

Screens

Paul Manias

COLLABORATORS

	<i>TITLE :</i> Screens		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY	Paul Manias	June 4, 2025	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1	Screens	1
1.1	Screens.GPI	1
1.2	Screens.GPI Functions	1
1.3	games.library/AddScreen	2
1.4	games.library/DeleteScreen	6
1.5	games.library/ShowScreen	6
1.6	games.library/HideDisplay	7
1.7	games.library/Switch	7
1.8	games.library/AutoSwitch	8
1.9	games.library/SwapBuffers	9
1.10	games.library/RemakeScreen	9
1.11	games.library/MovePicture	10
1.12	games.library/ResetPicture	11
1.13	games.library/ColourMorph	11
1.14	games.library/ColourToPalette	12
1.15	games.library/PaletteMorph	13
1.16	games.library/PaletteToColour	14
1.17	games.library/ChangeColours	15
1.18	games.library/BlankColours	15
1.19	games.library/UpdatePalette	15
1.20	games.library/UpdateRGB12	16
1.21	games.library/UpdateRGB24	16
1.22	games.library/InitRasterList	17
1.23	games.library/UpdateRasterList	19
1.24	games.library/UpdateRasterLines	20
1.25	games.library/UpdateRasterCommand	20
1.26	games.library/UpdateRasterCommands	21
1.27	games.library/RemoveRasterList	21
1.28	games.library/HideRasterList	22
1.29	games.library/ShowRasterList	22

1.30	games.library/InitSprite	23
1.31	games.library/UpdateSprite	25
1.32	games.library/MoveSprite	25
1.33	games.library/HideSprite	26
1.34	games.library/UpdateSpriteList	26
1.35	games.library/HideSpriteList	27
1.36	games.library/RemoveAllSprites	27
1.37	games.library/ReturnAllSprites	28
1.38	games.library/AllocVideoMem	28
1.39	games.library/WaitVBL	29
1.40	games.library/WaitSVBL	29
1.41	games.library/WaitRastLine	30

Chapter 1

Screens

1.1 Screens.GPI

Name: SCREENS.GPI AUTODOC
Version: 0.5 Beta.
Date: 15 February 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.

1.2 Screens.GPI Functions

SCREENS.GPI
AddScreen ()
AllocVideoMem ()
AutoSwitch ()
DeleteScreen ()
HideDisplay ()
MovePicture ()
RemakeScreen ()
ResetPicture ()
Switch ()
ShowScreen ()
SwapBuffers ()
WaitVBL ()
WaitSVBL ()
WaitRastLine ()

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

```

UpdateRGB12 ()
UpdateRGB24 ()

RasterList Functions
InitRasterList ()
HideRasterList ()
RemoveRasterList ()
ShowRasterList ()
UpdateRasterList ()
UpdateRasterLines ()
UpdateRasterCommand ()
UpdateRasterCommands ()

Sprite Functions
InitSprite ()
FreeSprite ()
HideSprite ()
HideSpriteList ()
MoveSprite ()
RemoveAllSprites ()
ReturnAllSprites ()
UpdateSprite ()
UpdateSpriteList ()

```

1.3 games.library/AddScreen

games.library/AddScreen

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

SYNOPSIS

```

ErrorCode = AddScreen(GameScreen)
           d0                a0

```

```

ErrorCode = AddScreen(TagList)
           d0                a0

```

```

ULONG AddScreen(struct GameScreen *)

```

```

ULONG AddScreenTags(unsigned long ...)

```

FUNCTION

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

This function is fully supportive of GMS TagLists, see TAGS for more information.

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

INPUTS GameScreen - Pointer to a valid GameScreen structure or Taglist.

Here follows a description of each GameScreen field:

GS_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.

GS_MemPtr1, GS_MemPtr2, GS_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 `AddScreen()` will assume that the values are valid pointers to video memory and will use them as such.

GS_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.

GS_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 24 bit colours, unless you set the `COL12BIT` flag in `GS_ScrAttrib`.

GS_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 `InitRasterList()` for more information on rasterlists.

GS_AmtColours

The amount of colours in the screen palette, as pointed to by `GS_Palette`. If you set this value to NULL then `AddScreen()` will fill it in for you, via a check to `GS_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.

GS_ScrWidth, GS_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.

These fields will inherit the values specified by the user if they are set at zero.

GS_PicWidth, GS_PicByteWidth, GS_PicHeight

Defines the picture height and width. The picture is the display data that shows through onto screen. It can be larger than the screen area, but must never be smaller than the screen area. The pixel width must be divisible by 16. If you omit the `GS_ScrType` field (further down) then you are not expected to set the `PicByteWidth` value. In most circumstances setting `PicByteWidth` is unnecessary as it will be initialised from the `PicWidth` value.

These fields will inherit the values from `GS_ScrWidth` and `GS_ScrHeight` if they are set at zero.

GS_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.

GS_ScrXOffset, GS_ScrYOffset

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 permissible.

GS_PicXOffset, GS_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 MovePicture(). It is perfectly legal to preset these values before you call ShowScreen().

GS_ScrAttrib

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

DBLBUFFER - Allocates an extra screen buffer which is placed in GS_MemPtr2. See the SwapBuffers() function for more information on double buffering.

TPLBUFFER - Allocates two extra buffers which are placed in GS_MemPtr2 and GS_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.

SBUFFER - 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.

NOSCRBDR - Allows sprites and other displayable objects to appear outside of the viewport. Works on AGA only.

CENTRE - Centres the screen by calculating the correct offsets for GS_ScrXOffset and GS_ScrYOffset for any screen mode. The new settings will over-write any previous

values in these fields.

GETSCRMODE - Obtains the preferred user screen mode and writes it to GS_ScrMode.

GS_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, and 24Bit colours. NB: If you require compatibility for NTSC, ScrMode will not help you. Instead you must set ScrHeight to 200.

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 high resolution screen (1/2 lores).

SHIRES - Specifies a super-high resolution screen (1/4 lores).

LACED - Creates an interlaced display (1/2 pixel height).

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

COL12BIT - Inform GMS that we will be using 12 bit colours with this screen.

COL24BIT - Inform GMS that we will be using 24 bit colours with this screen (default).

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.

GS_ScrType

The display data type - either PLANAR, INTERLEAVED or CHUNKY8. Descriptions of these display types are out of the scope of this autodoc, so if you require further information perhaps you should try the RKM's. Note that for planar screens the bitplanes are stored sequentially, one after the other. There is no scattering of planar bitplane memory.

If you set this field to NULL then AddScreen() will initialise it to the preferred user screen type. This is exceptionally useful as some screen types are faster than others for certain effects. ScreenType independence is strongly encouraged because of this reason.

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

DeleteScreen, ShowScreen

1.4 games.library/DeleteScreen

games.library/DeleteScreen

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

SYNOPSIS

```
DeleteScreen(GameScreen)
           a0
```

```
void DeleteScreen(struct GameScreen *);
```

FUNCTION

This function will deallocate everything that was initialised when you called AddScreen().

If the screen you delete is currently active when you call this function, intuition 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 MemPtr1, MemPtr2 and MemPtr3 in the GameScreen structure, if those fields were allocated by AddScreen().

INPUTS GameScreen - Pointer to an initialised GameScreen structure.

SEE ALSO

AddScreen, HideDisplay, ShowScreen

1.5 games.library/ShowScreen

games.library/ShowScreen

NAME ShowScreen -- Displays an initialised game screen.

SYNOPSIS

```
ShowScreen(GameScreen)
           a0
```

```
void ShowScreen(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 AddInputHandler() 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

AddScreen, HideDisplay, DeleteScreen

1.6 games.library/HideDisplay

games.library/HideDisplay

NAME HideDisplay -- Hides the GMS display from view.

SYNOPSIS

```
GameScreen = HideDisplay()  
d0
```

```
struct GameScreen * HideDisplay(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 AutoSwitch() 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

ShowScreen, Switch, {"AutoSwitch" LINK "AutoSwitch()"}

1.7 games.library/Switch

games.library/Switch

NAME Switch -- Stops your task and resumes execution of the next primary task in the queue.

SYNOPSIS

```
Switch()
```

```
void Switch(void)
```

FUNCTION

Switches your task over to the next task in the queue. This

function will not return until the user reactivates your task, so your tasks execution is effectively stopped. Any secondary processes and interrupts that you have spawned will continue to execute, so multi-tasking can still be effective.

If the next task is screen-based, then your screen display will be removed and the new screen will be displayed. If you have any secondary tasks running, then take note: You must not allow them to use the drawing/blitter operations as your display memory may be temporarily moved to free up video memory. Blitting to an invisible display is also considered to be bad practice as most GMS tasks require all available blitter time. We also ask you to refrain from using the audio functions as the next task will probably be needing all available channels.

If there are no more GMS tasks in the queue, then the screen display will return to intuition. GMS supports two methods of screen switching to intuition, 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 wait until the close gadget is pressed, whereupon your game will continue where it left off.

Switch-To-Screen opens an intuition 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

AutoSwitch, HideDisplay, WaitSVBL

1.8 games.library/AutoSwitch

games.library/AutoSwitch

NAME

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

SYNOPSIS

AutoSwitch()

void AutoSwitch(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 wait until the close gadget is pressed, whereupon your game will continue where it left off.

Switch-To-Screen opens an intuition 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

Switch, HideDisplay, WaitSVBL

1.9 games.library/SwapBuffers

games.library/SwapBuffers

NAME SwapBuffers -- Switch the screen display buffers.

SYNOPSIS

```
SwapBuffers(GameScreen)
           a0
```

```
void SwapBuffers(struct GameScreen *)
```

FUNCTION

Swaps GS_MemPtr1 and GS_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.

1.10 games.library/RemakeScreen

games.library/RemakeScreen

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

SYNOPSIS

```
RemakeScreen(GameScreen)
           a0
```

```
void RemakeScreen(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 ShowScreen().

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

INPUTS GameScreen - Pointer to an initialised GameScreen structure.

1.11 games.library/MovePicture

games.library/MovePicture

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

SYNOPSIS

```
MovePicture(GameScreen)
           a0
```

```
void MovePicture(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. 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 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

ResetPicture

1.12 games.library/ResetPicture

games.library/ResetPicture

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

SYNOPSIS

```
ResetPicture(GameScreen)
           a0
```

```
void ResetPicture(struct GameScreen *)
```

FUNCTION

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

INPUTS GameScreen - Pointer to an initialised GameScreen structure.

RESULT PicXOffset and PicYOffset in the GameScreen will be cleared.

SEE ALSO

MovePicture

1.13 games.library/ColourMorph

games.library/ColourMorph

NAME ColourMorph -- Fades a of set of colours into one colour value.

SYNOPSIS

```
FadeState = ColourMorph(GameScreen, FadeState, Speed, StartColour,
           d0           a0           d0           d1           d3
           AmtColours, SrcColour, DestColour)
           d4           d2           d5
```

```
UWORD ColourMorph(struct GameScreen *, UWORD FadeState, UWORD Speed,
           ULONG StartColour, ULONG AmtColours,
           ULONG SrcColour, ULONG DestColour)
```

FUNCTION

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 proportional fading for 24 bit colours, and additive fading for 12 bit colours.

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 {
    WaitSVBL();
    FadeState = ColourMorph(GameScreen, FadeState, 1, 0, 32, 0xFF00AA, 0xA7BC30);
}
while (FadeState != NULL)
```

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.
 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 FadeState - Returns NULL if the fade has finished.

SEE ALSO

PaletteToColour, PaletteMorph, ColourToPalette

1.14 games.library/ColourToPalette

games.library/ColourToPalette

NAME ColourToPalette -- Fades a set of colours into a range of values.

SYNOPSIS

```
FadeState = ColourToPalette(GameScreen, FadeState, Speed,
    d0                a0                d0                d1
    StartColour, AmtColours, Palette,
    d3                d4                a1
    Colour)
    d2
```

```
UWORD ColourToPalette(struct GameScreen *, UWORD FadeState,
    UWORD Speed, UWORD StartColour,
    UWORD AmtColours, APTR Palette,
    ULONG Colour);
```

FUNCTION

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 proportional fading for 24 bit colours, and additive fading for 12 bit colours.

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 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.
 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 FadeState - Returns NULL if the fade has finished.

SEE ALSO

PaletteMorph, ColourToPalette, ColourMorph

1.15 games.library/PaletteMorph

games.library/PaletteMorph

NAME PaletteMorph -- Fades a set of colours into a new set of values.

SYNOPSIS

```
FadeState = PaletteMorph(GameScreen, FadeState, Speed, StartColour
    d0                a0                d0                d1                d3
    AmtColours, SrcPalette, DestPalette)
    d4                a1                a2
```

```
UWORD PaletteMorph(struct GameScreen *, UWORD FadeState,
    UWORD Speed, UWORD StartColour,
    UWORD AmtColours, APTR SrcPalette,
    APTR DestPalette)
```

FUNCTION

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 proportional fading for 24 bit colours, and additive fading for 12 bit colours.

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 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.
 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 FadeState - Returns NULL if the fade has finished.

SEE ALSO

ColourToPalette, PaletteToColour, ColourMorph

1.16 games.library/PaletteToColour

games.library/PaletteToColour

NAME PaletteToColour -- Fades a set of colours into a specific colour value.

SYNOPSIS

```
FadeState = PaletteToColour(GameScreen, FadeState, Speed,
    d0          a0          d0          d1
    StartColour, AmtColours, Palette,
    d3          d4          a1
    Colour)
    d2
```

```
UWORD PaletteToColour(struct GameScreen *, UWORD FadeState,
    UWORD Speed, ULONG StartColour,
    ULONG AmtColours, APTR Palette, ULONG Colour)
```

FUNCTION

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 proportional fading for 24 bit colours, and additive fading for 12 bit colours.

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 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.
 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 FadeState - Returns NULL if the fade has finished.

SEE ALSO

PaletteMorph, PaletteToColour, ColourMorph

1.17 games.library/ChangeColours

games.library/ChangeColours

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

SYNOPSIS

```
ChangeColours(GameScreen, Colours, StartColour, AmtColours)
                a0          a1          d0          d1
```

```
void ChangeColours(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 - Pointer to a list of colours, either 12 bit or 24 bit 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.

1.18 games.library/BlankColours

games.library/BlankColours

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

SYNOPSIS

```
BlankColours(GameScreen)
              a0
```

```
void BlankColours(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.

1.19 games.library/UpdatePalette

games.library/UpdatePalette

NAME UpdatePalette - Updates an entire GameScreen palette to new colour values.

SYNOPSIS

```
UpdatePalette(GameScreen)
    a0
```

```
void UpdatePalette(struct GameScreen *)
```

FUNCTION

Updates an entire GameScreen palette to new colour values as set in GS_Palette. This function works for both 12 bit and 24 bit GameScreen palettes.

Under current circumstances the changes will appear immediately after the next vertical blank.

INPUTS GameScreen - Pointer to an initialised GameScreen structure.

SEE ALSO

UpdateRGB12, UpdateRGB24

1.20 games.library/UpdateRGB12

games.library/UpdateRGB12

NAME UpdateRGB12 -- Updates a 12 bit RGB colour value.

SYNOPSIS

```
UpdateRGB12(GameScreen, Colour, RGB)
    a0    d0    d1
```

```
void UpdateRGB12(struct GameScreen *, UWORD Colour, UWORD RGB)
```

FUNCTION

INPUTS GameScreen - Pointer to an initialised GameScreen structure.

Colour - The colour number to update, range of 0-31.

RGB - Colour value in standard RGB format (eg \$fff for white)

SEE ALSO

UpdateRGB24

1.21 games.library/UpdateRGB24

games.library/UpdateRGB24

NAME UpdateRGB24 -- Updates a 24 bit RRGGBB colour value.

SYNOPSIS

```
UpdateRGB24(GameScreen, Colour, RRGGBB)
    a0    d0    d1
```

```
void UpdateRGB24(struct GameScreen *, ULONG Colour, UWORD RGB)
```

FUNCTION

INPUTS GameScreen - Pointer to an initialised GameScreen structure.
 Colour - The colour number to update, range of 0-31.
 RGB - Colour value in standard RRGGBB format.

SEE ALSO

UpdateRGB12

1.22 games.library/InitRasterList

games.library/InitRasterList

NAME InitRasterList -- Initialise a new rasterlist.

SYNOPSIS

```
ErrorCode = InitRasterList(GameScreen)
    d0                                a0
```

```
UWORD InitRasterList(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. Do not use this command unless you have set the COL12BIT flag in GS_ScrAttrib.

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 GS_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. Do not use this command unless you have set the COL12BIT flag in GS_ScrAttrib.

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 GS_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.

GS_RasterList in this structure must contain a pointer 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 GS_RasterList.

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

SEE ALSO

UpdateRasterList, ShowRasterList, HideRasterList,
RemoveRasterList, games/games.i

1.23 games.library/UpdateRasterList

games.library/UpdateRasterList

NAME UpdateRasterList -- Update an existing rasterlist.

SYNOPSIS

```
UpdateRasterList (GameScreen)
                  a0
```

```
void UpdateRasterList(struct GameScreen *)
```

FUNCTION

Completely updates a rasterlist's commands and waitline's to whatever values GS_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 any changes are 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 UpdateRastCommands() or UpdateRastLines(), which can be a bit faster in certain situations.

INPUTS GameScreen - Pointer to an initialised GameScreen structure.

SEE ALSO

InitRasterList, ShowRasterList, HideRasterList,
RemoveRasterList, UpdateRastCommands, UpdateRastLines,
games/games.i

1.24 games.library/UpdateRasterLines

games.library/UpdateRasterLines

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

SYNOPSIS

```
void UpdateRasterLines(GameScreen)
                        a0
```

```
void UpdateRasterLines(struct GameScreen *)
```

FUNCTION

Updates every occurrence 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 UpdateRasterList() routine.

INPUTS GameScreen - Pointer to an initialised GameScreen structure.

SEE ALSO

UpdateRasterCommand, UpdateRasterCommands, UpdateRasterList

1.25 games.library/UpdateRasterCommand

games.library/UpdateRasterCommand

NAME UpdateRasterCommand -- Update a single rasterlist command.

SYNOPSIS

```
UpdateRasterCommand(GameScreen, Command)
                    a0          a2
```

```
void UpdateRasterCommand(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 UpdateRasterList() or UpdateRasterCommands().

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

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

SEE ALSO

UpdateRasterCommands, UpdateRasterLines, UpdateRasterList

1.26 games.library/UpdateRasterCommands

games.library/UpdateRasterCommands

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

SYNOPSIS

```
UpdateRasterCommands(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 UpdateRasterCommand(). For all the commands, use UpdateRasterList().

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

UpdateRasterCommand, UpdateRasterLines, UpdateRasterList

1.27 games.library/RemoveRasterList

games.library/RemoveRasterList

NAME RemoveRasterList -- Hide and delete RasterList from memory.

SYNOPSIS

```
RemoveRasterList(GameScreen)
                  a0
```

```
void RemoveRasterList(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 InitRasterList().

INPUTS GameScreen - Pointer to an initialised GameScreen structure.

SEE ALSO

InitRasterList, ShowRasterList, HideRasterList, RemoveRasterList,
games/games.i

1.28 games.library/HideRasterList

games.library/HideRasterList

NAME HideRasterList -- Hide a rasterlist from the display.

SYNOPSIS

```
HideRasterList(GameScreen)
                a0
```

```
void HideRasterList(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 ShowRasterList().

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

InitRasterList, RemoveRasterList, ShowRasterList, HideRasterList,
UpdateRasterList

1.29 games.library/ShowRasterList

games.library/ShowRasterList

NAME ShowRasterList -- Display a rasterlist on screen.

SYNOPSIS

```
ShowRasterList(GameScreen)
                a0
```

```
void ShowRasterList(struct GameScreen *)
```

FUNCTION

Display a rasterlist on the screen. The pointer to the rasterlist must lie in GS_RasterList, and must have been initialised by a call to InitRasterList().

INPUTS GameScreen - Pointer to an initialised GameScreen structure.

SEE ALSO

InitRasterList, HideRasterList, ShowRasterList, RemoveRasterList,
UpdateRasterList

1.30 games.library/InitSprite

games.library/InitSprite

NAME InitSprite -- Initialise a sprite structure.

SYNOPSIS

```
ErrorCode = InitSprite(GameScreen,Sprite)
           d0             a0             a1
```

```
ULONG InitSprite(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 UpdateSprite(), MoveSprite() 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.
Sprite - Looks like this:

```
STRUCTURE SPR,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_XCoord       ;X position (screen relative).
WORD SPR_YCoord       ;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_XCoord

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

SPR_YCoord

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

`MoveSprite`, `UpdateSprite`, `UpdateSpriteList`, `HideSpriteList`, `games/games.i`

1.31 `games.library/UpdateSprite`

`games.library/UpdateSprite`

NAME `UpdateSprite` -- Place a sprite on the screen.

SYNOPSIS

```
UpdateSprite(GameScreen, Sprite)
             a0      a1
```

```
void UpdateSprite(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

`InitSprite`, `MoveSprite`

1.32 `games.library/MoveSprite`

`games.library/MoveSprite`

NAME `MoveSprite` -- Move a sprite to a new screen location.

SYNOPSIS

```
MoveSprite(GameScreen, Sprite)
             a0      a1
```

```
void MoveSprite(struct GameScreen *, struct Sprite *)
```

FUNCTION

Moves a sprite to a new screen location according to the X and Y co-ordinates found SPR_XCoord and SPR_YCoord in the Sprite structure. This function does not act on any other Sprite fields and is intended for use with 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

InitSprite, UpdateSprite

1.33 games.library/HideSprite

games.library/HideSprite

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

SYNOPSIS

```
HideSprite(GameScreen, Sprite)
           a0          a1
```

```
void HideSprite(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

HideSpriteList

1.34 games.library/UpdateSpriteList

games.library/UpdateSpriteList

NAME UpdateSpriteList -- Update a list of initialised sprites.

SYNOPSIS

```
UpdateSpriteList(GameScreen, SpriteList)
                a0          a1
```

```
void UpdateSpriteList(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 `UpdateSprite()`, which can be quite time consuming.

INPUTS `GameScreen` - Pointer to an initialised `GameScreen` structure.
`SpriteList` - 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    Sprite1
    dc.l    Sprite2
    dc.l    Sprite3
    dc.l    Sprite4
    dc.l    LISTEND
```

SEE ALSO

`UpdateSprite`

1.35 `games.library/HideSpriteList`

`games.library/HideSpriteList`

NAME `HideSpriteList` -- Hide sprites as specified by a `SpriteList`.

SYNOPSIS

```
HideSpriteList(GameScreen, SpriteList)
                a0          a1
```

```
void HideSpriteList(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 `HideSprite()`, which can be quite time consuming.

INPUTS `GameScreen` - Pointer to an initialised `GameScreen` structure.
`SpriteList` - 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    Sprite1
    dc.l    Sprite2
    dc.l    Sprite3
    dc.l    Sprite4
    dc.l    LISTEND
```

SEE ALSO

`HideSprite`

1.36 `games.library/RemoveAllSprites`

games.library/RemoveAllSprites

NAME RemoveAllSprites -- Remove all sprites from the display.

SYNOPSIS

```
RemoveAllSprites(GameScreen)
                a0
```

```
void RemoveAllSprites(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 ReturnAllSprites().

INPUTS GameScreen - Pointer to an initialised GameScreen structure.

SEE ALSO

ReturnAllSprites

1.37 games.library/ReturnAllSprites

games.library/ReturnAllSprites

NAME ReturnAllSprites -- Return all sprites to the display.

SYNOPSIS

```
ReturnAllSprites(GameScreen)
                a0
```

```
void ReturnAllSprites(struct GameScreen *)
```

FUNCTION

Returns all sprites that were previously removed by RemoveAllSprites().

INPUTS GameScreen - Pointer to an initialised GameScreen structure.

SEE ALSO

RemoveAllSprites

1.38 games.library/AllocVideoMem

games.library/AllocVideoMem

NAME AllocVideoMem -- Allocate blitter memory.

SYNOPSIS

```
Memory = AllocVideoMem(Size)
        d0          d0
```

```
APTR AllocVideoMem(ULONG Size)
```

FUNCTION

Allocates a block of memory suitable for the video display. This type of memory is also compatible with the Blitter.GPI, 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.

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

SEE ALSO

FreeMemBlock

1.39 games.library/WaitVBL

games.library/WaitVBL

NAME WaitVBL -- Waits for a vertical blank.

SYNOPSIS

```
WaitVBL()
```

```
void WaitVBL(void);
```

FUNCTION

Waits until the horizontal beam reaches the exact start of the VBL (just after the screen data stops on the display). This is a dynamic function, so even if you move your screen around using RemakeScreen(), the wait line will move along with it, giving you more (or less) VBL space.

NOTE Use WaitSVBL() if you want automatic screen switching checks.

SEE ALSO

WaitRastLine, WaitSVBL

1.40 games.library/WaitSVBL

games.library/WaitSVBL

NAME WaitSVBL -- Check for switch then wait for a vertical blank.

SYNOPSIS

```
WaitSVBL()
```

```
void WaitSVBL(void);
```

FUNCTION

Waits until the horizontal beam reaches the Vertical BBlank 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 has an implanted AutoSwitch() call to make screen switching very easy to implement.

SEE ALSO

WaitRastLine, WaitVBL

1.41 games.library/WaitRastLine

games.library/WaitRastLine

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

SYNOPSIS

```
WaitRastLine(LineNumber)
             d0
```

```
void WaitRastLine(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 0-255 for a standard 320x256 screen. It is permissible 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

WaitSVBL, WaitVBL
