Default

COLLABORATORS

	TITLE : Default	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		December 25, 2022	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Default

1.1 games.library

```
                Name:         GAMES.LIBRARY AUTODOC
Version:      0.3 Beta.
Date:         06 September 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 within cannot be
        treated as official until this autodoc reaches version 1.0.

 GAMES.LIBRARY


                Add_InputHandler()

                Add_Interrupt()

                AllocMemBlock()

                FreeMemBlock()

                FastRandom()

                GetPicInfo()

                Init_GPI()

                LoadPic()

                QuickLoad()

                Read_Mouse()

                Read_JoyPort()

                Read_JoyStick()

                Read_JoyPad()
```

```
Read_SegaPad()

Read_Analogue()

Read_Key()

Rem_InputHandler()

Rem_Interrupt()

Remove_GPI()

SetGMSPrefs()

SetUserPri()

SlowRandom()

SmartLoad()

SmartSave()

SmartUnpack()

UnpackPic()

Wait_LMB()

Wait_Fire()

Wait_Time()
 SCREENS.GPI

Add_Screen()

Delete_Screen()

Show_Screen()

Hide_Screen()

ReturnToOS()

AutoOSReturn()

SwapBuffers()

Wait_VBL()

Wait_OSVBL()

Wait_RastLine()

Remake_Screen()

Move_Picture()
```

```
                Reset_Picture()

                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_RasterLines()

                Update_RasterCommand()

                Update_RasterCommands()

                Remove_RasterList()

                Hide_RasterList()

                Show_RasterList()

                Init_Sprite()

                Update_Sprite()

                Move_Sprite()

                Hide_Sprite()

                Update_SpriteList()

                Hide_SpriteList()
                 Remove_AllSprites()
Return_AllSprites()

BLITTER.GPI (Work in progress, Ideas please)

SOUND.GPI

                AllocAudio()
```

```
                FreeAudio()

                InitSound()

                FreeSound()

                CheckChannel()

                PlaySound()

                PlaySoundDAC1()

                PlaySoundDAC2()

                PlaySoundDAC3()

                PlaySoundDAC4()

                PlaySoundPri()

                PlaySoundPriDAC1()

                PlaySoundPriDAC2()

                PlaySoundPriDAC3()

                PlaySoundPriDAC4()
                 SetVolume()
 FadeVolume()
 InitPlayer()
 PlayMOD()
 StopPlayer()

 VECTORS.GPI
 Ideas Please!

 NETWORK.GPI
 Ideas Please!

 DEBUG.GPI
 Ideas Please!

 VOXEL.GPI?  Ideas/Code?
```

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.  If the
  function fails then it will return with NULL.

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 =  4
  GPI_SOUND   =  8
  GPI_NETWORK =  12
  GPI_VECTORS =  16
  GPI_DEBUG   =  20
  GPI_ANIM    =  24
  GPI_REKO    =  28
  GPI_TEXT    =  32

RESULT  GPIBase - Poniter 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 - Pointer 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
  ZBXY = Read_Mouse(PortName)
         d0                 d0

  ULONG Read_Mouse(UWORD PortName);

FUNCTION
  Reads  the  mouse port and returns any changes in its co-ordinates.
  The status of the mouse is returned in ZXBYStatus (a packed state).
  If  the user was not using the mouse, then ZBXYStatus will return a
  NULL value.

  The first time you call this function it may return nonsense values
  in  the  X/Y  directions.   Therefore  you  must  call  it  in  the
  initialisation  section of your program before using it in the rest
  of your program.

  This function also requires that the input handler has already been
  installed by GMS (Show_Screen() will do this for you).

  JoyPorts 3 and 4 are not supported by this function.

EXAMPLE If you are having trouble unpacking the ZBXYStatus value in C, here
  is some code to get the X, Y and Z values.

        XPos += (BYTE)(ZBXY>>8);
        YPos += (BYTE)ZBXY;
        ZPos += (BYTE)(ZBXY>>24);

  To read the left mouse button:

  if (ZBXY&MB_LMB) {
     /* LeftMouse pushed... */
  }

INPUT PortName = JPORT1 or JPORT2.
```

```
RESULT  ZBXY - Contains changes in direction and button states.

        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)

  Button status bits are:

  MB_LMB = 16
  MB_RMB = 17
  MB_MMB = 18

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, Analogue JoySticks,
  SegaPads, CD32 JoyPads, the mouse, and the keyboard.

  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

  If  using  JT_ZBXY,  the  first  time you call this function it may
  return  nonsense  values in the X/Y directions.  Therefore you must
  call  it in the initialisation section of your program before using
  it in the rest of your program.

SEE ALSO
  Read_Mouse, Read_JoyStick, Read_JoyPad, Read_SegaPad, Read_Analogue,
  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
  JoyBits = 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 JoyBits will
  return a NULL value.

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

INPUTS  PortName - JPORT1, JPORT2, JPORT3 or JPORT4.
```

```
RESULT  JoyBits - 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
  JS_FIRE3  =  8

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

1.7 games.library/Read_Analogue

```
games.library/Read_Analogue

NAME  Read_Analogue -- Read an analogue joystick from the given port.

SYNOPSIS
        ZBXYStatus = Read_Analogue(PortName)
            d0                     d0

  ULONG Read_Analogue(UWORD PortName);

FUNCTION
  Reads  an analogue 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.

  The first time you call this function it may return nonsense values
  in  the  X/Y  directions.  Therefore  you  must  call  it  in  the
  initialisation  section of your program before using it in the rest
  of your program.

  JoyPorts 3 and 4 are not supported by this function.

EXAMPLE If you are having trouble unpacking the ZBXYStatus value in C, here
  is some code to get the X, Y and Z values.

        XPos += (BYTE)(ZBXY>>8);
        YPos += (BYTE)ZBXY;
        ZPos += (BYTE)(ZBXY>>24);

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.

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
  JoyBits = 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 JoyBits format.  If the user was not using
  the joypad, then JoyBits will return a NULL value.

INPUTS  PortName - JPORT1 or JPORT2.

RESULT  JoyBits - 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_Analogue,
  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
  JoyBits = Read_SegaPad(PortName)
     d0                  d0

  ULONG Read_SegaPad(UWORD PortName)

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

INPUTS  PortName - JPORT1 or JPORT2.

RESULT  JoyBits - 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!  Somone test it and tell me if
  it works OK.

SEE ALSO
  Read_JoyPort, Read_JoyStick, Read_JoyPad, Read_Analogue,
  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
  KeyValue = Read_Key(KeyStruct)
          d0                  a0

  UBYTE 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 KeyValue
  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.   You  can  see  what these special keys are in
  games.i.

  Qualifiers  have  automatic  effects  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 four keys since your previous check.  If you are
  calling Read_Key() every vertical blank, you are already supporting
  typing  speeds  of  an  astronomical 600 words per minute, so it is
  only  necessary  to  check  KP_Key1.  If you are only grabbing keys
  every 1/2 second, then all fields should be checked.

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_Key1              ;Newest KeyPress.
      UBYTE  KP_Key2              ;...
      UBYTE  KP_Key3              ;...
      UBYTE  KP_Key4              ;Oldest KeyPress.

RESULT  KeyValue - Contains  the  latest keypress value, ie is identical to
    KP_Key1.

  KeyStruct  - Updated to hold new key data.  You may receive as much
    as  4  keys  in  the provided fields.  Key fields containing zero
    indicate  that no key was pressed (any following fields will also
    be zero).

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

  UWORD FastRandom(UWORD Range);

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

  This  routine  will  generally get all the numbers in fairly random
  sequences.  However do not use it in fast-running loops, ie:
```

```
  for (i=0; i<100; i++) {
      Array[i] = FastRandom(50);
  }

  You  will  not get very good numbers unless you use SlowRandom() in
  the above case.

  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 excellent 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
  Waits until the horizontal beam reaches the exact start of the VBL.
  Even if you move your screen around using Remake_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
  Waits until the horizontal beam reaches the exact start of the VBL.
  Even  if  you move your screen around using Move_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(WORD LineNumber)

FUNCTION
  Waits  for  the  strobe  to  reach  the  scan-line  specified  in
  LineNumber.    The   recognised  range  is  dependent  on  the  low
  resolution  height  of your screen, eg 0-256 for a standard 320x256
  screen.  It is permissable to enter negative values and values that
  exceed this range, but only do so if absolutely necessary.

  This  function  has  been  specially  written  to avoid beam misses
  caused by the untimely activation of interrupts.

INPUTS  LineNumber - Vertical beam position to wait for.

BUGS  If  you  enter  a  large value that is well beyond the range limit,
  like  #350,  the strobe will never reach this line because line 350
  doesn't  even  exist.   This  will  cause  your program to lock up.
  Please keep your values restricted to the height of your screen.

SEE ALSO
  Wait_OSVBL, Wait_VBL.
```

1.19 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 any of the Read functions (eg Read_Key())
  then  it  is  vital  that this function is active.  This is because
  some  of  the  Read  functions  are  hooked  into the input handler
  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
```

1.20 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 certain Read functions (eg Read_Key()).

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

SEE ALSO
  Add_InputHandler
```

1.21 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 - Pointer  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
```

1.22 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 - Pointer to an interrupt base returned from Add_Interrupt().

SEE ALSO
  Add_Interrupt, games.i
```

1.23 games.library/SmartLoad

```
games.library/SmartLoad

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

SYNOPSIS
  MemLocation = SmartLoad(FileName, Destination, Password, MemType)
      d0                     a0          a1         d0       d1

  ULONG SmartLoad(char *FileName, APTR Destination, ULONG Password,
                  ULONG MemType)

FUNCTION
  Loads  in  a  file  and  depacks  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.

  SmartLoad()  is  written  to  be  as  intelligent  as possible when
  loading  the  file.   This  includes keeping memory usage as low as
  possible,  and  searching  the  current directory for a file if any
  disk  assignment  cannot be found.  Future revisions of SmartLoad()
  are likely to contain more of these types of intelligent features.

  Currently supported packers are XPK (external), PowerPacker (inter-
```

```
  nal)  and  RNC (internal).  The recommended packing method for your
  files  is  the  traditional  RNC packer, which does not require any
  extra buffers for unpacking.

  Files  packed  with  XPK  require  the  xpkmaster.library  and  the
  relevant  compressor  in  your  LIBS:  directory, if the file is to
  unpack.  Keep this in mind when distributing your game.

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

  If  you  give  the  Destination  for  the  file then the MemType is
  ignored.

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.
  Password - If the file is encrypted, supply a key here.
  MemType - Memory Type (only required if Destination is NULL)

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

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

1.24 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(char *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, and
  has  limited  directory  searching  as  in SmartLoad(), if the file
  cannot immediately be found.

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

  If  you  give  the  Destination  for  the  file then the MemType is
  ignored.

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
```

1.25 games.library/SmartUnpack

```
games.library/SmartUnpack

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

SYNOPSIS
  MemLocation = SmartUnpack(Source, Destination, Password, MemType)
           d0                       a0        a1         d0        d1

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

FUNCTION
  Attempts  to unpack a data area if it can assess the packing method
  used.   The  data  should begin with an ID longword followed by the
  size  of  the  original data before it was packed.  The data itself
  must  follow directly after this.  Any packer that does not do this
  will not be supported by this function.

  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,  the  MemType is
  ignored.

  This  function  currently  supports  XPK  (external)  and  the  RNC
  (internal)  packer  types.  The RNC packer can unpack directly over
  itself  (ie  Source  and  Destination can be the same).  Do not try
  this with the XPK packer - it won't work!

NOTE  Remember   to   free   any   memory   returned   in   MemLocation   with
  FreeMemBlock() if you wanted the allocation.

INPUTS  Source - Ptr to start of packed data (must be an ID header).
  Destination - Destination for unpacked data or NULL for allocation.
  Password - FileKey or NULL if none is used.
  MemType - Memory type (only supply if Destination is NULL).

RESULT  MemLocation - Ptr to the unpacked data.

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

1.26 games.library/SmartSave

```
NAME  SmartSave -- Save a file to disk using a packer algorithm.

SYNOPSIS
  ErrorCode = SmartSave(FileName, Source, SrcLength)
     d0                 a0       a1       d0

  UWORD SmartSave(char *FileName, APTR Source, ULONG SrcLength)

FUNCTION
  Saves a file to disk, and if possible, packing it before-hand.  The
  currently  supported  packing method is XPK-NUKE, but GMSPrefs will
  soon allow the user to select any XPK packing method.

INPUTS  FileName - Name of the file to save to.
  Source - Pointer to the start of the source data.
  SrcLength - Amount of data to save.

RESULT  ErrorCode - A standard GMS errorcode.  NULL indicates success.

SEE ALSO
  SmartLoad, SmartUnpack, games/games.i
```

1.27 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  the ENV:GMSPrefs file.  This
  priority setting can be altered in the GMSPrefs utility.

  This function should be used in all your programs written with GMS,
  as part of the initialisation procedure.

SEE ALSO
  exec/SetTaskPri
```

1.28 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.  This function
  is  intended  for  use  by the GMSPrefs program, there should be no
  reason for you to use it in your own game.

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.
```

1.29 games.library/LoadPic

```
games.library/Loadpic

NAME  LoadPic -- Load in a recognised picture file.

SYNOPSIS
  ErrorCode = LoadPic(FileName, Picture)
     d0                a0         a1

  ULONG LoadPic(char *FileName, struct Picture *)

FUNCTION

INPUT FileName - The picture file to load.
  Picture - Pointer to a Picture structure.

RESULT  ErrorCode - Returns NULL if successful.
```

1.30 games.library/UnpackPic

```
games.library/Loadpic

NAME  UnpackPic -- Unpack a picture to a designated buffer.

SYNOPSIS
  ErrorCode = UnpackPic(Source, Picture)
     d0                 a1      a0

  ULONG UnpackPic(APTR Source, struct Picture *)

FUNCTION
  Unpacks  the  data  contained in a recognised picture header to the
  data  destination  given  in  the Picture structure.  If you do not
```

```
  give a data destination, then the destination will be allocated for
  you and placed in SS_Data.

  If  this  function cannot identify the source header, then the call
  will fail.  Currently the only supported format is IFF, but GIF and
  JPEG support will be added later.

INPUT Source - Pointer to the header of the picture source.
  Picture - Pointer to a Picture structure.

RESULT  ErrorCode - Returns NULL if successful.
```

1.31 games.library/GetPicInfo

```
games.library/GetPicInfo

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

SYNOPSIS
  ErrorCode = GetPicInfo(Picture)
     d0                  a1

  ULONG GetPicInfo(struct Picture *)

FUNCTION
  Not implemented yet.

INPUT
  Picture - Pointer to a Picture structure.

RESULT  ErrorCode - Returns NULL if successful.
```

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
  identical to AllocVec().  It exists here because AllocVec() is only
  available  on V36+ machines.  Also it uses memory headers and tails
  so that you may successfully identify allocated memory blocks.

  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 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 also check for valid
    memory allocations by looking at the ID header.  "MEMH" is placed
    at -8(MemBlock).

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 is the most
  reliable  and  crash-proof freemem function currently on the Amiga.

  If  the  mem header or tail is missing, then it can be assumed that
  something  has written over the boundaries of your memblock, or you
  are  attempting  to  free a non-existant allocation.  Normally this
  would  cause  a  complete system crash, but instead we simply alert
  you to the fact, and you can continue on.

  It  does  pay  to  save  your work and reset your machine if such a
  message  appears,  as  it  indicates that important memory data may
  have been destroyed.

NOTE  Never free the same MemBlock twice.

INPUT MemBlock - Points to the start of a memblock.

SEE ALSO

                AllocMemBlock
                , exec/memory.i
```

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

   STRUCTURE  GameScreen,0       ;A GameScreen structure
  ULONG SS_VERSION         ;Vesion - "GSV1"
  APTR  SS_Stats           ;Reserved, do not touch.
  APTR  SS_MemPtr1         ;Ptr to screen 1
  APTR  SS_MemPtr2         ;Ptr to screen 2 (double buffer)
  APTR  SS_MemPtr3         ;Ptr to screen 3 (triple buffer)
  APTR  SS_ScreenLink      ;Ptr to a linked screen.
  APTR  SS_Palette         ;Ptr to a palette.
  APTR  SS_RasterList      ;Ptr to a raster list.
  ULONG SS_AmtColours      ;The amount of colours on screen.
  UWORD SS_ScrWidth        ;The width of the visible screen.
  UWORD SS_ScrHeight       ;The height of the visible screen.
  UWORD SS_PicWidth        ;The width of the entire screen.
  UWORD SS_PicHeight       ;The height of the entire screen.
  UWORD SS_Planes          ;The amount of planes in da screen.
  WORD  SS_ScrXOffset      ;X offset for top of screen.
  WORD  SS_ScrYOffset      ;Y offset for top of screen.
  WORD  SS_PicXOffset      ;X offset for picture.
  WORD  SS_PicYOffset      ;Y offset for picture.
  ULONG SS_ScrAttrib       ;Special Attributes.
  UWORD SS_ScrMode         ;What screen mode is it?
  UBYTE SS_ScrType         ;Interleaved/Planar/Chunky?
  UBYTE SS_Displayed       ;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 will 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 COL24BIT 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_ScrWidth, SS_ScrHeight
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.

SS_PicWidth, SS_PicHeight
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. Note that the width of the picture must be divisible by 16.

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_ScrXOffset, SS_SrcYOffset
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. A setting of 0,0 should be sufficient, unless you are going to create an extra large screen (eg overscan). Negative values are permissable.

SS_PicXOffset, SS_PicYOffset
These two fields set the offsets for the picture "behind" the

screen. If you want to do any sort of hardware scrolling, you will want to use these values in conjunction with Move_Picture(). It is perfectly legal to preset these values before you call Show_Screen().

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 horizontal picture scrolling.

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

HBUFFER - Allocates extra space to allow you to horizontally scroll up to 50 screens in both X directions.

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

BLKBDR - Turns all colours outside of the display window to black. Works on AGA only.

NOSPRBDR - Allows sprites to appear outside of the screen display window. Works on AGA only.

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 low resolution screen. This is the default, so you do not have to specify it if you don't want to.

HIRES - Specifies a hi-resolution screen.

SHIRES - Specifies a superhi-resolution screen.

INTERLACED - Creates an interlaced display.

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 dependant on the amount of planes in the screen.

```
    COL24BIT   - 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, perhaps you should 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 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, if those fields were allocated by
  Add_Screen().

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 calling this function will result in a
  complete take-over of the viewport.

  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
        GameScreen = Hide_Screen()
     d0

  APTR Hide_Screen(void)

FUNCTION
  Hides  the  currently  displayed screen from view.  This will cause
  the OS viewport to be returned, but your task will still be running
  "in the background".

  If  no  GameScreen  is present then this function does nothing, and
  returns a NULL value.

  On  its  own  this is not good for screen-switching - use functions
  like AutoOSReturn() for that.

RESULT  GameScreen - Points  to  the  structure  of the GameScreen that has
    been  hidden  by  this function.  Otherwise NULL if no GameScreen
    was active.
```

```
SEE ALSO
  Show_Screen, Add_Screen, Delete_Screen, ReturnToOS, AutoOSReturn,
  Wait_OSVBL.
```

1.38 games.library/ReturnToOS

```
games.library/ReturnToOS

NAME  ReturnToOS -- Returns the screen display to intuition.

SYNOPSIS
  ReturnToOS()

  void ReturnToOS(void)

FUNCTION
  Returns  the  screen display to intuition immediately.  Your game's
  execution  will  be  halted until the user brings your screen back.

  GMS  supports two methods of screen switching, Switch-To-Window and
  Switch-To-Screen.   The  method  used depends on the setting in the
  GMSPrefs utility.

  Switch-To-Window drops out to workbench and places a window on the
  screen.   It  will  busy-wait  until  the  close gadget is pressed,
  whereupon your game will continue where it left off.

  Switch-To-Screen opens an intution screen and busy-waits until that
  screen comes to the front.  At that point the intuition screen will
  be closed and your game will resume execution.

SEE ALSO
  AutoOSReturn, Hide_Screen, Wait_OSVBL
```

1.39 games.library/AutoOSReturn

```
games.library/AutoOSReturn

NAME  AutoOSReturn -- Returns  the  screen  display  to  intuition if the
    Left-Amiga + M key combination was pressed.

SYNOPSIS
  AutoOSReturn()

  void AutoOSReturn(void)

FUNCTION
  Returns  the  screen  display  to intuition if the user pressed the
  Left-Amiga+M key combination.  Your game's execution will be halted
  until  the  user brings your screen back.
```

```
  GMS  supports two methods of screen switching, Switch-To-Window and
  Switch-To-Screen.   The  method  used depends on the setting in the
  GMSPrefs utility.

  Switch-To-Window drops out to workbench and places a window on the
  screen.   It  will  busy-wait  until  the  close gadget is pressed,
  whereupon your game will continue where it left off.

  Switch-To-Screen opens an intution screen and busy-waits until that
  screen comes to the front.  At that point the intuition screen will
  be closed and your game will resume execution.

SEE ALSO
  ReturnToOS, Hide_Screen, Wait_OSVBL
```

1.40 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 physically change these pointers yourself.

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

1.41 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
```

```
  void Remake_Screen(struct GameScreen *)

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 - Pointer to an initialised GameScreen structure.
```

1.42 games.library/Move_Picture

```
                games.library/Move_Picture

NAME  Move_Picture -- Moves the screen to specified X/Y values.

SYNOPSIS
  Move_Picture(GameScreen)
                        a0

  void Move_Picture(struct GameScreen *)

FUNCTION
  This  routine  has two uses:  Moving the picture to any position on
  the display, and for Hardware Scrolling.

  It  will  take  the  values  from  PicXOffset and PicYOffset in the
  GameScreen  structure and use them to set the new picture position.
  It  doesn't  matter how far away the new position is, this function
  will execute at the same speed for all values.

  You  must have set the HSCROLL bit for horizontal scrolling and the
  VSCROLL  bit  for  vertical  scrolling  if  you  wish  to  use this
  function.   If  you  set the HBUFFER flag in ScrAttrib then you can
  also  use this function to legally hardware-scroll up to 50 screens
  in   either  X  direction.   Do  not  draw  graphics  beyond  these
  boundaries or your program may crash.

NOTES If  the graphics hardware does not support hardware scrolling, this
  routine will probably blit the entire picture to the new position.
  This is very slow but is the only other option.

  The  execution  time  for  this  function on ECS/AGA is 2/3rds of a
  single rasterline on my A1200+Fast.

INPUTS  GameScreen - Pointer to an initialised GameScreen structure.
    The  PicXOffset  and  PicYOffset  values  will be used to set the
    picture's new on-screen position.
```

```
SEE ALSO

                Reset_Picture
```

1.43 games.library/Reset_Picture

```
                games.library/Reset_Picture

NAME  Reset_Picture -- Resets the picture position to position 0X, 0Y.

SYNOPSIS
  Reset_Picture(GameScreen)
                  a0

  void Reset_Picture(struct GameScreen *)

FUNCTION
  Resets  the  picture  position to 0X, 0Y.  This is method is faster
  than clearing the PicXOffset and PicYOffset fields before a call to
  Move_Screen().

INPUTS  GameScreen - Pointer to an initialised GameScreen structure.

RESULT  PicXOffset and PicYOffset in the GameScreen will be cleared.

SEE ALSO

                Move_Picture
```

1.44 games.library/B12_FadeToBlack

```
                games.library/B12_FadeToBlack

NAME  B12_FadeToBlack -- Fade all colours to black.

SYNOPSIS
  FadeState = B12_FadeToBlack(GameScreen, FadeState)
     d0                          a0          d0

  UWORD B12_FadeToBlack(struct GameScreen *, UWORD FadeState)

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().

EXAMPLE FadeState = 0;
```

```
  do {
          Wait_OSVBL;
     FadeState = B12_FadeToBlack(GameScreen,FadeState);
     }
  while (FadeState != 0)

INPUTS  GameScreen - An initialised GameScreen structure.
  FadeState - Initialise  to  zero,  then  keep  sending the returned
    value back until you get a NULL in this field.

RESULT  FadeState - Returns NULL if fade has finished.

SEE ALSO

                B24_FadeToBlack
```

1.45 games.library/B12_FadeToWhite

```
                games.library/B12_FadeToWhite

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

SYNOPSIS
  FadeState = B12_FadeToWhite(GameScreen, FadeState, StartCol, AmtCols)
     d0                          a0          d0         d1       d2

  UWORD B12_FadeToWhite(struct GameScreen *, UWORD FadeState,
                        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 FadeState = 0;
  do {
          Wait_OSVBL;
     FadeState = B12_FadeToWhite(GameScreen,FadeState,00,32);
     }
  while (FadeState != 0)

INPUTS  GameScreen - Pointer to an initialised GameScreen structure.
  FadeState - Initialise  to  zero,  then  keep  sending the returned
    value back until you get a NULL in this field.
  StartCol - The colour to start the fade from.
  AmtCols - The amount of colours to fade from StartCol.

RESULT
  FadeState - Send  this  value back to the function until it returns
    NULL.
```

```
SEE ALSO

                B24_FadeToWhite
```

1.46 games.library/B12_FadeToPalette

```
                games.library/B12_FadeToPalette

NAME  B12_FadeToPalette -- Fade the current palette to another palette.

SYNOPSIS
  FadeState = B12_FadeToPalette(GameScreen, Palette, FadeState,
     d0                            a0         a1        d0
                                StartCol, AmtCols)
                                   d1       d2

  UWORD B12_FadeToPalette(struct GameScreen *, APTR Palette,
                          UWORD FadeState, 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 - Pointer to an initialised GameScreen structure.
  Palette - Ptr to a valid palette (colour array).
  FadeState - Initialise  to  zero,  then  keep  sending the returned
    value back until you get a NULL in this field.

RESULT  FadeState - Returns NULL if the fade has finished.

SEE ALSO

                B24_FadeToPalette
```

1.47 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, FadeState, RGB)
     d0                            a0          d0       d1

  UWORD B24_FadeToColour(struct GameScreen *, UWORD FadeState,
                         UWORD RGB);
```

```
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 - Pointer to an initialised GameScreen structure.
  RGB - The colour to fade to, in Red-Green-Blue format.
  FadeState - Initialise  to  zero,  then  keep  sending the returned
    value back until you get a NULL in this field.

RESULT  FadeState - Returns NULL if the fade has finished.

SEE ALSO

                B24_FadeToColour
```

1.48 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, FadeState, Speed)
     d0                          a0          d0        d1

  UWORD B24_FadeToBlack(struct GameScreen *, UWORD FadeState,
                        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 - Pointer 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.
  FadeState - Initialise  to  zero,  then  keep  sending the returned
    value back until you get a NULL in this field.

RESULT  FadeSate - Returns NULL if the fade has finished.

SEE ALSO

                B12_FadeToBlack
```

1.49 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, FadeState, Speed)
     d0                          a0          d0        d1

  UWORD B24_FadeToWhite(struct GameScreen *, UWORD FadeState,
                        UWORD Speed);

FUNCTION
  Fades  all  the colours in the 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.

INPUTS  GameScreen - Pointer to an initialised GameScreen structure.
  FadeState - Initialise  to  zero,  then  keep  sending the returned
    value back until you get a NULL in this field.

RESULT  FadeState - Initialise  to  zero,  then  keep  sending the returned
    value back until you get a NULL in this field.

SEE ALSO

                B12_FadeToWhite
```

1.50 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, FadeState, Palette, Speed)
     d0                             a0         d0        a1       d1

  UWORD B24_FadeToPalette(struct GameScreen *, UWORD FadeState,
                          APTR Palette, UWORD Speed)

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 - Pointer to an initialised GameScreen structure.
  Palette - Ptr to a 24 bit palette with the same amount of colours
    as are in the screen.
  FadeState - Initialise  to  zero,  then  keep  sending the returned
    value back until you get a NULL in this field.
```

```
RESULT  FadeState - Returns NULL if the fade has finished.

SEE ALSO
  B12_FadeToPalette
```

1.51 games.library/B24_FadeToColour

```
                games.library/B24_FadeToColour

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

SYNOPSIS
  FadeState = B24_FadeToColour(GameScreen, FadeState, Colour, Speed)
     d0                            a0        d0         d2      d1

  UWORD B24_FadeToColour(struct GameScreen *, UWORD FadeState,
                         UWORD Colour, UWORD Speed)

FUNCTION
  This  will  fade  all the colours in your screen's internal palette
  to a single 24 bit colour value.

INPUTS  GameScreen - Pointer to an initialised GameScreen structure.
  Colour     - A 24Bit colour, ie $00RRGGBB format.
  FadeState  - Initialise  to  zero,  then  keep  sending the returned
    value back until you get a NULL in this field.

RESULT  FadeState - Returns NULL if the fade has finished.

SEE ALSO

                B12_FadeToColour
```

1.52 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.  Alterations will only be
  made to the screen's internal palette.
```

```
INPUTS  GameScreen - Pointer 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.
```

1.53 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 picture detail after calling this
  routine.

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

1.54 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.  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 many of these commands can in some way be
```

emulated, so all is not lost on that front.

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 screen's vertical limits (ie negative numbers and numbers greater than the screen height), but no more than a value of 10.

Note that the purpose of this command is to specify the screen position at which the next command will be executed. All line values must be specified in lo-res pixels, regardless of your screen resolution.

COL12 <ColNum>,<RGB>
Changes a 12 bit colour value to another.

COL24 <ColNum>,<RRGGBB>
Same as the COL12 command, but uses 24 bit colours. Do not use this command unless you have set the COL24BIT 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 COL24BIT 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 may simply do nothing 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 - Pointer 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 = Not enough memory was available for one of the
                allocations.

    ERR_NOPTR = You didn't put an address pointer 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
```

1.55 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  updates  a  rasterlist's  commands  and  waitline's  to
  whatever  values SS_RasterList may now hold.  The length of time to
  do  this  depends on how big your rasterlist is (generally, it will
  do the update very fast though).

  Make sure that the new information provided is within the limits of
  your  original  values,  for example you cannot make changes to the
  amount of colours used in a NEWPALETTE command.
```

```
  If you only want to update the lines or the command datas, then you
  can call Update_RastCommands() or UpdateRastLines(), which can be a
  bit faster in certain situations.

INPUTS  GameScreen - Pointer to an initialised GameScreen structure.

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

1.56 games.library/Update_RasterLines

```
games.library/Update_RasterLines

NAME  Update_RasterLines -- Updates all the WaitLine's in an active
    rasterlist.

SYNOPSIS
  void Update_RasterLines(GameScreen)
                          a0

  void Update_RasterLines(struct GameScreen *)

FUNCTION
  Updates every occurance of a WAITLINE command in an active
  rasterlist.  This includes the update of waitline's within commands
  such as COL12LIST and COL24LIST.  All other commands are excluded
  from being updated by this function.

  This function has been provided because for other functions it can
  be unsafe to update single WAITLINE commands.  Whenever you want
  one or more raster line's updated we insist that you use this or
  the Update_RasterList() routine.

INPUTS  GameScreen - Pointer to an initialised GameScreen structure.

SEE ALSO
  Update_RasterCommand, Update_RasterCommands, Update_RasterList
```

1.57 games.library/Update_RasterCommand

```
games.library/Update_RasterCommand

NAME  Update_RasterCommand -- Update a single rasterlist command.

SYNOPSIS
  Update_RasterCommand(GameScreen, Command)
                       a0          a2

  void Update_RasterCommand(struct GameScreen *, APTR Command)
```

```
FUNCTION
  Updates a single raster command.  This is the fastest way to update
  any  single  command  in  a rasterlist.  For the update of multiple
  commands, use Update_RasterList() or Update_RasterCommands().

  You  must  never  use  this  command  to update changes in WAITLINE
  commands.  Doing  so  can  have  unpredictable  effects  on  other
  WAITLINE commands present in the screen.

INPUTS  GameScreen - Pointer to an initialised GameScreen structure.
  Command - Points to the rasterlist command to be updated.

SEE ALSO
  Update_RasterCommands, Update_RasterLines, Update_RasterList
```

1.58 games.library/Update_RasterCommands

```
games.library/Update_RasterCommands

NAME  Update_RasterCommands -- Update a group of rasterlist commands'.

SYNOPSIS
  Update_RasterCommands(GameScreen, Command, Amount)
                            a0         a2       d0

FUNCTION
  Updates a group of raster commands in a screen's active rasterlist.
  This  is  the  fastest  way  to  update a group of commands without
  having  to  do  a  complete rasterlist update.  If you only want to
  update  a  single command, use Update_RasterCommand().  For all the
  commands, use Update_RasterList().

  You  must  never  use  this  command  to update changes in WAITLINE
  commands.  Doing  so  can  have  unpredictable  effects  on  other
  WAITLINE commands present in the screen.

INPUTS  GameScreen - Pointer to an initialised GameScreen structure.
  Command - Points to the first rasterlist command to be updated.
  Amount - The amount of commands to be updated.

SEE ALSO
  Update_RasterCommand, Update_RasterLines, Update_RasterList
```

1.59 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
  Removes the memory used by the rasterlist's internal setup.  If the
  rasterlist  is  currently displayed then it will be hidden from the
  view before the deletion.

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

INPUTS  GameScreen - Pointer to an initialised GameScreen structure.

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

1.60 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  the  internal rasterlist 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 in this function so that the rasterlist is not
  removed while the beam is still executing its instructions.

INPUTS  GameScreen - Pointer to an initialised GameScreen structure.

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

1.61 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 pointer to the rasterlist
  must lie in SS_RasterList, and must have been initialised by a call
  to Init_RasterList().

INPUTS  GameScreen - Pointer to an initialised GameScreen structure.

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

1.62 games.library/Init_Sprite

```
games.library/Init_Sprite

NAME  Init_Sprite -- Initialise a sprite structure.

SYNOPSIS
  ErrorCode = Init_Sprite(GameScreen,Sprite)
     d0                   a0         a1

  ULONG Init_Sprite(struct GameScreen *,struct Sprite *)

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.

  If  it is impossible to show the sprite, then an error code will be
  returned.   In  such  a  case it helps to have a blitter routine as
  back  up,  so  that  you can instead display the sprite as a BOB on
  screen.

  Sprites  are  very  much  dependent  on the machine hardware, so be
  aware that the image may not show on some machines.

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

    STRUCTURE SpriteStruct,0
  ULONG SPR_VERSION         ;Structure version "SPV1".
  APTR  SPR_Stats           ;Reserved.
  UWORD SPR_Number          ;Sprite bank number.
  APTR  SPR_Data            ;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          ;Amt of planes per bank (2).
UWORD SPR_Resolution      ;HIRES/LORES/SHIRES/XLONG
UWORD SPR_FieldPriority   ;Playfield priority.
ULONG SPR_SpriteSize      ;Reserved.
ULONG SPR_FrameSize       ;Reserved.
LABEL SPV1_SIZEOF
```

Here follows a description of each field:

SPR_VERSION
The version of the structure. Currently this is "SPV1". 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.

SPR_Number
The bank number that this sprite is going to use.

SPR_Data
Points to the beginning of the sprite data (starts with the two control words).

SPR_XPos
Defines the horizontal position of the sprite when displayed. Negative or extreme values that put the sprite outside of the screen are permitted.

SPR_YPos
Defines the vertical position of the sprite when displayed. Negative or extreme values that put the sprite outside of the screen are permitted.

SPR_Frame
The number of the frame to display. The first frame is 0, the last frame is defined by the amount of following graphics for the sprite.

SPR_Width
The width of the sprite in pixels. Under OCS/ECS the only available range is 16 pixels. Under AGA this is extended by permission of values 32 and 64.

SPR_Height
The height of the sprite in pixels. A valid range is between 0 and 256.

SPR_AmtColours
The amount of colours used by this sprite. This will be either 4 colours or 16 colours if the sprite is to work on OCS/ECS/AGA.

SPR_ColStart
The colour bank at which the colours are going to start for this sprite. This value goes up in increments of 16, eg 0,16,32,48... Under OCS/ECS you must set this value to 16. For AGA the maximum limit is 240. Note that under current hardware conditions, all sprites must share the same colour bank. Do not attempt to set a

```
  different colour bank for each individual sprite.

  SPR_Planes
  Specifies the amount of planes used per bank.  Set this value to 2.

  SPR_Resolution
  Defines  the  display  mode  for the sprite.  Possible flags are:

    LORES      - Puts the sprite in low resolution.  (Default)

    HIRES      - Specifies a high resolution sprite.

    SHIRES     - Specifies a super-high resolution sprite.

    XLONG      - Use  this  flag  if  you  want  to  join two sprites
                 together  on  the X axis.  The  second sprite's data
                 must  follow  the  first  sprite  and  fit  the same
                 attributes.

  SPR_FieldPriority
  Defines  the  position  of  the  sprite  in  relation to the screen
  playfields.   If  set to 0 then the sprite is at the very front, if
  set to 1 then the sprite is one field behind, and so on.

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

1.63 games.library/Update_Sprite

```
                games.library/Update_Sprite

NAME  Update_Sprite -- Place a sprite on the screen.

SYNOPSIS
  Update_Sprite(GameScreen, Sprite)
                   a0         a1

  void Update_Sprite(struct GameScreen *, struct Sprite *)

FUNCTION
  Updates  the sprite co-ordinates (screen location) and recalculates
  the sprite image pointers for animation.

  This  function  cannot  make  sudden changes to the width, colours,
  resolution, or height of the sprite.

INPUTS  GameScreen - Pointer to an initialised GameScreen structure.
  Sprite - Pointer to an initialised Sprite structure.

SEE ALSO

                Init_Sprite
                ,
                Move_Sprite
```

1.64 games.library/Move_Sprite

```
                games.library/Move_Sprite

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

SYNOPSIS
  Move_Sprite(GameScreen, Sprite)
                 a0         a1

  void Move_Sprite(struct GameScreen *, struct Sprite *)

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 list support as static sprites are a rarity in games.

INPUTS  GameScreen - Pointer to an initialised GameScreen structure.
  Sprite - Pointer to an initialised Sprite structure.

SEE ALSO

                Init_Sprite
                , Update_Sprite
```

1.65 games.library/Hide_Sprite

```
                games.library/Hide_Sprite

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

SYNOPSIS
  Hide_Sprite(GameScreen, Sprite)
                 a0         a1

  void Hide_Sprite(struct GameScreen *, struct Sprite *)

FUNCTION
  Hides a sprite from the screen display.

INPUTS  GameScreen - Pointer to an initialised GameScreen structure.
  Sprite - Pointer to an initialised Sprite structure.
```

```
SEE ALSO

                Hide_SpriteList
```

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

  SpriteList:
    dc.l   "LIST"
    dc.l   SpriteStruct1
    dc.l   SpriteStruct2
    dc.l   SpriteStruct3
    dc.l   SpriteStruct4
    dc.l   LISTEND

SEE ALSO
  Update_Sprite
```

1.67 games.library/Hide_SpriteList

```
games.library/Hide_SpriteList

NAME  Hide_SpriteList -- Hide sprites as specified by a SpriteList.

SYNOPSIS
  Hide_SpriteList(GameScreen, SpriteList)
                        a0         a1

  void Hide_SpriteList(struct GameScreen *, APTR SpriteList)

FUNCTION
  Hide a series of currently displayed sprites from the screen.  This
  function  is provided as an alternative to making constant calls to
```

```
  Hide_Sprite(), which can be quite time consuming.

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

  SpriteList:
    dc.l   "LIST"
    dc.l   SpriteStruct1
    dc.l   SpriteStruct2
    dc.l   SpriteStruct3
    dc.l   SpriteStruct4
    dc.l   LISTEND

SEE ALSO
  Hide_Sprite
```

1.68 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
  Removes  all  displayed  sprites from the screen simply by altering
  the  DMA Controller.  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 - Pointer to an initialised GameScreen structure.

SEE ALSO
  Return_AllSprites
```

1.69 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 - Pointer to an initialised GameScreen structure.

SEE ALSO
  Remove_AllSprites
```

1.70 games.library/

```
games.library/

NAME

SYNOPSIS

FUNCTION

INPUTS

RESULT

SEE ALSO
```

1.71 games.library/AllocAudio

```
                games.library/AllocAudio

NAME  AllocAudio -- Attempt to allocate the audio channels.

SYNOPSIS
  ErrorCode = AllocAudio()
     d0

  ULONG 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
```

1.72 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
```

1.73 games.library/InitSound

```
                games.library/InitSound

NAME  InitSound -- Initialise a sound structure for the play routines.

SYNOPSIS
  ErrorCode = InitSound(Sound)
     d0                 a0

  ULONG InitSound(struct Sound *)

FUNCTION
  This function will initialise a sound for use in the play routines.
  Its  main  job  is to load and assess the sound header, and fill in
  any missing fields.  It can also unpack sounds in some cases.

  If  the  following  fields  in  the Sound structure are detected as
  being NULL, InitSound() will fill them in for you:

    SAM_Data
    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.  This is imperative if the sound is in
  RAW format, for obvious reasons.
```

```
  Lists  are fully supported by this function, just pass a pointer to
  a standard "LIST" structure instead of a Sound.  (See Lists).

NOTE  If  the sound is in RAW format, then this function will have little
  effect, so you should set most of the fields yourself.

INPUTS  Sound - Pointer  to a single sound structure, or for multiple
    initialisations, a list of Sound's.

    STRUCTURE Sound,0
  ULONG SAM_VERSION          ;"SMV1"
  APTR  SAM_Stats            ;Reserved.
  UWORD SAM_Channel          ;Channel
  WORD  SAM_Priority         ;Priority
  APTR  SAM_Header           ;Sample info header, if any.
  APTR  SAM_Data             ;Address of sample data.
  ULONG SAM_Length           ;Length of sample data in WORDS.
  UWORD SAM_Octave           ;Octave/Note setting.
  UWORD SAM_Volume           ;Volume of sample (1 - 100).
  ULONG SAM_Attrib           ;Sound attributes.
  APTR  SAM_File             ;The file for the sound.
  LABEL SAM_SIZEOF

  SAM_VERSION
  The version of the structure, currently "SMV1".

  SAM_Channel
  The  channel  that  you  want  to play through.  Acceptable channel
  numbers are 0, 1, 2 and 3 (a total of 4 available channels).

  SAM_Priority
  The  priority  of  your sound goes here.  This field is used by the
  PlaySoundPri() function to determine if your sound should be played
  when the channel is busy.  Naturally, higher values are played over
  samples with lower values.

  SAM_Header
  Points to the very start of the sample, which in most cases will be
  the  something  like an IFF 8SVX header.  If the sample data is RAW
  then simply point to the start of the data here.

  SAM_Data
  This  field  points  to the actual data that is going to be played.
  Init_Sound()  will  fill this field in for you if you initialise it
  to 0.

  SAM_Length
  The  length of the sample data in words.  This field will be filled
  in for you if the sound has a recognised header (eg IFF).

  SAM_Octave
  The octave at which to play this sample.  The highest pitched value
  is  OCT_G0S,  the  lowest is OCT_A7S.  There are about 94 available
  settings, see games/sound.i to look at the complete list.

  SAM_Volume
```

```
  The volume of the sound, which lies in the range 0 - 100.  A volume
  of zero will not be heard, a volume of 100 is the loudest.

  SAM_Attrib
  Specifies the attributes for the sound.

    SBIT8   - Sound data is 8 bit (only set this if raw).

    SBIT16  - Sound data is 16 bit (only set this if raw).

    SMODVOL - Modulates the volume with the next channel.

    SMODPER - Modulate the sound's period with the next channel.

    SREPEAT - Repeats the sample forever.

  SAM_File
  If your sound is contained on disk, place a pointer to the filename
  here.   This  will  cause InitSound() to load the sound data in for
  you  (via  a  call  to SmartLoad()) and fill in the Header and Data
  fields.   The  rest  of  the  initialisation procedure will then be
  carried out.

SEE ALSO

                FreeSound
```

1.74 games.library/FreeSound

```
                games.library/FreeSound

NAME  FreeSound -- Free any allocations made in an initialised sound.

SYNOPSIS
  FreeSound(Sound)
            a0

  void FreeSound(struct Sound *)

FUNCTION
  Frees  any  allocations  made  in  the  initialisation  of  a Sound
  structure.   You have to call this function at some point for every
  initialised  Sound, otherwise resources may be withheld on the exit
  of your program.

  This function is fully supportive of LIST's.

INPUTS  Sound - Pointer to an intialised sound structure.

SEE ALSO

                InitSound
```

1.75 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.

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.  A result of 1 indicates that
    the channel is busy.
```

1.76 games.library/PlaySound

```
games.library/PlaySound

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

SYNOPSIS
  PlaySound(Sound)
            a0

  void PlaySound(struct Sound *)

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

  You  must  have initialised the sound structure before calling this
  function.

INPUTS  Sound - Pointer to a valid sound structure.

SEE ALSO
  PlaySoundDACx, PlaySoundPri, PlaySoundPriDACx
```

1.77 games.library/PlaySoundDACx

```
games.library/PlaySoundDACx

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

```
SYNOPSIS
  PlaySoundDACx(Sound)
                a0

  void PlaySoundDACx(struct Sound *)

  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 Sound - Pointer to a valid sound structure.

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

1.78 games.library/PlaySoundPriDACx

```
games.library/PlaySoundPriDACx

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

SYNOPSIS
  PlaySoundPriDACx(Sound)
                   a0

  void PlaySoundPriDACx(struct Sound *)

  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.

  This function supports prioritisation of sound effects.

INPUTS  Sound - Pointer to a valid sound structure.

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

1.79 games.library/PlaySoundPri

```
games.library/PlaySoundPri

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

SYNOPSIS
  PlaySoundPri(Sound)
               a0

  void PlaySoundPri(struct Sound *)

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!

  It is recommended that you use CHANNEL_ALL in the SAM_Channel field
  so  that  your  game  makes  maximum use of all the available sound
  channels.

INPUTS  Sound - Pointer to a valid sound structure.

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

1.80 games.library/

```
games.library/

NAME

SYNOPSIS

FUNCTION

INPUTS
```

RESULT

SEE ALSO