Screens

Paul Manias

COLLABORATORS			
	TITLE : Screens		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY	Paul Manias	July 26, 2024	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

1 Screens **1**

1.1 Screens Documentation . . . 1
1.2 Module: Screens . . . 1
1.3 Screens Module Overview . . . 2
1.4 Screens: Display Buffering . . . 3
1.5 Screens: Video Locking . . . 4
1.6 Screens: AllocVideoMem() . . . 4
1.7 Screens: BlankColours() . . . 5
1.8 Screens: BlankOn() . . . 5
1.9 Screens: BlankOff() . . . 5
1.10 Screens: ColourMorph() . . . 6
1.11 Screens: ColourToPalette() . . . 7
1.12 Screens/ChangeColours . . . 7
1.13 Screens: FreeVideoMem() . . . 8
1.14 Screens: GetScrType() . . . 8
1.15 Screens: HideDisplay() . . . 8
1.16 Screens: LockVideo() . . . 9
1.17 Screens: MoveBitmap() . . . 10
1.18 Screens: PaletteMorph() . . . 10
1.19 Screens: PaletteToColour() . . . 11
1.20 Screens: RefreshScreen() . . . 12
1.21 Screens: RemakeScreen() . . . 12
1.22 Screens: ResetBitmap() . . . 13
1.23 Screens: ReturnDisplay() . . . 13
1.24 Screens: SwapBuffers() . . . 13
1.25 Screens: TakeDisplay() . . . 14
1.26 Screens: UnlockVideo() . . . 14
1.27 Screens: UpdateColour() . . . 15
1.28 Screens: UpdatePalette() . . . 15
1.29 Screens: WaitRastLine() . . . 15

1.30 Screens: WaitSwitch() 16
1.31 Screens: WaitAVBL() 16
1.32 Screens: WaitVBL() 17
1.33 Screens: 17

Chapter 1

Screens

1.1 Screens Documentation

```
MODULE DOCUMENTATION
Name:      SCREENS
Version:   0.9 Beta
Date:      December 1997
Author:    Paul Manias
Copyright: DreamWorld Productions, 1996-1997.  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.

CHANGES VERSION 0.9B
Added:   WaitSwitch()
         LockVideo()
         UnlockVideo()
         Display Buffering
         Video Locking

Renamed: MovePicture() to MoveBitmap()
         ResetPicture() to ResetBitmap()

Edited:  MoveBitmap()

Deleted: Removed all Sprite references and functions.  Sprites will not
         appear in V1.0.

         All of the raster functions have been removed, as there is now
         an OO Raster object.
```

1.2 Module: Screens

```
 INTRODUCTION

 OBJECTS
 Raster
 Screen
```

THEORY
Display Buffering
Video Locking

FUNCTIONS
AllocVideoMem()
BlankOn()
BlankOff()
FreeVideoMem()
GetScrType()
HideDisplay()
LockVideo()
MoveBitmap()
ResetBitmap()
ReturnDisplay()
SwapBuffers()
TakeDisplay()
UnlockVideo()
WaitSwitch()
WaitAVBL()
WaitVBL()
WaitRastLine()

Colour Functions
BlankColours()
ChangeColours()
ColourMorph()
ColourToPalette()
PaletteToColour()
PaletteMorph()
UpdatePalette()
UpdateColour()

1.3 Screens Module Overview

SCREEN SUPPORT OVERVIEW

The Screens module was the first module to be designed, and speed was a major factor while it was being programmed. To keep it fast, the Screen object has been kept highly simplified while not giving away any powerful features. One example of this power is that you can move the screen around by changing its coordinates, and even dynamically alter its width and height without any adverse affect on the picture display (see the Redimension demo).

There exists a wide range of functions, including special effects such as proportional fading, which allows you to add some very smooth and impressive touches to your programs. The Raster support provides an easy to use gateway to the copper chip. Using the available commands you can acheive affects like mirrors and smooth gradients of colour.

The Screens module is further supported in GMSPrefs, allowing the user to select his preferred screen modes. A powerful feature is being able to

select the screen type, so you can change the display type from ILBM to Chunky for example. This can give you a great speed up if your hardware allows you to use such modes and if the game would benefit from such a change (eg 3D vectors). It is even possible to do things such as upgrading a game to hi-res interlaced, or running in different video modes such as DBLPAL.

1.4 Screens: Display Buffering

DISPLAY BUFFERING

Display buffering is an important issue when you want to double buffer or triple buffer a screen. The technique prevents flickering when drawing to the screen, and in the case of triple buffering it will speed up the program loop.

There are different ways to achieve display buffering, I will explain two of them:

Waiting Method
This method is used for double buffering and some cases of triple buffering. It involves waiting for the vertical blank after you have drawn your data, then calling SwapBuffers(). This will usually look something like this:

```
  while (loop) {
     Move(Bob);              /* Move objects */
     Draw(Bob);              /* Draw objects to bitmap */
     WaitVBL();              /* Wait for a vertical blank */
     SwapBuffers(Screen);    /* Swap the display buffers */
  }
```

You can see some real examples of this in the various demo source code.

Triple Buffering Method
This method can also be used for double buffering, but is provided specifically for triple buffering. When the screen module is loaded, it installs a special vertical blank interrupt that lies in the background. This interrupt looks at the current screen and checks the Switch field for TRUE. If set, the interrupt will switch the screen buffers while the program is running.

At this stage it is important to know what the MemPtrX fields mean when triple buffering:

```
  MemPtr1 - On display.
  MemPtr2 - Waiting for display/VBL ready.
  MemPtr3 - Being drawn to.
```

What this allows us to do is draw to the MemPtr3 buffer while the interrupt takes care of switching the MemPtr2 buffer to the display. Here is an example:

```
  Display(Screen);                      /* Display screen */

  Screen->Bitmap->Data = Screen->MemPtr2;

  while (loop) {

     /* This section will draw to the buffer in Bitmap->Data.  Because a
     ** switch will probably happen in this area you can understand why
     ** you are not allowed to read from the MemPtrX fields in such an
     ** "unsafe" area.
     */

     Move(Bob);
     Draw(Bob);

     /* At this point we have to make sure that the display has been
     ** switched.  If this is the case then we are running too fast and
     ** will have to wait until a buffer becomes ready.
     */

     WaitSwitch(Screen);                    /* Wait for switch to turn FALSE */
     Screen->Bitmap->Data = Screen->MemPtr3; /* Update for drawing */
     Screen->Switch = TRUE;                 /* We are ready to switch again */
  }
```

1.5 Screens: Video Locking

VIDEO LOCKING

1.6 Screens: AllocVideoMem()

```
FUNCTION
Name:     AllocVideoMem()
Short:    Allocate blitter memory.
Synopsis: APTR AllocVideoMem(LONG Size [d0], LONG Flags [d1])

DESCRIPTION
Allocates  a  block of memory suitable for the video display.  This type of
memory  is  also compatible with the blitter module, and should continue to
do so for all hardware configurations.

The  memory  will  be  tracked  as  outlined in AllocMemBlock() if resource
tracking is turned on.

INPUTS
Size  - The Size of the memory to allocate.
Flags - Memory flags as outlined in FreeMemBlock().

RESULT
Pointer  to  the  allocated memory.  All video memory is formatted with 0's
when allocated.  Returns NULL if error.
```

```
SEE ALSO
Kernel:  FreeMemBlock()
Screens: FreeVideoMem()
```

1.7 Screens: BlankColours()

```
FUNCTION
Name:     BlankColours()
Short:    Drives all screen colours to zero (black).
Synopsis: void BlankColours(*Screen [a0])

DESCRIPTION
Drives all the colours in a screen palette to zero, which will give a black
screen  (but  only  if  the ScrType makes use of the palette register).  If
successul,  you  won't be able to see any picture detail after calling this
routine.

INPUTS
Screen - Pointer to an initialised Screen structure.

SEE ALSO
Screens: BlankOn()
```

1.8 Screens: BlankOn()

```
FUNCTION
Name:     BlankOn()
Short:    Blanks out the entire display until BlankOff() is called.
Synopsis: void BlankOn(void)

DESCRIPTION
After  calling  this function the screen display will be completely blanked
out  until  BlankOff()  is  called.  For  the duration that the display is
blanked out, there will be no visible screen effects whatsoever.  Note that
Display(Screen) is completely incapable of ending a screen blanking period,
but  once  the  screen  display  is returned any screen alterations will be
visible.

This  function  is  intended  for  making  a  clean  transition between two
screens, ie closing one screen then opening another.

SEE ALSO
Screens: BlankOff()
```

1.9 Screens: BlankOff()

```
FUNCTION
Name:     BlankOff()
Short:    Gives back the display after BlankOn() was called.
```

```
Synopsis: void BlankOff(void)

DESCRIPTION
This  function  returns  the  screen  display after calling BlankOn().  Any
hidden  visual  changes  that occurred after the BlankOn() call will become
immediately visible after calling this function.

SEE ALSO
Screens: BlankOn()
```

1.10 Screens: ColourMorph()

```
FUNCTION
Name:     ColourMorph()
Short:    Fades a of set of colours into one colour value.
Synopsis: WORD ColourMorph(*Screen [a0], WORD FadeState [d0],
            WORD Speed [d1], LONG StartColour [d3], LONG AmtColours [d4],
            LONG SrcColour [d2], LONG DestColour [d5])

DESCRIPTION
Fades  the  screen from one colour into another colour.  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.

This function uses the proportional fading algorithm to acheive its effect.

NOTE
All  fading  functions  ignore  the colour values that are kept internally.
This  will  cause  problems  for  you  if you do not know what your current
palette looks like when using these functions.

EXAMPLE

    FadeState = NULL;
    do {
       WaitVBL();
       FadeState = ColourMorph(Screen,FadeState,1,0,32,0xFF00AA,0xA7BC30);
    }
    while (FadeState != NULL)

INPUTS
Screen      - Pointer to an initialised Screen structure.
FadeState   - Initialise  to  zero,  then keep sending the returned value
        back until you get a NULL in this field.
Speed     - The required speed for the fade.
SrcColour   - The colour that you are fading from, 0xRRGGBB format.
DestColour  - The colour that you are fading to, 0xRRGGBB format.
StartColour - The colour to start fading from (0 ... AmtColours-1).
AmtColours  - The amount of colours to fade (1 ... MaximumColours).  You
        must never use a value of 0 here.

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

```
SEE ALSO
Screens: ColourToPalette()
         PaletteMorph()
         PaletteToColour()
```

1.11 Screens: ColourToPalette()

```
FUNCTION
Name:      ColourToPalette()
Short:     Fades a set of colours into a range of values.
Synopsis:  WORD ColourToPalette(*Screen [a0], WORD FadeState [d0],
      WORD Speed [d1], WORD StartColour [d3], WORD AmtColours [d4],
            APTR Palette [a1], LONG Colour [d2])

DESCRIPTION
Fades a set of colours of the same value, into a range of colours specified
in  Palette.   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.

This function uses the proportional fading algorithm to acheive its effect.

NOTE
All  fading  functions  ignore  the colour values that are kept internally.
This  will  cause  problems  for  you  if you do not know what your current
palette  looks like when using these functions.  Keep track of your current
palette values to help you with functions like PaletteMorph().

INPUTS
Screen      - Pointer to an initialised Screen structure.
FadeState   - Initialise  to  zero,  then keep sending the returned value
         back until you get a NULL in this field.
Speed     - The required speed for the fade.
Palette     - Pointer to the palette used as the source.
Colour      - The colour that you are fading from, 0xRRGGBB format.
StartColour - The colour to start fading from (0 ... AmtColours-1).
AmtColours  - The amount of colours to fade (1 ... MaximumColours).  You
               must never use a value of 0 here.

RESULT
Returns NULL if the fade has finished.

SEE ALSO
Screens: ColourMorph()
         ColourToPalette()
         PaletteMorph()
```

1.12 Screens/ChangeColours

```
FUNCTION
Name:      ChangeColours()
Short:     Change a set of colours in a Screen's internal palette.
```

```
Synopsis:  void ChangeColours(*Screen [a0], APTR Colours [a1],
            LONG StartColour [d0], LONG AmtColours [d1])

DESCRIPTION
Changes all colours within the set range.  Alterations will only be made to
the screen's internal palette.

INPUTS
Screen      - Pointer to an initialised Screen structure.
Colours     - Pointer to a list of 24 bit colours.
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.
```

1.13 Screens: FreeVideoMem()

```
FUNCTION
Name:     FreeVideoMem()
Short:    Frees a memory block allocated from FreeVideoMem().
Synopsis: void FreeVideoMem(APTR MemBlock [d0])

DESCRIPTION
Frees a memory block allocated from AllocVideoMem().

INPUT
MemBlock - The memory block to be freed.

SEE ALSO
Kernel:  AllocMemBlock()
Screens: AllocVideoMem()
```

1.14 Screens: GetScrType()

```
FUNCTION
Name:     GetScrType()
Short:    Gets the default/user screen type.
Synopsis: LONG GetScrType(void)

DESCRIPTION
Returns  the  screen  type that is being used as the default in the screens
module.  This function is often used by other modules, since the ScrType is
a common field in structures not initialised by the Screens module.

RESULT
The default screen type (eg PLANAR).
```

1.15 Screens: HideDisplay()

```
FUNCTION
Name:     HideDisplay()
Short:    Hides the entire display from view.
Synopsis: *Screen = HideDisplay(void)

DESCRIPTION
This function is private and is for internal use only.

This  function will hide the GMS display from view.  This will cause the OS
viewport to be returned, but GMS will still be running "in the background".
If GMS is not running on top of another OS then the GMS DeskTop screen will
be displayed and the calling task will be put to the back.

If  no  GMS  screens  are  on  display then this function does nothing, and
returns a NULL value.

RESULT
Pointer  to  the  structure  of  the  Screen  that  has been hidden by this
function.  Otherwise NULL if no Screen was active.

SEE ALSO
Kernel: Display()
```

1.16 Screens: LockVideo()

```
FUNCTION
Name:     LockVideo()
Short:    Locks a screen at the front for video operations.
Synopsis: void LockVideo(*Screen [a0])

DESCRIPTION
This  function  is  required  whenever  you  want to draw something to your
screen,  or if you want to use the MemPtrX fields.  Locking a screen is the
only way to guarantee that the MemPtrX fields are pointing to video memory.

Attempting  to  draw  to  an  unlocked  screen with CPU or blitter can have
disasterous results, so use this function as often as necessary.

Some  systems  will  grant  locks  immediately, which is valid for most UMA
computers  and  standard  Amigas.  Other systems using graphics cards will
usually not do this.  Your task will have to wait until enough video memory
is ready, which may be until another task drops a screen lock or until the
user moves you to the front of the display.

NOTE
To keep the system stable, screen locks will nest.

INPUT
Screen - Pointer to an initialised screen structure.

SEE ALSO
Screens: UnlockVideo()
Theory:  Video Locking
```

1.17 Screens: MoveBitmap()

```
FUNCTION
Name:     MoveBitmap()
Short:    Moves the screen bitmap to specified X/Y values.
Synopsis: void MoveBitmap(*Screen [a0])

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

It will take the values from BmpXOffset and BmpYOffset in the Screen
structure and use them to set the new picture position. This function will
execute at the same speed for all offset 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 as you will damage the system.

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 normal execution time for this function on ECS/AGA is 2/3rds of a
single raster line on an A1200+Fast.

INPUT
Screen - Pointer to an initialised Screen structure. The BmpXOffset and
         BmpYOffset values will be used to set the picture's new on-screen
         position.

SEE ALSO
Screens: ResetBitmap()
```

1.18 Screens: PaletteMorph()

```
FUNCTION
Name:     PaletteMorph()
Short:    Fades a set of colours into a new set of values.
Synopsis: WORD PaletteMorph(*Screen [a0], WORD FadeState [d0],
           WORD Speed [d1], WORD StartColour [d3], WORD AmtColours [d4],
           LONG *SrcPalette [a1], APTR DestPalette [a2])

DESCRIPTION
This function will take the palette in SrcPalette, and use it to fade a
colour set into the palette given in DestPalette. 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.

This function uses the proportional fading algorithm to acheive its effect.
```

```
NOTE
All  fading  functions  ignore  the colour values that are kept internally.
This  will  cause  problems  for  you  if you do not know what your current
palette  looks  like  when  using  these  functions.  Keep  track  of your
palette's  values  and  point  to  them in SrcPalette if you find that this
problem is occurring for you.

INPUTS
Screen      - Pointer to an initialised Screen structure.
FadeState   - Initialise  to  zero,  then keep sending the returned value
         back until you get a NULL in this field.
Speed     - The required speed for the fade.
SrcPalette  - Pointer to the palette used as the source.
Destpalette - Pointer to the palette that you want to fade to.
StartColour - The colour to start fading from (0 ... AmtColours-1).
AmtColours  - The amount of colours to fade (1 ... MaximumColours).  You
         must never use a value of 0 here.

RESULT
Returns NULL if the fade has finished.

SEE ALSO
Screens: ColourMorph()
         ColourToPalette()
         PaletteToColour()
```

1.19 Screens: PaletteToColour()

```
FUNCTION
Name:   PaletteToColour()
Short:    Fades a set of colours into a specific colour value.
Synopsis: WORD PaletteToColour(*Screen [a0], WORD FadeState [d0],
     WORD Speed [d1], LONG StartColour [d3], LONG AmtColours [d4],
           APTR Palette [a1], LONG Colour [d2])

DESCRIPTION
This function will fade a set of various colour values into a single colour
value.   This  is  useful  for fading the screen to black for example.  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.

This function uses the proportional fading algorithm to acheive its effect.

NOTE
All  fading  functions  ignore  the colour values that are kept internally.
This  will  cause  problems  for  you  if you do not know what your current
palette looks like when using these functions.

INPUTS
Screen      - Pointer to an initialised Screen structure.
FadeState   - Initialise  to  zero,  then keep sending the returned value
         back until you get a NULL in this field.
Speed     - The required speed for the fade.
```

```
Palette     - Pointer to the palette used as the source.
Colour      - The colour you want to fade to, in 0xRRGGBB format.
StartColour - The colour to start fading from (0 ... AmtColours-1).
AmtColours  - The amount of colours to fade (1 ... MaximumColours).  You
         must never use a value of 0 here.

RESULT
Returns NULL if the fade has finished.

SEE ALSO
Screens: ColourMorph()
         PaletteMorph()
         PaletteToColour()
```

1.20 Screens: RefreshScreen()

```
FUNCTION
Name:     RefreshScreen()
Short:    Updates the screen display.
Synopsis: void RefreshScreen(*Screen [a0]);

DESCRIPTION



INPUT
Screen - Pointer to an initialised Screen structure.

SEE ALSO
Screens: WaitVBL()
```

1.21 Screens: RemakeScreen()

```
FUNCTION
Name:      RemakeScreen()
Short:     Remakes the screen display according to its size, width, and
            position on the monitor.
Synopsis:  void RemakeScreen(*Screen [a0])

DESCRIPTION
Remakes  the  Screen's  viewport  as quickly as possible.  If the Screen is
currently  hidden,  then  the  changes  will show up the next time you call
Display(Screen).

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

INPUT
Screen - Pointer to an initialised Screen structure.
```

1.22 Screens: ResetBitmap()

```
FUNCTION
Name:     ResetBitmap()
Short:    Resets the picture position to position 0X, 0Y.
Synopsis: void ResetBitmap(*Screen [a0])

DESCRIPTION
Resets the picture position to 0X, 0Y.  This method is faster than clearing
the PicXOffset and PicYOffset fields and then calling MoveBitmap().

INPUT
Screen - Pointer to an initialised Screen structure.

RESULT
PicXOffset and PicYOffset in the Screen will be cleared.

SEE ALSO
Screens: MoveBitmap
```

1.23 Screens: ReturnDisplay()

```
FUNCTION
Name:     ReturnDisplay()
Short:    Private function.
Synopsis: Screen = ReturnDisplay(void);

DESCRIPTION
This function is private and is for internal use only.

Returns  the  monitor  display to the OS that GMS is running on.  This is a
special  function  in  the  monitor drivers, and is reserved for use in the
screens module.

RESULT
Pointer to the Screen that was removed.

SEE ALSO
Screens: TakeDisplay()
```

1.24 Screens: SwapBuffers()

```
FUNCTION
Name:     SwapBuffers()
Short:    Switch the screen display buffers.
Synopsis: void SwapBuffers(*Screen [a0])

DESCRIPTION
If  the screen is double buffered, this function swaps Screen->MemPtr1 with
Screen->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

INPUT
Screen - Pointer to an initialised Screen structure.

SEE ALSO
Theory: Dislay Buffering
```

1.25 Screens: TakeDisplay()

```
FUNCTION
Name:     TakeDisplay()
Short:    Private function.
Synopsis: LONG TakeDisplay(*Screen [a0])

DESCRIPTION
Takes  the  display from the Operating System that GMS is running on.  This
is  a  special  function in the monitor drivers, and is reserved for use in
the screens module.

INPUTS
Screen - Pointer to an initialised Screen structure.

RESULT
Returns ERR_OK if successful.

SEE ALSO
Screens: ReturnDisplay()
```

1.26 Screens: UnlockVideo()

```
FUNCTION
Name:     UnlockVideo()
Short:
Synopsis: void UnlockVideo(*Screen [a0])

DESCRIPTION




INPUTS
Screen - Pointer to an initialised Screen object.

SEE ALSO
Screens: LockVideo()
Theory:  Video Locking
```

1.27 Screens: UpdateColour()

```
FUNCTION
Name:     UpdateColour()
Short:    Updates a 24 bit $RRGGBB value in a Screen palette.
Synopsis: void UpdateRGB(*Screen [a0], LONG Colour [d0], LONG RRGGBB [d1])

DESCRIPTION
Updates a single colour value in the screen's palette. The change is
immediately visible following the next vertical blank.

INPUTS
Screen - Pointer to an initialised Screen object.
Colour - The colour number to update, between 0 and Screen->AmtColours.
RRGGBB - Colour value in standard RRGGBB format.

SEE ALSO
Screens: UpdatePalette()
```

1.28 Screens: UpdatePalette()

```
FUNCTION
Name:     UpdatePalette()
Short:    Updates an entire Screen palette to new colour values.
Synopsis: void UpdatePalette(*Screen [a0])

DESCRIPTION
Updates an entire Screen palette to new colour values, as pointed to in
the GS_Palette field. If GS_Palette is NULL then all of the screens
colours will be turned black.

This function has no effect on true colour screens.

NOTE
Palette changes will appear when the next vertical blank occurs.

INPUTS
Screen - Pointer to an initialised Screen structure.

SEE ALSO
Screens: UpdateColour()
```

1.29 Screens: WaitRastLine()

```
FUNCTION
Name:     WaitRastLine()
Short:    Waits for the strobe to reach a specific line.
Synopsis: void WaitRastLine(WORD LineNumber [d0])

DESCRIPTION
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 0-255 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. Please keep your values restricted to the height of your screen.

SEE ALSO
Screens: WaitVBL()

1.30 Screens: WaitSwitch()

FUNCTION
Name: WaitSwitch()
Short: Wait for the buffers to be switched at the vertical blank.
Synopsis: void WaitSwitch(*Screen [a0])

DESCRIPTION

INPUT
Screen - Pointer to an initialised Screen object.

SEE ALSO
Theory: Display Buffering

1.31 Screens: WaitAVBL()

FUNCTION
Name: WaitAVBL()
Short: Waits for a vertical blank.
Synopsis: LONG WaitAVBL(void)

DESCRIPTION
Waits until the horizontal beam reaches the Vertical BLank area. This routine will try and give you as much VBL space as possible, usually by waiting for the exact point where the display stops. If this is not possible, then it will wait for the beam to reach the top of the monitor display.

This version of WaitVBL() will automatically pause your task if the user moves the focus to a different screen. This can be exceptionally useful, as it will prevent your program from stealing resources when the user wants

to do something else.

RESULT
Currently returns null.

SEE ALSO
Screens: WaitRastLine()

1.32 Screens: WaitVBL()

FUNCTION
Name: WaitVBL()
Short: Waits for a vertical blank.
Synopsis: LONG WaitVBL(void)

DESCRIPTION
Waits until the horizontal beam reaches the Vertical BLank area. This routine will try and give you as much VBL space as possible, usually by waiting for the exact point where the display stops. If this is not possible, then it will wait for the beam to reach the top of the monitor display.

RESULT
Currently returns null.

SEE ALSO
Screens: WaitRastLine()

1.33 Screens:

FUNCTION
Name:
Short:
Synopsis:

DESCRIPTION

INPUTS

RESULT

SEE ALSO