Default

COLLABORATORS			
	TITLE : Default		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		July 20, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Default

1.1 games.library

```
Name:          GAMES.LIBRARY AUTODOC
Version:       0.2 Beta.
Date:          07 July 1996
Author:        Paul Manias
Copyright:     DreamWorld Productions, 1996.  All rights reserved.
Notes:         This document is still being written and will contain errors
               in a number of places.  The information here is definitely not
               official - you have been warned!
```

```
 Init_GPI()
 Remove_GPI()
 Read_Mouse()
 Read_JoyPort()
 Read_JoyStick()
 Read_JoyPad()
 Read_SegaPad()
 Read_Analog()
 Read_Key()
 FastRandom()
 SlowRandom()
 Wait_LMB()
 Wait_Fire()
 Wait_Time()
 NoRequesters()
 SetFilter()
 Add_InputHandler()
 Rem_InputHandler()
 Add_Interrupt()
 Rem_Interrupt()
 SmartLoad()
 QuickLoad()
 SmartUnpack()
 SetUserPri()
 SetGMSPrefs()
 UnpackPic()
 GetPicInfo()
 SetPassword()
 AllocMemBlock()
```

```
FreeMemBlock()

SCREENS.GPI
Add_Screen()
Delete_Screen()
Show_Screen()
Hide_Screen()
SwapBuffers()
Remake_Screen()
Move_Screen()
HWScroll_Horizontal()
HWScroll_Vertical()
HWScroll_Reset()
B12_FadeToBlack()
B12_FadeToWhite()
B12_FadeToPalette()
B12_FadeToColour()
B24_FadeToBlack()
B24_FadeToWhite()
B24_FadeToPalette()
B24_FadeToColour()
Change_Colours()
Blank_Colours()

Init_RasterList()
Update_RasterList()
Update_RastLines()
Update_RastCommand()
Update_RastAllCommands()
Remove_RasterList()
Hide_RasterList()
Show_RasterList()

Init_Sprite()
Update_Sprite()
Move_Sprite()
Remove_Sprite()
Update_SpriteList()
Remove_SpriteList()
Remove_AllSprites()
Return_AllSprites()

BLITTER.GPI (Work in progress)
Init_BOB()
Init_BOBList()
Blit_BOB()
Blit_BOBList()
Clear_BOB()
Clear_BOBList()
Blit_VerticalSeries()
Blit_ScaledBMap()
Blit_Pattern()
Blit_Line()
Blit_Bitmap()
SnR_Backgrounds()
Clear_Screen()
Clear_ScreenArea()
```

```
 SOUND.GPI
 AllocAudio()
 FreeAudio()
 InitSound()
 CheckChannel()
 PlaySound()
 PlaySoundDAC1()
 PlaySoundDAC2()
 PlaySoundDAC3()
 PlaySoundDAC4()
 PlaySoundPri()
 PlaySoundPriDAC1()
 PlaySoundPriDAC2()
 PlaySoundPriDAC3()
 PlaySoundPriDAC4()
 SetVolume()
 FadeVolume()
 InitSTPlayer()
 PlaySTMOD()
 StopSTPlayer()

 VECTORS.GPI

 NETWORK.GPI

 DEBUG.GPI
```

1.2 games.library/Init_GPI

```
games.library/Init_GPI

NAME  Init_GPI - Load in a GPI and initialise it for function calls.

SYNOPSIS
  GPIBase = Init_GPI (GPINumber).
            d0                d0

  APTR Init_GPI(UWORD GPI_ID);

FUNCTION
  Loads  in  a  GPI  and  initialises  it  ready  for function calls.
  Currently  there are three GPI's that require initialisation if you
  want to use them:

  Debug.GPI
  Network.GPI
  Vectors.GPI

  If  GPIBase returns with an address pointer then the initialisation
  was successful and the GPI's functions are ready to use.

NOTE  The GPIBase is the same as a library base pointer.  Because of this
  it  is  perfectly  legal  to  make  direct calls to the GPI itself.
  However you should only do this if you have very good reason to, eg
```

```
  if you are developing a new GPI.

  As  the  Debug,  Network and Vector GPI's are not present yet, this
  function is a bit useless for the moment :-)

INPUTS  GPINumber - A recognised GPI ID Number, which is one of:

  GPI_SCREENS =  0
  GPI_BLITTER =  1
  GPI_SOUND   =  2
  GPI_NETWORK =  3
  GPI_VECTORS =  4
  GPI_DEBUG   =  5

RESULT  GPIBase - Ptr to the GPIBase or NULL if error.

SEE ALSO
  Remove_GPI
```

1.3 games.library/Remove_GPI

```
games.library/Remove_GPI

NAME  Remove_GPI  -- Remove a GPI that was previously initialised.

SYNOPSIS
  Remove_GPI(GPIBase)
                    a0

  ULONG Remove_GPI(APTR GPIBase);

FUNCTION
  Informs  the  Games.Library  that  you  no  longer  wish to use the
  specified  GPI's  functions.   You cannot make any calls to the GPI
  after removing it.

INPUTS  GPIBase - Ptr to a valid GPIBase returned from Init_GPI().

SEE ALSO
  Init_GPI
```

1.4 games.library/Read_Mouse

```
games.library/Read_Mouse

NAME  Read_Mouse -- Gets the current mouse co-ordinates and button states.

SYNOPSIS
  XYCoord = Read_Mouse (PortName)
            d0                  d0

  ULONG Read_Mouse(UWORD PortName);
```

```
FUNCTION
  Reads  the  mouse port and returns its current co-ordinates.  Mouse
  movement is restricted to the current screen's height and width.

NOTE  The  range  of  the  co-ordinates is redefined each time you open a
  screen.   You  can  alter the mouse limits by changing the relative
  values  in the GMSBase.  Remember, unlike the joyport routines this
  function  returns  screen  relative  co-ordinates,  rather than X/Y
  direction switches.

INPUT PortName = JPORT1 or JPORT2.

RESULT  XYCoord - (XCoord)<<16+(YCoord).

        MouseXx, MouseYx and MouseButtonsx and also updated accordingly in
        the GMSBase.

SEE ALSO
  games/gamesbase.i
```

1.5 games.library/Read_JoyPort

```
games.library/Read_JoyPort

NAME  Read_JoyPort -- Reads any joystick device in a given joyport.

SYNOPSIS
  JoyStatus = Read_JoyPort(PortName, ReturnType)
      d0                   d0         d1

  ULONG Read_JoyPort(UWORD PortName, UWORD ReturnType)

FUNCTION
  Reads  the  joyport  and returns its status in the required format,
  regardless  of  what  playing  device  is  plugged  in.   Currently
  supported   devices   are  standard  JoySticks,  Analog  JoySticks,
  SegaPads  and  CD32  JoyPads.   The  mouse is not supported by this
  function.

  Unlike  the  lowlevel.library  equivalent  of  this  function, this
  version is much faster and does not need to evaluate what device is
  currently  plugged in.  It simply reads the specified joy type from
  GMSPrefs and jumps to the correct routine.

  Future  devices  may  be  added  to  this  function  - this will be
  transparent to your program so that you can support devices that do
  not exist yet.

INPUTS  PortName - JPORT1, JPORT2, JPORT3 or JPORT4.
  ReturnType - JT_SWITCH: JoyStatus returns with switched bitflags.
               JT_ZBXY:   JoyStatus returns with the ZBXY format.

RESULT  JoyStatus - Status  of  the  JoyPort  in  one  of the following two
    formats:
```

```
  For JT_SWITCH : JS_LEFT   = 0
                  JS_RIGHT  = 1
                  JS_UP     = 2
                  JS_DOWN   = 3
                  JS_ZIN    = 4
                  JS_ZOUT   = 5
                  JS_FIRE1  = 6
                  JS_FIRE2  = 7
                  JS_PLAY   = 8
                  JS_RWD    = 9
                  JS_FFW    = 10
                  JS_GREEN  = 11
                  JS_YELLOW = 12

  For JT_ZBXY :

        BYTE | BIT RANGE | DATA
  -----+-----------+---------------------------------------
    1  |  0 -  7   | Y Direction
    2  |  8 - 15   | X Direction
    3  | 16 - 23   | Button status bits.
    4  | 23 - 31   | Z Direction (currently not supported)

   JB_FIRE1 = 16
   JB_FIRE2 = 17

SEE ALSO
  Read_JoyStick, Read_JoyPad, Read_SegaPad, Read_Analog, games/games.i
```

1.6 games.library/Read_JoyStick

```
games.library/Read_JoyStick

NAME  Read_Joystick -- Read the joystick status from a given joyport.

SYNOPSIS
  JoyStatus = Read_JoyStick(PortName)
           d0                       d0

  ULONG Read_JoyStick(UWORD Portname);

FUNCTION
  Interprets  the  current  status  of  a joystick in the given port.
  Ports  3 and 4 are recognised as extended joysticks in the parallel
  port.   If the user was not using the joystick, then JoyStatus will
  return a NULL value.

NOTE  Try  to  use  Read_JoyPort(),  as  that gives the same results, but
  supports Joypads, Analog joysticks etc.

INPUTS  PortName - JPORT1, JPORT2, JPORT3 or JPORT4.

RESULT  JoyStatus - The  current joystick status bits.  These are:
```

```
  JS_LEFT   =  0
  JS_RIGHT  =  1
  JS_UP     =  2
  JS_DOWN   =  3
  JS_FIRE1  =  6
  JS_FIRE2  =  7

SEE ALSO
  Read_JoyPort, Read_JoyPad, Read_SegaPad, Read_Analog, games/games.i
```

1.7 games.library/Read_Analog

```
games.library/Read_Analog

NAME  Read_Analog -- Read an analog joystick from the given port.

SYNOPSIS
        ZBXYStatus = Read_Analog(PortName)
            d0                   d0

  ULONG Read_Analog(UWORD PortName);

FUNCTION
  Reads an analog joystick in either port 1 or port 2.  The status of
  the  joystick  is  returned in ZXBYStatus (a packed state).  If the
  user was not using the joystick, then ZBXYStatus will return a NULL
  value.

  JoyPorts 3 and 4 are not supported by this function.

INPUTS  PortName - JPORT1 or JPORT2.

RESULT  ZBXYStatus - Current status of the analog joystick.

  The status data looks like this:

        BYTE | BIT RANGE | DATA
  -----+-----------+---------------------------------------
    1  |  0 -  7   | Y Direction
    2  |  8 - 15   | X Direction
    3  | 16 - 23   | Button status bits.
    4  | 23 - 31   | Z Direction (currently not supported)

  Note  that the further the joystick is pushed in a given direction,
  the  higher  the  value  returned  for the relevant byte.  Negative
  values denote a push in the opposite direction.

  eg.  If (ZXBYStatus&$0000ff00)>>8 = -3, then this signals an X
       change of 3 places left.

             If (ZXBYStatus&$000000ff) = 4, then this signals a Y change of
       4 places down.

BUGS  NOT IMPLEMENTED YET.
```

```
SEE ALSO
  Read_JoyPort, Read_JoyStick, Read_SegaPad, Read_JoyPad.
```

1.8 games.library/Read_JoyPad

```
games.library/Read_JoyPad

NAME  Read_JoyPad -- Reads a CD32 joypad from a specified port number.

SYNOPSIS
  JoyStatus = Read_JoyPad(PortName)
          d0                      d0

  ULONG Read_JoyPad(UWORD PortName);

FUNCTION
  Reads  a  standard  Amiga JoyPad (ie a CD32 joypad) and returns its
  current  status  in  the JoyStatus bit format.  If the user was not
  using the joypad, then JoyStatus will return a NULL value.

INPUTS  PortName - JPORT1 or JPORT2.

RESULT  JoyStatus - Current joypad status bits. These are:

  JS_LEFT   = 0
  JS_RIGHT  = 1
  JS_UP     = 2
  JS_DOWN   = 3
  JS_RED    = 6
  JS_BLUE   = 7
  JS_PLAY   = 8
  JS_RWD    = 9
  JS_FFW    = 10
  JS_GREEN  = 11
  JS_YELLOW = 12

  The red and blue buttons are the equivalent of fire buttons 1 and 2
  on a standard joystick.

BUGS  I have not tested this!

SEE ALSO
  Read_JoyPort, Read_JoyStick, Read_SegaPad, Read_Analog,
  games/games.i
```

1.9 games.library/Read_SegaPad

```
games.library/Read_SegaPad

NAME  Read_SegaPad - Reads a Sega joypad from a specified port number.

SYNOPSIS
```

```
  JoyStatus = Read_SegaPad(PortName)
     d0                    d0

  ULONG Read_SegaPad(UWORD PortName)

FUNCTION
  Reads  a standard Sega JoyPad and returns its current status in the
  JoyStatus  bit format.  If the user was not using the SegaPad, then
  JoyStatus will return a NULL value.

INPUTS  PortName - JPORT1 or JPORT2.

RESULT  JoyStatus - Current joypad status bits. The flags are:

  JS_LEFT  =  0
  JS_RIGHT =  1
  JS_UP    =  2
  JS_DOWN  =  3
  JS_FIRE1 =  6
  JS_FIRE2 =  7

BUGS  This  has  not  even  been  tested  by me!  Probably doesn't work -
  someone send me a reader that has been tested please.

SEE ALSO
  Read_JoyPort, Read_JoyStick, Read_JoyPad, Read_Analog, games/games.i
```

1.10 games.library/Read_Key

```
games.library/Read_Key

NAME  Read_Key -- Reads the keyboard and returns any new keypresses.

SYNOPSIS
  KeyValues = Read_Key(KeyStruct)
          d0                   a0

  ULONG Read_Key(struct KeyStruct *);

FUNCTION
  Checks  to  see  if  there  was  a keypress since the last time you
  called  this  routine.   If there were no keypresses then KeyValues
  will return a null value.

  Most  key values are returned as ANSI, which is of the range 1-127.
  Special  keys  (eg  Cursor Keys, function Keys etc) are held in the
  range of 128-255.

  A  shift  qualifier  has  an automatic effect on the ANSI value (eg
  shift+c  will  return  "C").   Alt  keys, Ctrl keys, and Amiga keys
  have no effect on the ANSI value.

  The  KeyStruct  is  also updated for future reference.  A KeyStruct
  will  hold  up  to  three keys since your previous check.  For this
  reason you should call Read_Key every VBL when the user is required
```

```
  to enter more than a few characters (eg their name).  Otherwise you
  could lose some of their input.

INPUT KeyStruct - Pointer to a valid KeyStruct.  This structure is in the
    form of:

        STRUCTURE  KP,00
      UWORD  KP_ID                ;Updated by function, ignore.
      UBYTE  KP_Qualifier1        ;Latest key press.
      UBYTE  KP_Key1
      UBYTE  KP_Qualifier2        ;Older key press.
      UBYTE  KP_Key2
      UBYTE  KP_Qualifier3        ;Oldest key press.
      UBYTE  KP_Key3

RESULT  KeyValues - Contains the latest keypress value in the first byte (8
  bits)  and  its  qualifier  in  the  upper  byte  (bits 9 - 16).  A
  previous  keypress  may  also  be  held in the upper word if it was
  missed.  Visually:

  (Qualifier2<<8+Key2)<<16 + (Qualifier1<<8+Key1)

  KeyStruct - Updated to hold new key data.  You can test for a third
    keypress inside here.

SEE ALSO
  Add_InputHandler, games/games.i
```

1.11 games.library/FastRandom

```
games.library/FastRandom

NAME  FastRandom -- Create a random number from a given range.

SYNOPSIS
        Random = FastRandom(Range)
         d0.w               d1

  WORD FastRandom(UWORD Range);

FUNCTION
  Creates  a  random number as quickly as possible.  The routine only
  has one divide to determine the range and will automatically change
  the randseed value each time you call it.

  This  routine  will  generally get all the numbers in fairly random
  sequences.

  Remember  that  all  generated numbers fall BELOW the Range, ie the
  Range  is an "unreachable" number.  Add 1 to your range if you want
  this number included.

INPUTS  Range - A  range  between  1 and 32767.  An invalid range of 0 will
    result in a division by zero error.
```

```
RESULT  Random - A number greater or equal to 0, and less than Range.

SEE ALSO
  SlowRandom, examples/random.c
```

1.12 games.library/SlowRandom

```
games.library/SlowRandom

NAME  SlowRandom -- Create a random number from a given range.

SYNOPSIS
  Random = SlowRandom(Range)
    d0                d1

  ULONG SlowRandom(UWORD Range);

FUNCTION
  Creates  a  very good random number in a relatively short amount of
  time.   The  routine  takes  approximately  two  times  longer than
  FastRandom,   but  is  guaranteed  of  giving  good  random  number
  sequences.

  Remember  that  all  generated numbers fall BELOW the Range, ie the
  Range  is an "unreachable" number.  Add 1 to your range if you want
  this number included.

INPUTS  Range - A range between 1 and 32767.

RESULT  Random - A number greater or equal to 0, and less than Range.

SEE ALSO
  FastRandom, examples/random.c
```

1.13 games.library/Wait_LMB

```
games.library/Wait_LMB

NAME  Wait_LMB -- Wait for the user to hit the left mouse button.

SYNOPSIS
  Wait_LMB()

  void Wait_LMB(void);

FUNCTION
  Waits  for  the  user  to  hit  the left mouse button.  It will not
  return to your program until this event occurs.  Multi-tasking time
  will  be  increased  while  waiting and an implanted AutoOSReturn()
  call supports screen switching.

SEE ALSO
```

```
  Read_Mouse, Wait_Fire.
```

1.14 games.library/Wait_Fire

```
games.library/Wait_Fire

NAME  Wait_Fire -- Wait for the user to hit a fire button.

SYNOPSIS
  Wait_Fire(PortName)
        d0

  void Wait_Fire(UWORD PortName);

FUNCTION
  Waits  for  the user to hit the fire button.  It will not return to
  your  program  until this event occurs.  Multi-tasking time will be
  increased  while  waiting  and  an  implanted  AutoOSReturn()  call
  supports screen switching.

INPUTS  PortName - JPORT1, JPORT2, JPORT3 or JPORT4.

SEE ALSO
  Read_Joystick, Read_JoyPad, Read_SegaPad, Wait_LMB, games.i
```

1.15 games.library/Wait_Time

```
games.library/Wait_Time

NAME  Wait_Time -- Wait for a specified amount of micro-seconds.

SYNOPSIS
  Wait_Time(MicroSeconds)
                d0

  void  Wait_Time(UWORD MicroSeconds);

FUNCTION
  Waits for a specified amount of micro-seconds.  During this time it
  will   reduce   the   task  priority  and  make  regular  calls  to
  AutoOSReturn() for you.

SEE ALSO
  Wait_VBL, Wait_OSVBL
```

1.16 games.library/Wait_VBL

```
games.library/Wait_VBL

NAME  Wait_VBL -- Waits for a vertical blank.
```

```
SYNOPSIS
  Wait_VBL()

  void Wait_VBL(void);

FUNCTION
  This  is  a  very  intelligent  routine and will wait for the exact
  position  of  the  VBL.   Even if you move your screen around using
  Reposition_Screen,  the  wait  line will move along with it, giving
  you more (or less) VBL space.

NOTE  Use Wait_OSVBL if you want automatic screen switching checks.

SEE ALSO
  Wait_RastLine, Wait_OSVBL.
```

1.17 games.library/Wait_OSVBL

```
games.library/Wait_OSVBL

NAME  Wait_OSVBL -- Waits for a vertical blank.

SYNOPSIS
  Wait_OSVBL()

  void Wait_OSVBL(void);

FUNCTION
  This  is  a  very  intelligent  routine and will wait for the exact
  position  of  the  VBL.   Even if you move your screen around using
  Reposition_Screen,  the  wait  line will move along with it, giving
  you more (or less) VBL space.

  This version has an implanted AutoOSReturn call to make screen
  switching very easy to implement.

SEE ALSO
  Wait_RastLine, Wait_VBL.
```

1.18 games.library/Wait_RastLine

```
games.library/Wait_RastLine

NAME  Wait_RastLine -- Waits for the strobe to reach a specific line.

SYNOPSIS
  Wait_RastLine(LineNumber)
                d0

  void Wait_RastLine(UWORD LineNumber)
```

```
FUNCTION
  This  routine waits for the strobe to reach the scan-line specified
  in  LineNumber.   The recognised range is 1-311 lo-res pixels for a
  PAL screen.

  Wait_Rastline  is  a  clever  routine.  All the VBL wait routines I
  have  ever  seen never consider the fact that somtimes an interrupt
  can  be  activated  while  you're  waiting for the line to come up.
  This causes you to completely miss the VBL, and you are stalled for
  an   extra   frame   which   can   have   horrible   effects   on   screen.
  Unfortuntely there is also a bug in the hardware which makes this a
  difficult problem to avoid!

  Luckily  I've  managed  to  fix  all this so rest assured that this
  function works very well.

NOTES Here is the bug in the hardware concerning VPOSR and VHPOSR.  When
  the strobe reaches line 255, this happens:

  Line:  254
  Line:  255
  Line:  000    <- Bug here.
  Line:  256
  Line:  257
  Line:  ...

  If  your VBL synchronisations have failed in the past, now you know
  why...  Yet another good reason to use library routines!

INPUTS  LineNumber - Vertical beam position to wait for.

BUGS  If  you enter a large value beyond the range limit, like #400 for a
  non-interlaced  PAL  screen,  the strobe will never reach this line
  because  line 400 doesn't even exist.  This will cause your program
  to  lock  up.   So make sure you keep your values between 1 and 310
  for normal screens and 1 - 511 for interlaced screens.

SEE ALSO
  Wait_OSVBL, Wait_VBL.
```

1.19 games.library/NoRequesters

```
games.library/NoRequesters

NAME  NoRequesters -- Shuts down all requesters.

SYNOPSIS
  NoRequesters()

  void NoRequesters(void);

FUNCTION
  Shuts  down all requesters in a system-friendly way.  If you intend
  to  load/save  files  through DOS, you should call this function at
```

```
  the start of your program.
```

1.20 games.library/SetFilter

```
games.library/SetFilter

NAME  SetFilter -- Set the LED/Sound filter to an ON or OFF state.

SYNOPSIS
  SetFilter(NewStatus)
             d0

  void SetFilter(UWORD NewStatus)

FUNCTION
  Sets  the  sound  filter to the required state.  The power LED will
  dim or brighten to reflect the change.

INPUT NewStatus - Either FILTER_ON or FILTER_OFF
```

1.21 games.library/Add_InputHandler

```
games.library/Add_InputHandler

NAME  Add_InputHandler -- Add an input handler to the system.

SYNOPSIS
  Add_InputHandler()

  void Add_InputHandler(void)

FUNCTION
  Add  an  input handler at the highest priority to delete all system
  input  events.   The  idea  behind this is to prevent input falling
  through  to system screens and to give you more CPU time by killing
  all inputs.

  If  you  are  going  to  use  Read_Key() then it is vital that this
  function  is active.  This is because Read_Key() is hooked into the
  input handler routine that this function provides.

NOTE  By  default  this  function  is  always  called  by  Show_Screen().
  Therefore  you only need to call this routine if you are using some
  other screen opening routine.

SEE ALSO
  Rem_InputHandler
```

1.22 games.library/Rem_InputHandler

```
games.library/Rem_InputHandler

NAME  Rem_InputHandler -- Remove the active input handler.

SYNOPSIS
  Rem_InputHandler()

  void Rem_InputHandler(void)

FUNCTION
  Removes the active input handler from the system.  As a result this
  will also deactivate the Read_Key() function.

NOTE  Delete_Screen() automatically calls this function so that any input
  handlers set up by Show_Screen() are removed.

SEE ALSO
  Add_InputHandler
```

1.23 games.library/Add_Interrupt

```
games.library/Add_Interrupt

NAME  Add_Interrupt -- Activate a custom written hardware interrupt.

SYNOPSIS
  IntBase = Add_Interrupt(Interrupt, IntNum, IntPri)
     d0                       a0        d0      d1

  ULONG Add_Interrupt(APTR Interrupt, UWORD IntNum, BYTE IntPri)

FUNCTION
  Initialises  a  system-friendly hardware interrupt and activates it
  immediately.   See  the  SetIntVector() and AddIntServer() descrip-
  tions in the exec.library for more details on system interrupts.

INPUTS  Interrupt - Ptr to your interrupt routine.
  IntNum - The hardware interrupt bit.
  IntPri - The priority of the interrupt, -126 to +127.

RESULT  IntBase  - Ptr to the interrupt base, you have to save this address
  and pass it back to Rem_Interrupt() before your program exits.

SEE ALSO
  Rem_Interrupt, exec/SetVector, hardware/custom.i, games/games.i
```

1.24 games.library/Rem_Interrupt

```
games.library/Rem_Interrupt

NAME  Rem_Interrupt -- Remove an active interrupt.
```

```
SYNOPSIS
  Rem_Interrupt(IntBase)
                d0

  void Rem_Interrupt(ULONG IntBase)

FUNCTION
  Disable  and  remove  an  active  interrupt  from the system.  This
  function is identical to RemIntServer() in the exec.library, but is
  a little easier to handle.

INPUT IntBase - Ptr to an interrupt base returned from Add_Interrupt().

SEE ALSO
  Add_Interrupt, games.i
```

1.25 games.library/SmartLoad

```
games.library/SmartLoad

NAME  SmartLoad -- Load in a file and depack it if possible.

SYNOPSIS
  MemLocation = SmartLoad(FileName, Destination, MemType)
      d0                    a0          a1         d0

  ULONG SmartLoad(APTR FileName, APTR Destination, ULONG MemType)

FUNCTION
  Load  in a file and depack it if necessary.  If the function cannot
  find  a  recognised packer for the file then it will assume that it
  is not packed, and load it in without alteration.

  If  you  pass  NULL  as  the  Destination address, SmartLoad() will
  allocate  the  memory  for  you  but you must give the MemType (see
  exec/memory.h).

  If  you  do  give  the  Destination  you  do  not have to pass this
  function the MemType.

NOTE  This function is not fully working yet!

INPUTS  FileName - Ptr to a null terminated string containing a file name.
  Destination - Destination for unpacked data or NULL for allocation.
  MemType - Memory Type (only required if Destination is NULL)

RESULT  MemLocation - Ptr to the loaded data or NULL if failure.

SEE ALSO
  QuickLoad, SmartUnpack, SetPassword, exec/memory.i
```

1.26 games.library/QuickLoad

```
games.library/QuickLoad

NAME  QuickLoad -- Load in a file without any depacking.

SYNOPSIS
  MemLocation = QuickLoad(FileName, Destination, MemType)
       d0                 a0           a1        d0

  APTR QuickLoad(STRPTR FileName, APTR Destination, ULONG MemType)

FUNCTION
  Loads  in a file without attempting to depack it.  The advantage of
  this  function is that it will assess the file size and load it all
  in for you.  It can also allocate the memory space if required.

  If  you  pass  NULL  as the Destination address, SmartUnpack() will
  allocate  the  memory  for  you  but you must give the MemType (see
  exec/memory.h).

  If  you  do  give  the  Destination  you  do  not have to pass this
  function the MemType.

NOTE  If  you  wanted  the  allocation  you  will  have  to  free it with
  FreeMemBlock() when you are finished with it.

INPUTS  FileName - Ptr to a null terminated string containing a file name.
  Destination - Destination for unpacked data or NULL for allocation.
  MemType - Memory Type (only required if Destination is NULL)

RESULT  MemLocation - Ptr to the loaded data or NULL if failure.

SEE ALSO
  SmartLoad, SmartUnpack, exec/memory.i
```

1.27 games.library/SmartUnpack

```
games.library/SmartUnpack

NAME  SmartUnpack -- Unpack data from one memory location to another.

SYNOPSIS
  MemLocation = SmartUnpack(Source, Destination, MemType)
           d0                 a0           a1         d0

  APTR SmartUnpack(APTR Source, APTR Destination, ULONG MemType)

FUNCTION
  Attempts  to unpack a data area if it can assess the packing method
  used.  The data should begin with an ID number followed by the size
  of the original data before it was packed.  The data itself must be
  found after this.
```

```
  If you pass NULL as the destination address, SmartUnpack() will
  allocate the memory for you but you must give the MemType (see
  exec/memory.h).  If you give the destination you do not have to
  pass this function the MemType.

NOTE  Currently this function only supports the RNC packer type.  It will
  support the XPK library very soon so that other files can depacked.

  If you wanted the allocation you will have to free it with
  FreeMemBlock() when you are finished with it.

INPUTS  Source - Ptr to start of packed data (must be an ID header).
  Destination - Destination for unpacked data or NULL for allocation.
  MemType - Memory type (only required if Destination is NULL).

RESULT  MemLocation - Ptr to the unpacked data.

SEE ALSO
  SmartLoad, exec/memory.i
```

1.28 games.library/SetUserPri

```
games.library/SetUserPri

NAME  SetUserPri -- Set your task to a user selected priority.

SYNOPSIS
  SetUserPri()

  void SetUserPri(void)

FUNCTION
  Sets your task to a user selected priority.  This priority will
  depend on the UserPri setting in GMSBase, which comes from the
  ENV:GMSPrefs file.  The priority setting can be altered in the
  GMSPrefs utility.
```

1.29 games.library/SetGMSPrefs

```
games.library/SetGMSPrefs

NAME  SetGMSPrefs -- Initialise a new set of preferences.

SYNOPSIS
  ErrorCode = SetGMSPrefs(PrefsStruct)
     d0                   a0

  UWORD SetGMSPrefs(APTR PrefsStruct)

FUNCTION
  Initialise a new set of GMS preferences in the games.library.  This
  will overwrite the prefs previously set in memory.
```

```
INPUT PrefsStruct  -  Ptr  to  a valid preferences structure.  Details of
  this  structure  are  not  available  to you for the moment, so you
  can't actually make any use of this function just yet :-)

RESULT  ErrorCode - Returns NULL if successful.
```

1.30 games.library/GetPicInfo

```
games.library/GetPicInfo

NAME  GetPicInfo -- Get the information on a recognised picture header.

SYNOPSIS
  ErrorCode = GetPicInfo(Picture, PicInfo)
     d0                   a1       a2

  UWORD GetPicInfo(APTR Picture, APTR PicInfo)

FUNCTION
  NOT IMPLEMENTED YET.

INPUT

RESULT  ErrorCode - Returns NULL if successful.
```

1.31 games.library/SetPassword

```
games.library/SetPassword

NAME  SetPassword -- Set a password for unpacking/decrypting files in
                SmartLoad and SmartUnpack.

SYNOPSIS
  SetPassword(Password)
                d0

  void SetPassword(ULONG Password)

FUNCTION
  Set  a  password for the decryption of specially packed files.  For
  example  the RNC packer type allows you to use an encryption key in
  the packing/depacking of files.

INPUT Password - A password/encryption key of 4 bytes (1 longword).

SEE ALSO
  SmartLoad, SmartUnpack
```

1.32 games.library/AllocMemBlock

```
games.library/AllocMemBlock

NAME  AllocMemBlock -- Allocate a new memory block.

SYNOPSIS
  MemBlock = AllocMemBlock(Size, MemType)
     d0                    d0     d1

  APTR AllocMemBlock(ULONG Size, ULONG MemType)

FUNCTION
  Allocates a memory block from the system - this function is almost,
  or  probably  is  identical  to AllocVec().  It exists here because
  AllocVec() is only available on V36+ machines.

  See  AllocMem()  in  the  exec.library  for  more details on memory
  allocation.

INPUT Size - Size of the required memblock in bytes.
  MemType - The type of memory as outlaid in exec/memory.i

RESULT  MemBlock - Ptr  to  the start of your allocated memblock or NULL if
    failure.   If the allocation was the successful then -4(MemBlock)
    will  contain  the  size  of your allocated memory.  You can read
    this  value, but DON'T write to it!  You can do whatever you like
    to the rest of your memory block of course.

SEE ALSO
  FreeMemBlock, exec/memory.i
```

1.33 games.library/FreeMemBlock

```
games.library/FreeMemBlock

NAME  FreeMemBlock -- Free a previously allocated mem block.

SYNOPSIS
  FreeMemBlock(MemBlock)
               a0

  void FreeMemBlock(APTR MemBlock)

FUNCTION
  Frees a memory area allocated by AllocMemBlock().  This function is
  essentially the same as FreeVec(), but works on all Amigas.

NOTE  Never free the same MemBlock twice.

INPUT MemBlock - Points to the start of a memblock.

SEE ALSO
  AllocMemBlock
```

1.34 games.library/Add_Screen

```
games.library/Add_Screen

NAME  Add_Screen -- Sets up a screen from given parameters.

SYNOPSIS
  ErrorCode = Add_Screen(GameScreen)
     d0                  a0

FUNCTION
  Initialises  a GameScreen structure by allocating the screen memory
  and making the copperlist.  A little more complex than it sounds...

  After  calling  this function you need to call Show_Screen() to get
  the screen on the display.

INPUTS  GameScreen - Ptr to a valid GameScreen structure.  Currently the
  structure look like this:

   STRUCTURE  GameScreen,0          ;A GameScreen structure
  ULONG SS_VERSION            ;Vesion - "GSV1"
  APTR  SS_MemPtr1            ;Ptr to screen 1
  APTR  SS_MemPtr2            ;Ptr to screen 2 (if buffered).
  APTR  SS_MemPtr3            ;Ptr to screen 3 (triple buffer!!)
  APTR  SS_ScreenLink         ;Ptr to a linked screen.
  APTR  SS_Palette            ;Ptr to the requested palette.
  APTR  SS_RasterList         ;Ptr to a raster list (or not).
  ULONG SS_AmtColours         ;The amount of colours on screen.
  UWORD SS_Scr_Height         ;The height of the visible screen.
  UWORD SS_Scr_Width          ;The width of the visible screen.
  UWORD SS_Scr_ByteWidth      ;Width of the screen in *bytes*.
  UWORD SS_Pic_Height         ;The height of the entire screen.
  UWORD SS_Pic_Width          ;The width of the entire screen.
  UWORD SS_Pic_ByteWidth      ;Width of the entire screen, bytes.
  UWORD SS_Planes             ;The amount of planes in da screen.
  WORD  SS_TopOfScrX          ;Hardware co-ordinate for TOS.
  WORD  SS_TopOfScrY          ;Hardware co-ordinate for LOS.
  UWORD SS_ScrollBuffer       ;Scroll buffer in pixels.
  WORD  SS_ScrollXCount       ;Offset of the horizontal axis.
  WORD  SS_ScrollYCount       ;Offset of the vertical axis.
  ULONG SS_ScrAttrib          ;Special Attributes are?
  UWORD SS_ScrMode            ;What screen mode is it?
  UBYTE SS_ScrType            ;Interleaved/Planar/Chunky?
  UBYTE SS_Displayed          ;Am I currently displayed?
  APTR  SS_Extended           ;For extended tag lists, not used.
  APTR  SS_ScreenStats        ;Reserved, do not touch.

  Here follows a description of each field:

  SS_VERSION
  The  version  of  the structure.  Currently this is "GSV1".  In the
  future  as  the  structure  grows,  you  will  be able to use other
  structure versions, but for now this is what you're stuck with.

  SS_MemPtr1, SS_MemPtr2, SS_MemPtr3
```

These fields point to the screen display data. They should be NULL if you want this function to allocate the memory for you (highly recommended). Otherwise Add_Screen() will assume that the values are valid pointers to video memory and use them as such.

SS_ScreenLink
If you want to set up a second screen at a different position in the viewport, or create an extra (double) playfield, point to the next GameScreen structure here.

SS_Palette
Points to the palette for this screen, or NULL if you want to install a clear palette (all colours black). By default your palette structure must be in 12 bit colours, unless you set the _24BITCOL flag in SS_ScrAttrib.

SS_RasterList
Points to a valid rasterlist structure, or NULL. RasterLists are made up of instructions that are executed as the monitor beam travels down the screen. See Init_RasterList() for more information on rasterlists.

SS_AmtColours
The amount of colours in the screen palette, as pointed to by SS_Palette. If you set this value to NULL then Add_Screen() will fill it in for you, via a check to SS_Planes. This parameter exists so that you can set colours that can't be accessed by the screen's bitmap. For example, if your screen is 16 colours but you want to set the colours for the sprites, then you can use a 32 colour palette.

SS_ScrHeight, SS_ScrWidth, SS_ScrByteWidth
Defines the screen height and width. This is the "window" that the picture data is displayed through. The width of the screen must be divisible by 16, and the byte width is defined as the ScreenWidth divided by 8.

SS_PicHeight, SS_PicWidth, SS_PicByteWidth
Defines the picture height and width. The picture is the display data that shows through on screen. It can be larger than the screen area, but must never be smaller than the screen area. If the picture is the same size as your screen, just duplicate the screen values here. Remember the width of the picture must be divisible by 16, and the byte width is defined as the PicWidth divided by 8.

SS_Planes
Specifies the amount of bitplanes that will be used by this screen. The amount of colours you can use is completely dependent on this value. For interleaved or planar screens you can calculate the amount of colours you get with the formula 2^n, where n is the amount of planes. If you are going to set up a 256 colour chunky screen, you must specify only 1 plane here.

SS_TopOfScrX, SS_TopOfScrY
Specifies the hardware offset for the screen, in lo-res pixels only (even if the screen itself is in hi-res). These two values are

added to the user's screen offset in GMSPrefs. For this reason a setting of 0,0 is usually sufficient, unless you are going to create an extra large screen (eg overscan). Negative values are permissable.

SS_ScrollBuffer
Controls the amount of blank space at the left of the screen. This is here only for horizontal hardware scrolling. It is normally set to 2 bytes (gives 16 blank pixels)

SS_ScrollXCount, SS_ScrollYCount
Counters for vertical and horizontal scrolling. These values are altered by HWScroll_Horizontal() and HWScroll_Vertical() - you are not expected to put anything in these parameters. The idea is that you can check them to see how many pixels you have scrolled in either direction. You should reset them to 0,0 before calling Add_Screen() if you are going to make use of these values in your code.

SS_ScrAttrib
Defines the special attributes for the screen. Current available are:

DBLBUFFER - Allocates an extra screen buffer which is placed in SS_MemPtr2. See the SwapBuffers() function for more information on double buffering.

TPLBUFFER - Allocates two extra buffers which are placed in SS_MemPtr2 and SS_MemPtr3. See the SwapBuffers() for more information on triple buffering.

Note: Never set the DBLBUFFER flag in conjunction with the TPLBUFFER flag!

PLAYFIELD - Must be set if this screen forms part of a playfield.

HSCROLL - Set if you want to use full horizontal hardware scrolling. This will allocate extra memory for safe scrolling outside of the picture's boundary.

VSCROLL - Set if you want to use vertical hardware scrolling.

SPRITES - Set if you intend to use sprites with this screen.

NOBURST - Enforce AGA burst modes to OFF.

BLKBDR - Turns all colours outside of the display window to black.

NOSPRBDR - Allows sprites to appear outside of the screen display window.

SS_ScrMode
Defines the display mode for the screen. If you do not fill in this field, you will get the default of Lo-Res, Planar, PAL, and 12Bit colours.

```
    LORES      - Specifies  a lo-resolution screen.  This is the def-
                 ault,  so you do not have to specify it if you don't
                 want to.

    HIRES      - Specifies a hi-resolution screen (double width).

    SHIRES     - Specifies a superhi-resolution screen (quad width).

    INTERLACED - Creates an interlaced display (double height).

    NTSC       - Forces an NTSC style display.  The default is PAL if
                 you do not set this bit.

    HAM        - HAM  mode.  The  amount of colours you get is depen-
                 dant on the amount of planes in the screen.

    _24BITCOL  - Inform GMS that we will be using 24 bit colours with
                 this screen.

  If  the  user  has  selected  mode  promotion in GMSPrefs, then the
  display  frequencies will be altered accordingly.  You cannot force
  mode promotion from inside your program.

  SS_ScrType
  The  display  data  type  -  either  PLANAR, INTERLEAVED or CHUNKY.
  Descriptions  of  these  display types are out of the scope of this
  autodoc, try the RKM's for more information on this.

RESULT  ErrorCode - NULL if successful.

BUGS  If  you  set  up  your  screen  structure  incorrectly or try to do
  something this routine doesn't, you will run into trouble.  Not all
  features  are  working  even  though  the flags are present, but it
  shouldn't be too long before this function is finished.

SEE ALSO
  Delete_Screen, Show_Screen, Hide_Screen, games/games.i
```

1.35 games.library/Delete_Screen

```
games.library/Delete_Screen

NAME  Delete_Screen -- Deactivates a screen, returns memory, etc.

SYNOPSIS
  Delete_Screen(GameScreen)
                a0

  void Delete_Screen(struct GameScreen *);

FUNCTION
  This   function  will  deallocate/deactivate  everything  that  was
  initialised when you called Add_Screen().

  If  the  screen  you  delete is currently active when you call this
```

```
  function,  intution will be given back the display.  If you want to
  get  around  this  initialise and display your next screen and then
  delete the old one.

  This  function  will clear SS_MemPtr1, SS_MemPtr2 and SS_MemPtr3 in
  the GameScreen structure.

INPUTS  GameScreen - Pointer to an initialised GameScreen structure.

SEE ALSO
  Add_Screen, Hide_Screen, Show_Screen
```

1.36 games.library/Show_Screen

```
games.library/Show_Screen

NAME  Show_Screen -- Displays an initialised game screen.

SYNOPSIS
  Show_Screen(GameScreen)
                a0

  void Show_Screen(struct GameScreen *)

FUNCTION
  Displays  an  initialised GameScreen.  A GameScreen is incompatible
  with  intuition screens so all system screens will be disabled as a
  result.

  This  function  makes a call to Add_InputHandler() to prevent input
  falling through to intuition screens.

  It  is  perfectly  admissable  to  call  this function when another
  GameScreen is already being displayed.

INPUTS  GameScreen - Pointer to an initialised GameScreen structure.

SEE ALSO
  Hide_Screen, Add_Screen, Delete_Screen.
```

1.37 games.library/Hide_Screen

```
games.library/Hide_Screen

NAME  Hide_Screen -- Hides any displayed GameScreen from view.

SYNOPSIS
        Hide_Screen()

  void Hide_Screen(void)

FUNCTION
```

```
  Hides  a  displayed  screen  from  view.  This  will  cause the OS
  viewport  to  be  returned, but your game will still be running "in
  the background".

  On  its  own  this  is  not a good multi-tasking function - use the
  special screen switching functions for that.

NOTE  It  is not necessary to pass a GameScreen to this function, it will
  return  the  system screens' regardless of what is being displayed.
  This makes the function quite useful for external debuggers.

SEE ALSO
  Show_Screen, Add_Screen, Delete_Screen.
```

1.38 games.library/SwapBuffers

```
games.library/SwapBuffers

NAME  SwapBuffers -- Switch the screen display buffers.

SYNOPSIS
  SwapBuffers(GameScreen)
                a0

  void SwapBuffers(struct GameScreen *)

FUNCTION
  Swaps  SS_MemPtr1  and  SS_MemPtr2 and activates the new bitmap for
  the  display.  If  triple  buffered,  then  all three MemPtr's are
  switched.  Visually:

  BEFORE          AFTER
  MemPtr1         MemPtr2
  MemPtr2  ---->  MemPtr3
  MemPtr3         MemPtr1

  You can get the addresses contained in these values, but you must
  never change these pointers once you have called Add_Screen().

INPUTS  GameScreen - Ptr to an initialised GameScreen structure.
```

1.39 games.library/Remake_Screen

```
games.library/Remake_Screen

NAME  Remake_Screen -- Remakes  the screen display according to its size,
                    width, and position on the monitor.

SYNOPSIS
  Remake_Screen(GameScreen)
                  a0
```

```
FUNCTION
  Remakes  the  GameScreen's  display  window as quickly as possible.
  Extreme  or  invalid  values  are  not  checked  for, so it is your
  responsibility to ensure all values are within their limits.

  If  the GameScreen is hidden then the changes will show up the next
  time you call Show_Screen().

  You cannot change the display mode, screen type or amount of screen
  colours with this function.

INPUTS  GameScreen - Ptr to an initialised GameScreen structure.

SEE ALSO
  Move_Screen
```

1.40 games.library/HWScroll_Horizontal

```
games.library/ScrollScr_Horizontal

NAME  HWScroll_Horizontal -- Scrolls the screen left and right.

SYNOPSIS
  HWScroll_Horizontal(GameScreen, XShift)
                               a0         d0

  void HWScroll_Horizontal(struct GameScreen *, UWORD XShift);

FUNCTION
  This  is  the  routine  for  hardware  scrolling  in the left/right
  direction.   You  MUST  have a knowledge of what hardware scrolling
  actually  does  before  attempting  to  use this routine.  See some
  examples to get a kickstart in this direction.

  If the graphics hardware does not support hardware scrolling, this
  routine  will  blit  the  entire  screen in the required direction.
  This is very slow but is the only other option.

  If  you  have  opened  a picture larger than the screen display and
  just  want  to  scroll  around  inside  of  that,  you can use this
  function for that purpose.

INPUTS  GameScreen - Ptr to an initialised GameScreen structure.
  XShift - The amount of pixels to move on the X axis.

BUGS  If  you  move the screen more than 50 screens in either the left or
  right  direction,  you  will  start  overwriting  something  else's
  memory.   If  your  levels are 100 screens wide though, I seriously
  think  you  should  look at what your idea of "gameplay" is, though
  ;-)
```

1.41 games.library/HWScroll_Vertical

```
games.library/HWScroll_Vertical

NAME  HWScroll_Vertical - Scrolls the screen up and down.

SYNOPSIS
  HWScroll_Vertical(GameScreen, YShift)
                        a0        d0

  void HWScroll_Vertical(struct GameScreen *, UWORD YShift);

FUNCTION
  Scrolls  the  screen  vertically  just  by  altering  the  required
  hardware bits.

  If the graphics hardware does not support hardware scrolling, this
  routine  will  instead  blit  the  entire  screen  in  the required
  direction.  This is very slow but is the only other option.

NOTE  Unlike  horizontal  scrolling,  there  is  no scroll limit for this
  function.

INPUTS  GameScreen - Ptr to an initialised GameScreen structure.
  YShift - Amount of pixels to shift. -Ve shifts up, +Ve shifts down.
```

1.42 games.library/B12_FadeToBlack

```
games.library/ B12_FadeToBlack

NAME  B12_FadeToBlack -- Fade all colours to black.

SYNOPSIS
  FadeState = B12_FadeToBlack(GameScreen)
     d0                           a0

  UWORD B12_FadeToBlack(struct GameScreen *)

FUNCTION
  Fades  all  the colours in the specified screen to black.  Once you
  call  this  function, you have to keep on calling it until it gives
  you  a  result  of NULL.  This allows you to put this function in a
  loop and do other things while the fade is active.

  For a 24 bit palette use B24_FadeToBlack().

INPUTS  GameScreen - An initialised GameScreen structure.

RESULT  FadeState - Returns NULL if fade has finished.

SEE ALSO
  B24_FadeToBlack.
```

1.43 games.library/B12_FadeToWhite

```
games.library/ B12_FadeToWhite

NAME  B12_FadeToWhite -- Fade (flash) all colours to white.

SYNOPSIS
  FadeState = B12_FadeToBlack(GameScreen, StartCol, AmtCols)
     d0                          a0          d0       d1

  UWORD B12_FadeToWhite(struct GameScreen *, UWORD StartCol,
                        UWORD AmtCols);

FUNCTION
  Fades  the colours in the specified screen to white.  Once you call
  this  function, you have to keep on calling it until it gives you a
  result of NULL.  This allows you to put this function in a loop and
  do other things while the fade is active.

  For a 24 bit palette use B24_FadeToWhite().

EXAMPLE
  In assembler, of course :-)

  Fade: CALL   Wait_OSVBL
        lea    GameScreen(pc),a0
        moveq  #00,d0
        moveq  #32,d1
        CALL   B12_FadeToWhite
        tst.w  d0
        bne.s  Fade

INPUTS  GameScreen - Ptr to an initialised GameScreen structure.
  StartCol - The colour to start the fade from.
  AmtCols - The amount of colours to fade from StartCol.

RESULT  FadeState - Returns NULL if the fade has finished.

SEE ALSO
  B24_FadeToWhite
```

1.44 games.library/B12_FadeToPalette

```
games.library/ B12_FadeToPalette

NAME{ub}  B12_FadeToPalette -- Fade the current palette to another palette.

SYNOPSIS
  FadeState = B12_FadeToPalette(GameScreen, Palette, StartCol, AmtCols)
     d0                            a0         a1        d0       d1

  UWORD B12_FadeToPalette(struct GameScreen *, APTR Palette,
                          UWORD StartCol, UWORD AmtCols);
```

```
FUNCTION
  This  is  what  some  may call a "palette morph" function.  It will
  take  the  given  screen's  internal palette and fade it to the one
  given  in Palette(a1).  This function is quite useful for fading in
  from black screens.

INPUTS  GameScreen - Ptr to an initialised GameScreen structure.
  Palette - Ptr to a valid palette (colour array).

RESULT  FadeState - Returns NULL if the fade has finished.

SEE ALSO
  B24_FadeToPalette
```

1.45 games.library/B12_FadeToColour

```
games.library/ B12_FadeToColour

NAME  B12_FadeToColour -- Fade all the colours in a screen to a single
          colour value.

SYNOPSIS
  FadeState = B12_FadeToColour(GameScreen, RGB)
     d0                           a0       d0

  UWORD B24_FadeToColour(struct GameScreen *, ULONG RRGGBB);

FUNCTION
  Fades  the  colours  in  the  given screen to a single colour type.
  Once  you  call this function, you have to keep on calling it until
  it  gives  you  a  result  of  NULL.   This  allows you to put this
  function in a loop and do other things while the fade is active.

INPUTS  GameScreen - Ptr to an initialised GameScreen structure.
  RGB - The colour to fade to, in Red-Green-Blue format.

RESULT  FadeSate - Returns NULL if the fade has finished.

SEE ALSO
  B24_FadeToColour
```

1.46 games.library/24BIT_FadeToBlack

```
games.library/ 24BIT_FadeToBlack

NAME  B24_FadeToBlack -- Fade all the colours in a screen to black.

SYNOPSIS
  FadeState = B24_FadeToBlack(GameScreen, Speed)
     d0                           a0         d0

  UWORD B24_FadeToBlack(struct GameScreen *, UWORD Speed)
```

```
FUNCTION
  Fades  all  the colours in the specified screen to black.  Once you
  call  this  function, you have to keep on calling it until it gives
  you  a  result  of NULL.  This allows you to put this function in a
  loop and do other things while the fade is active.

INPUTS  GameScreen - Ptr to an initialised GameScreen structure.
  Speed - Determines  the  rate  at which the fade will execute.  The
    higher the value, the less you will need to call this routine.

RESULT  FadeSate - Returns NULL if the fade has finished.

SEE ALSO
  B12_FadeToBlack
```

1.47 games.library/24BIT_FadeToWhite

```
games.library/ B24_FadeToWhite

NAME  B24_FadeToWhite -- Fade all the colours in a screen to white.

SYNOPSIS
  FadeState = B24_FadeToWhite(GameScreen)
     d0                          a0

  UWORD B24_FadeToWhite(struct GameScreen *)

FUNCTION
  NOT IMPLEMENTED YET.

INPUTS  GameScreen - Ptr to an initialised GameScreen structure.

RESULT  FadeSate - Returns NULL if the fade has finished.

SEE ALSO
  B12_FadeToWhite
```

1.48 games.library/B24_FadeToPalette

```
games.library/ B24_FadeToPalette

NAME  B24_FadeToPalette -- Fade a screen palette to a new set of values.

SYNOPSIS
  FadeState = B24_FadeToPalette(GameScreen, Palette)
     d0                            a0         a1

  UWORD B24_FadeToPalette(struct GameScreen *, APTR Palette)

FUNCTION
  This  is  what  some  may call a "palette morph" function.  It will
```

```
  take  the  given  screen's  internal palette and fade it to the one
  given  in Palette(a1).  This function is quite useful for fading in
  from black screens.

INPUTS  GameScreen - Ptr to an initialised GameScreen structure.
  Palette - Ptr to a 24 bit palette with the same amount of colours
    as are in the screen.

RESULT  FadeSate - Returns NULL if the fade has finished.

SEE ALSO
  B12_FadeToPalette
```

1.49 games.library/B24_FadeToColour

```
games.library/ B24_FadeToColour

NAME  B24_FadeToColour -- Fade a screen palette to a specific colour.

SYNOPSIS
  FadeState = B24_FadeToColour(GameScreen, Colour)
     d0                              a0        d0

  UWORD B24_FadeToColour(struct GameScreen *, Colour)

FUNCTION

  NOT IMPLEMENTED YET.

INPUTS  GameScreen - Ptr to an initialised GameScreen structure.
  Colour - A 24Bit colour, ie $00RRGGBB format.

RESULT  FadeSate - Returns NULL if the fade has finished.

SEE ALSO
  B12_FadeToColour
```

1.50 games.library/Change_Colours

```
games.library/Change_Colours

NAME  Change_Colours -- Change a set of colours in a GameScreen's internal
                     palette.

SYNOPSIS
  Change_Colours(GameScreen, Colours, StartColour, AmtColours)
                   a0         a1         d0           d1

  void Change_Colours(struct GameScreen *, APTR Colours,
                        ULONG StartColour, ULONG AmtColours).

FUNCTION
```

```
  Changes  all  colours within the set range.  No alterations will be
  made  to  your  external palette (SS_Palette) by this function, all
  changes are internal.

  It  is  important to only use 12 bit or 24 bit colours as indicated
  in your GameScreen structure.

INPUTS  GameScreen - Ptr to an initialised GameScreen structure.

  Colours - Ptr  to  a list of colours, either 12 bit or 24 depending
    on screen type.

  StartColour - The  first  colour to be affected by the change.  NB:
    The first colour is defined as 0.

  AmtColours - The amount of colours to be affected by the change.
    Must be at least 1.
```

1.51 games.library/Blank_Colours

```
games.library/Blank_Colours

NAME  Blank_Colours -- Drives all screen colours to zero (black).

SYNOPSIS
  Blank_Colours(GameScreen)
                  a0

  void Blank_Colours(struct GameScreen *)

FUNCTION
  Drives  all  the colours to zero, which should give a black screen.
  You won't be able to see any pixels after calling this routine.

INPUTS  GameScreen - Ptr to an initialised GameScreen structure.
```

1.52 games.library/Init_RasterList

```
games.library/Init_RasterList

NAME  Init_RasterList -- Initialise a new rasterlist.

SYNOPSIS
  ErrorCode = Init_RasterList(GameScreen)
     d0                          a0

  UWORD Init_RasterList(struct GameScreen *)

FUNCTION
  Initialises  a  new rasterlist in a GameScreen structure.  You have
  to make sure that SS_RasterList points to a valid rasterlist before
  calling this function.
```

A rasterlist is a group of commands executed at specific areas of the display. On current Amiga's, rasterlists are executed by the copper (copperlist's) at preset lines on the screen. When you call this function a copperlist will be set up according to the commands you give in your rasterlist structure. In the past creating copperlists was a major compatibility concern because you need to pass the copper direct hardware addresses. Thankfully with the Games.Library this is no longer such a problem.

There is still the issue of gfx boards not having a copper style chip on them. Luckily all these commands can in some way be emulated, so all is not lost on that front. All instructions work perfectly on OCS, ECS and AGA of course.

Current valid commands are:

WAITLINE <Line>
Waits for the vertical beam to reach the specified screen position. It is perfectly legal to enter numbers that go outside of your screens Y limits (ie negative numbers and numbers greater than the screen height).

The <Line> parameter is the vertical position on the screen where the command will be executed. This is always specified in lo-res pixels, even if the screen is in hi-res. Although line 0 starts at the top of your screen, negative numbers are still allowed, as are numbers that exceed the screen height.

COL12 <ColNum>,<RGB>
Changes a 12 bit colour value to another. Stick within the range

COL24 <ColNum>,<RRGGBB>
Same as the COL12 command, but uses 24 bit colours. Do not use this command unless you have set the _24BITCOL flag in SS_ScrAttrib.

COL12LIST <Line>,<Skip>,<ColNum>,<RGB>
Allows you to generate the classic coloured lines used by games and demos everywhere. This command is mostly useful for sky/background effects, although you could probably use it for all sorts of things.

COL24LIST <Line>,<Skip>,<ColNum>,<RRGGBB>
Allows you to generate the classic coloured lines used by games and demos everywhere. This command is mostly useful for sky/background effects, although you could probably use it for all sorts of things. Do not use this command unless you have set the _24BITCOL flag in SS_ScrAttrib.

SPRITE <SpriteStruct>
Re-activates a sprite bank at the specified line. This is commonly known as sprite-splitting. This function is considered "dangerous" and will not work on many gfx boards (although emulation is a certain possibility).

REPOINT <Bitmap>

```
  Repoints  the  screen bitmap to another area in chip ram, causing a
  screen split at the point that this command is executed.

  SCROLL <Offset>
  Alters the scroll position of a bitplane to within 16 pixels.  This
  is really only useful for scrolling parallax landscapes.

  FSCROLL <Offset1>,<Offset2>
  Alters  the  scroll position of a bitplane to within 16 + 4 quarter
  pixels.    This  is  really  only  useful  for  scrolling  parallax
  landscapes.

  FLOOD
  A  special  effect  that  reverses the bitplane modulo, causing the
  bitplane  to  repeat itself.  This effect is used as a novel way of
  "fading in" the screen.

  MIRROR
  Similar  to  Flood,  but  does a complete reversal of the modulo so
  that  the  bitplane is "flipped over".  See examples/AGAMirror.s to
  see how this works.

  RASTEND
  You must terminate your rasterlist with this command.

  [If you have any other ideas for commands, mail me.  - Paul]

INPUTS
  GameScreen - Ptr to an initialised GameScreen structure.
  SS_RasterList  in  this  structure must contain a ptr to a standard
  rasterlist.

  Look  at  the  examples  in this package to help you with designing
  your rasterlists.

RESULT  ErrorCode - Is NULL if the initialisation was successful.  Otherwise
    it will return one of the following values:

          ERR_NOMEM = No memory could be allocated for

    ERR_NOPTR = You didn't put a ptr in SS_RasterList.

    ERR_INUSE = A rasterlist is still in use by this screen (remove
                the old one).

SEE ALSO
  Update_RasterList, Show_RasterList, Hide_RasterList, Remove_RasterList,
  games/games.i
```

1.53 games.library/Update_RasterList

```
games.library/Update_RasterList

NAME  Update_RasterList -- Update an existing rasterlist.
```

```
SYNOPSIS
  Update_RasterList(GameScreen)
                        a0

  void Update_RasterList(struct GameScreen *)

FUNCTION
  Completely  update  a  raster  list's commands and line pointers to
  whatever  values  they  may  hold.   The  length of time to do this
  depends  on  how  big your rasterlist is (generally, it will do the
  update very fast though).

  If only want to update the lines or the command datas, then you can
  call Update_RastCommands or UpdateRastLines, which is a bit faster.

INPUTS  GameScreen - Ptr to an initialised GameScreen structure.

SEE ALSO
  Init_RasterList, Show_RasterList, Hide_RasterList, Remove_RasterList,
  Update_RastCommands, Update_RastLines, games/games.i
```

1.54 games.library/Remove_RasterList

```
games.library/Remove_RasterList

NAME  Remove_RasterList -- Hide and delete RasterList from memory.

SYNOPSIS
  Remove_RasterList(GameScreen)
                        a0

  void Remove_RasterList(struct GameScreen *)

FUNCTION
  Remove  a  rasterlist's  copper  set  up  from memory.  It will not
  affect RasterList data, but will clear the SS_RasterList pointer.

  Once  this  function is called the rasterlist is gone - if you want
  to redisplay your rasterlist, reinitialise it with Init_RasterList.

  Remember  that  if  you  want  to re-initialise the same rasterlist
  later on you will have to repoint SS_RasterList.

INPUTS  GameScreen - Ptr to an initialised GameScreen structure.

SEE ALSO
  Init_RasterList, Show_RasterList, Hide_RasterList, Remove_RasterList,
  games/games.i
```

1.55 games.library/Hide_RasterList

```
games.library/Hide_RasterList

NAME  Hide_RasterList -- Hide a rasterlist from the display.

SYNOPSIS
  Hide_RasterList(GameScreen)
                  a0

  void Hide_RasterList(struct GameScreen *)

FUNCTION
  Hides a rasterlist from the screen display.  This function does not
  delete an initialised raster or change the GameScreen structure in
  any way.  You can return the list to the display simply by calling
  Show_RasterList().

NOTE  There  is  a VBL delay to prevent the rasterlist from being removed
  when the beam is still executing its instructions.

INPUTS  GameScreen - Ptr to an initialised GameScreen structure.

SEE ALSO
  Init_RasterList, Remove_RasterList, Show_RasterList, Hide_RasterList,
  Update_RasterList
```

1.56 games.library/Show_RasterList

```
games.library/Show_RasterList

NAME  Show_RasterList -- Display a rasterlist on screen.

SYNOPSIS
  Show_RasterList(GameScreen)
                  a0

  void Show_RasterList(struct GameScreen *)

FUNCTION
  Display a rasterlist on the screen.  The ptr to the rasterlist must
  lie  in  SS_RasterList  and  be initialised, either from a previous
  call to Add_Screen() or Update_RasterList()??

INPUTS  GameScreen - Ptr to an initialised GameScreen structure.

SEE ALSO
  Init_RasterList, Hide_RasterList, Show_RasterList, Remove_RasterList,
  Update_RasterList
```

1.57 games.library/Init_Sprite

```
games.library/Init_Sprite
```

```
NAME  Init_Sprite -- Initialise a sprite structure.

SYNOPSIS
  Init_Sprite(GameScreen,SpriteStruct)
                         a1

  void Init_Sprite(struct GameScreen *,struct SpriteStruct *)

FUNCTION
  Initialises  a  sprite  ready  for  placement on the screen.  After
  calling this function you  can  use  sprite  functions such as
  Update_Sprite, Move_Sprite etc.

NOTE  Colour  bank  selections  are not permitted under ECS/OCS chipsets,
  so please use the default start colour of 16 if you can.

  Under  AGA  if any 16 colour sprites exist then all sprites use the
  same colour bank.

INPUTS  GameScreen - Ptr to an initialised GameScreen structure.
  SpriteStruct - Looks like this:

   STRUCTURE  SpriteStruct,0
  UWORD SPR_Number            ;Sprite number.
  APTR  SPR_Address           ;Pointer to Sprite graphic.
  WORD  SPR_XPos              ;X position (screen relative).
  WORD  SPR_YPos              ;Y position (screen relative).
  UWORD SPR_Frame             ;Current frame number.
  UWORD SPR_Width             ;Width in pixels.
  UWORD SPR_Height            ;Height in pixels.
  UWORD SPR_AmtColours        ;4/16
  UWORD SPR_ColStart          ;000/016/032/064/096/128/160/192/224
  UWORD SPR_Planes            ;2/4
  UWORD SPR_Resolution        ;HIRES/LORES/SHIRES/XLONG
  ULONG SPR_SpriteSize        ;Reserved.
  ULONG SPR_FrameSize         ;Reserved.

SEE ALSO
  Move_Sprite, Update_Sprite, Update_SpriteList, Remove_SpriteList,
  games/games.i
```

1.58 games.library/Update_Sprite

```
games.library/Update_Sprite

NAME  Update_Sprite -- Place a sprite on the screen.

SYNOPSIS
  Update_Sprite(GameScreen, SpriteStruct)

  void Update_Sprite(struct GameScreen *, struct SpriteStruct *)

FUNCTION
  Updates  the sprite co-ordinates (screen location) and recalculates
```

```
  the sprite image pointers for animation.

INPUTS  GameScreen - Ptr to an initialised GameScreen structure.
  SpriteStruct - Ptr to an initialised SpriteStruct.

SEE ALSO
  Init_Sprite, Move_Sprite
```

1.59 games.library/Move_Sprite

```
games.library/Move_Sprite

NAME  Move_Sprite -- Move a sprite to a new screen location.

SYNOPSIS
  Move_Sprite(GameScreen, SpriteStruct)
                a0            a1

  void Move_Sprite(struct GameScreen *, struct SpriteStruct *)

FUNCTION
  Moves  a  sprite  to a new screen location according to the X and Y
  co-ordinates  found in the SpriteSruct.  This function does not act
  on  any  other SpriteStruct values and is intended for non-animated
  sprites.

NOTES On  graphics  hardware  where sprites are not supported, the sprite
  may be drawn to screen as a BOB.

  There  is  no  Move_SpriteList  available  as  static sprites are a
  rarity in games.

INPUTS  GameScreen - Ptr to an initialised GameScreen structure.
  SpriteStruct - Ptr to an initialised SpriteStruct.

SEE ALSO
  Init_Sprite, Update_Sprite
```

1.60 games.library/Remove_Sprite

```
games.library/Remove_Sprite

NAME  Remove_Sprite -- Remove a sprite from the screen display.

SYNOPSIS
  Remove_Sprite(GameScreen, SpriteStruct)
                  a0            a1

  void Remove_Sprite(struct GameScreen *, struct SpriteStruct *)

FUNCTION
  Removes a sprite from the screen display.
```

```
INPUTS  GameScreen - Ptr to an initialised GameScreen structure.
  SpriteStruct - Ptr to an initialised SpriteStruct.

SEE ALSO
  Remove_SpriteList
```

1.61 games.library/Update_SpriteList

```
games.library/Update_SpriteList

NAME  Update_SpriteList -- Update a list of initialised sprites.

SYNOPSIS
  Update_SpriteList(GameScreen, SpriteList)
                        a0          a1

  void Update_SpriteList(struct GameScreen *, APTR SpriteList *)

FUNCTION
  Update  a  series of initialised sprites according to a SpriteList.
  This  function  is  provided  as  an alternative to making constant
  calls to Update_Sprite(), which can be quite time consuming.

INPUTS  GameScreen - Ptr to an initialised GameScreen structure.
  SpriteStruct - Ptr  to  a  SpriteList  containing a list of up to 8
  initialised sprites.  The list must be terminated by a -1, eg:

  SpriteList:
    dc.l   SpriteStruct1
    dc.l   SpriteStruct2
    dc.l   SpriteStruct3
    dc.l   SpriteStruct4
    dc.l   -1

SEE ALSO
  Update_Sprite
```

1.62 games.library/Remove_SpriteList

```
games.library/Remove_SpriteList

NAME  Remove_SpriteList -- Remove sprites as specified by a SpriteList.

SYNOPSIS
  Remove_SpriteList(GameScreen, SpriteList)
                        a0          a1

  void Remove_SpriteList(struct GameScreen *, APTR SpriteList *)

FUNCTION
  Remove  a  series  of  currently displayed sprites from the screen.
```

```
  This  function  is  provided  as  an alternative to making constant
  calls to Remove_Sprite(), which can be quite time consuming.

INPUTS  GameScreen - Ptr to an initialised GameScreen structure.
  SpriteStruct - Ptr  to  a  SpriteList  containing a list of up to 8
  initialised sprites.  The list must be terminated by a -1, eg:

  SpriteList:
    dc.l   SpriteStruct1
    dc.l   SpriteStruct2
    dc.l   SpriteStruct3
    dc.l   SpriteStruct4
    dc.l   -1

SEE ALSO
  Remove_Sprite
```

1.63 games.library/Remove_AllSprites

```
games.library/Remove_AllSprites

NAME  Remove_AllSprites -- Remove all sprites from the display.

SYNOPSIS
  Remove_AllSprites(GameScreen)
                        a0

  void Remove_AllSprites(struct GameScreen *)

FUNCTION
  Temoves  all  displayed  sprites from the screen simply by altering
  the  DMACONtroller.   This is the fastest way to remove all sprites
  from the display quickly and easily.

NOTE  After  you  have  called  this  function you cannot see any visible
  changes to sprites until you call Return_AllSprites().

INPUTS  GameScreen - Ptr to an initialised GameScreen structure.

SEE ALSO
  Return_AllSprites
```

1.64 games.library/Return_AllSprites

```
games.library/Return_AllSprites

NAME  Return_AllSprites -- Return all sprites to the display.

SYNOPSIS
  Return_AllSprites(GameScreen)
                        a0
```

```
  void Return_AllSprites(struct GameScreen *)

FUNCTION
  Returns all sprites that were previously removed by Remove_All
  Sprites().



INPUTS  GameScreen - Ptr to an initialised GameScreen structure.

SEE ALSO
  Remove_AllSprites
```

1.65 games.library/Init_BOB

```
games.library/Init_BOB

NAME  Init_BOB -- Initialise a BOB structure.

SYNOPSIS
  Init_BOB(GameScreen, BOBStruct)
              a0         a1

  void Init_BOB(struct GameScreen *, APTR BOBStruct)

FUNCTION
  !DO NOT USE ANY BLITTER FUNCTIONS YET!

INPUTS  GameScreen - Ptr to an initialised GameScreen structure.
  BOBStruct - Ptr to a BOB structure.

SEE ALSO
  Init_BOBList
```

1.66 games.library/Init_BOBList

```
games.library/Init_BOBList

NAME  Init_BOBList -- Initialise a list of BOB structures.

SYNOPSIS
  Init_BOBList(GameScreen, BOBList)
                  a0         a1

  void Init_BOBList(struct GameScreen *, APTR BOBList)

FUNCTION
  !DO NOT USE ANY BLITTER FUNCTIONS YET!

INPUTS  GameScreen - Ptr to an initialised GameScreen structure.
  BOBList - Ptr to a BOBList, looks like this.  The list must be terminated by a  ↩
     -1, eg:
```

```
  BOBList:
    dc.l   BOBStruct1
    dc.l   BOBStruct2
    dc.l   BOBStruct3
    dc.l   BOBStruct4
    dc.l   -1

SEE ALSO
  Init_BOB
```

1.67 games.library/Blit_BOB

```
games.library/Blit_BOB

NAME  Blit_BOB -- Blit a BOB to a GameScreen display.

SYNOPSIS
  Blit_BOB(GameScreen, BOBStruct)
                    a0          a1

  void Blit_BOB(struct GameScreen *, struct BOBStruct *)

FUNCTION
  !DO NOT USE ANY BLITTER FUNCTIONS YET!

INPUTS  GameScreen - Ptr to an initialised GameScreen structure.
  BOBStruct - Ptr to an initialised BOB structure.

SEE ALSO
  Init_BOB, Init_BOBList, Blit_BOBList
```

1.68 games.library/Blit_BOBList

```
games.library/Blit_BOBList

NAME  Blit_BOBList -- Blit to a GameScreen from a list of BOB structures.

SYNOPSIS
  Blit_BOBList(GameScreen, BOBList)
                   a0        a1

  void Blit_BOBList(struct GameScreen *, APTR BOBList)

FUNCTION

  !DO NOT USE ANY BLITTER FUNCTIONS YET!

INPUTS  GameScreen - Ptr to an initialised GameScreen structure.
  BOBList - Ptr to a BOBList containing a list of ptrs to BOBStructs.
  The list must be terminated by a -1, eg:
```

```
  BOBList:
    dc.l   BOBStruct1
    dc.l   BOBStruct2
    dc.l   BOBStruct3
    dc.l   BOBStruct4
    dc.l   -1

SEE ALSO
  Init_BOB, Init_BOBList, Blit_BOB
```

1.69 games.library/

```
games.library/

NAME

SYNOPSIS

FUNCTION

INPUTS

SEE ALSO
```

1.70 games.library/

```
games.library/

NAME

SYNOPSIS

FUNCTION

INPUTS

SEE ALSO
```

1.71 games.library/

```
games.library/

NAME

SYNOPSIS

FUNCTION

INPUTS
```

```
SEE ALSO
```

1.72 games.library/

```
games.library/

NAME

SYNOPSIS

FUNCTION

INPUTS

RESULT

SEE ALSO
```

1.73 games.library/

```
games.library/

NAME

SYNOPSIS

FUNCTION

INPUTS

RESULT

SEE ALSO
```

1.74 games.library/AllocAudio

```
games.library/AllocAudio

NAME  AllocAudio -- Attempt to allocate the audio channels.

SYNOPSIS
  ErrorCode = AllocAudio()
     d0

  UWORD AllocAudio(void)

FUNCTION
  Attempts  to  allocate all the audio channels for your own use.  If
  the  function  cannot  get  the  channels,  it  will return with an
  errorcode  of ERR_INUSE.  If the call is successful (NULL) then you
```

```
  can safely use all the sound functions within the games.library.

  This function should be called at the start of your program, and if
  successful you must call FreeAudio() before your program exits.

RESULT  ErrorCode - NULL if successful or ERR_INUSE if unsuccessful.

SEE ALSO
  FreeAudio
```

1.75 games.library/FreeAudio

```
games.library/FreeAudio

NAME  FreeAudio -- Free the audio channels for system use.

SYNOPSIS
  FreeAudio()

  void FreeAudio(void)

FUNCTION
  Frees  the  audio  channels  so that the system can use them again.
  You cannot make use of any of the audio channels after calling this
  function.

SEE ALSO
  AllocAudio
```

1.76 games.library/InitSound

```
games.library/InitSound

NAME  InitSound -- Initialise a sound structure that is not of type RAW.

SYNOPSIS
  ErrorCode = InitSound(SoundStruct)
     d0

  UWORD InitSound(APTR SoundStruct)

FUNCTION
  This  function is mostly here for convenience.  It is not necessary
  to  call  this function for every sound that you wish to use.  But,
  if  you  are  loading  IFF sounds from disk it is quite wise to use
  this  function.   This would enable the user to change sounds as he
  sees fit without affecting your program.

  Once  called this function will fill in the following fields in the
  SoundStruct, BUT only if they have no values in them (ie they equal
  zero).
```

```
  SAM_Length
  SAM_Period
  SAM_Volume

  If  you  don't want some or all of these fields written too, simply
  fill  them  in  before-hand.   Similarly you should ensure that the
  fields that do need to be filled out, are initialised to zero.

NOTE  If  the  sound  is  in  RAW format, then this function will have no
  effect (ie you have to set up all the fields yourself).

INPUTS  SoundStruct - Ptr to a valid sound structure.

SEE ALSO
  InitSoundList
```

1.77 games.library/CheckChannel

```
games.library/CheckChannel

NAME  CheckChannel -- Checks the current activity of a sound channel.

SYNOPSIS
  Status = CheckChannel(Channel)
    d0                  d0.w

  UWORD CheckChannel(UWORD Channel)

FUNCTION
  Checks  the  specified  channel  to  see if it has any data playing
  through  it.   If it does, then the function will return a value of
  NULL.

INPUTS  Channel - Either 1, 2, 3 or 4.

RESULT  Status - The  current  status  of  the  channel,  a  result of NULL
    indicates  that  the channel is free.  Any other result indicates
    that the channel is busy.
```

1.78 games.library/PlaySound

```
games.library/PlaySound

NAME  PlaySound -- Play a sound through an audio channel.

SYNOPSIS
  PlaySound(SoundStruct)
               a1

  void PlaySound(APTR SoundStruct)

FUNCTION
```

```
  Plays  a  sound  according  to the settings in the sound structure.
  This  function  executes immediately, and ignores all channel/sound
  priorities.

INPUTS  SoundStruct - Ptr to a valid sound structure.

SEE ALSO
  PlaySoundDACx, PlaySoundPri, PlaySoundPriDACx
```

1.79 games.library/PlaySoundDACx

```
games.library/PlaySoundDACx

NAME  PlaySoundDACx -- Play a sound ignoring the setting in SAM_Channel.

SYNOPSIS
  PlaySoundDACx(SoundStruct)
                    a0

  void PlaySoundDACx(APTR SoundStruct)

  Where  'x'  is  either 1, 2, 3 or 4, which is a direct reference to
  the channel number.

FUNCTION
  DAC  stands  for Direct Access to Channel.  This is the fastest way
  to  play  a  sound  as  it  goes  directly  to that channel's sound
  routine,  but  it  is  not  very  easy to work with.  This function
  exists  for  intelligently  changing  from  full channel access for
  sound effects, to one channel access while music is playing.

  When  setting  up your sounds you should make sure that you use all
  four  channels  in  your  structures.  If the music is off, use the
  normal PlaySoundPri() function.  If the music is on, and if it uses
  all  but one of the channels, use this function to re-route all the
  sound effects through the spare channel.

NOTE  This  function  ignores  sound  priorities, and will play the sound
  regardless of wether the channel is busy or not.

INPUTS  SoundStruct - Ptr to a valid sound structure.

SEE ALSO
  PlaySound, PlaySoundPri, PlaySoundPriDACx, games/games.i
```

1.80 games.library/PlaySoundPriDACx

```
games.library/PlaySoundPriDACx

NAME  PlaySoundPriDACx -- Play a sound ignoring the setting in SAM_Channel.

SYNOPSIS
```

```
  PlaySoundPriDACx(SoundStruct)
                   a0

  void PlaySoundPriDACx(APTR SoundStruct)

  Where 'x' is either 1, 2, 3 or 4, which is a direct reference to
  the channel number.

FUNCTION
  DAC stands for Direct Access to Channel. This is the fastest way
  to play a prioritised sound as it goes directly to that channel's
  sound routine, but it is not very easy to work with. This function
  exists for intelligently changing from full channel access for
  sound effects, to one channel access while music is playing.

  When setting up your sounds you should make sure that you use all
  four channels in your structures. If the music is off, use the
  normal PlaySoundPri() function. If the music is on, and if it uses
  all but one of the channels, use this function to re-route all the
  sound effects through the spare channel.

INPUTS  SoundStruct - Ptr to a valid sound structure.

SEE ALSO
  PlaySoundDACx, PlaySound, PlaySoundPri, games/games.i
```

1.81 games.library/PlaySoundPri

```
games.library/PlaySoundPri

NAME  PlaySoundPri -- Play a sound if it can equal or better a channel's
    priority.

SYNOPSIS
  PlaySoundPri(SoundStruct)
               a0

  void PlaySoundPri(struct SoundStruct *)

FUNCTION
  Plays a sound according to the settings in the sound structure, IF
  it equals or betters the channel's current priority setting.

  Prioritisation of sounds allows you to play sound effects according
  to their importance. Make sure that you take care in ordering your
  sounds so that they play effectively!

INPUTS  SoundStruct - Ptr to a valid sound structure.

SEE ALSO
  PlaySound, PlaySoundPriDACx, PlaySoundDACx, games/games.i
```

1.82 games.library/

games.library/

NAME

SYNOPSIS

FUNCTION

INPUTS

RESULT

SEE ALSO