't
              possible (isn't it???) to wait till the final end of a write
              operation. That's why I use a simple delay of 2 seconds after
              any write operation in Death-Mode. If your filesystem is to
              slow to write all the data within this time your partition
              will be _not_ validated!!!
              You've been warned!!!

INPUTS
     flags - use ROSRWF_DISKWAIT if you want to write to a floppy
     length - number of bytes to write
     offset - NULL for overwrite mode, else a number of bytes to skip
              before writing into an existing file
     name - pointer to a null-terminated string
     buffer - pointer to buffer where data is taken from

RESULT
     actLength - number of bytes written or -1 if error
     errorcode - DOS error code on failure (may be NULL!!!)

NOTE
     This function interrupts the Death-Mode state with
             ROSSysCall()
             !

SEE ALSO
     DOS.lib/Write(),
             ROSDeleteFile()
```