

Palette

COLLABORATORS

	<i>TITLE :</i> Palette	
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>
WRITTEN BY		July 31, 2024
<i>SIGNATURE</i>		

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1	Palette	1
1.1	Palette	1
1.2	asyncfade	1
1.3	asyncfadestatus	2
1.4	fade	2
1.5	freepalette	2
1.6	initpalette	3
1.7	loadpalette	3
1.8	red	3
1.9	green	3
1.10	blue	4
1.11	createpalette	4
1.12	rgb	4
1.13	nbcolour	4
1.14	palrgb	4
1.15	getscreenpalette	5
1.16	getpicturepalette	5
1.17	displaypalette	5
1.18	usepalette	5
1.19	fadeout	5

Chapter 1

Palette

1.1 Palette

PureBasic 'Palette' library

Palette are very important for all displayed elements. There are very good supported and you can do almost everything you need. Special functions like very fast fading routines allow you to achieve very good effects.

Commands summary:

- Blue
- CreatePalette
- DisplayPalette
- FreePalette
- Fade
- FadeOut
- GetPicturePalette
- GetScreenPalette
- Green
- InitPalette
- LoadPalette
- NbColour
- PalRrgb
- Red
- Rgb
- UsePalette

Example:

Fading workbench

1.2 asyncfade

SYNTAX

ASyncFade(#Palettel, #Palette2, Step, NbLoop, ScreenID)

STATEMENT

Same as `NFade()` routine, but doesn't halt the program. The fade is executed in the background.

You can use `NAsyncStatus()` to find out if the background fade is finished or not.

1.3 `asyncfadestatus`

SYNTAX

```
Result.b = ASyncFadeStatus
```

STATEMENT

Return '-1' if the Fade is still running or '0' if the fade has finished.

Example:

```
Repeat                ; Typical loop to wait for the end
  NVWait              ; of the background fade.
Until NASyncStatus = 0 ;
```

1.4 `fade`

SYNTAX

```
Fade(#Palette1, #Palette2, ScreenID, Step, NbLoop)
```

STATEMENT

Do a nice fade between the two palettes. The palettes must have the same number of colours or it could crash. Step controls the speed of the Fade (1 is the fastest, >1 numbers will slow down the fade speed). NbLoop controls how many loops the Fade must do before exiting. By default the Fade ALWAYS executes 255 loops. So you can adjust it manually (ie: with a Step of 2, you should use an NbLoop of $255/2 \approx 127$)

This function is optimized for speed, and gives very good results on any Amigas (020 recommended though), with high-coloured screens (upto 256 colours).

1.5 `freepalette`

SYNTAX

```
FreePalette(#Palette)
```

STATEMENT

Free the memory allocated to the given #Palette.

1.6 initpalette

SYNTAX

```
result.l = InitPalette(#NumPaletteMax)
```

FUNCTION

Init all the Palette environments for later use. You must put this function at the top of your source code if you want to use the Palette commands.

#NumPaletteMax : Maximum number of Palettes to handle.

1.7 loadpalette

SYNTAX

```
Result.l = LoadPalette(#Palette, FileName$)
```

FUNCTION

Load an palette from a standard IFF file (picture, brush or single palette are all supported). The palette is initialized with the value found inside the file. If the palette has been correctly loaded, the 'Result' value is positive (ie: > 0), else an error has happened:

Possible 'Result' values:

- 1 : File not found
- 2 : Not an IFF File
- 3 : Palette information not found in this IFF file

1.8 red

SYNTAX

```
Red.w = Red(ColourIndex)
```

FUNCTION

Return the Red value of the colour found in the current palette. Returned value is always between 0 and 255.

1.9 green

SYNTAX

```
Green.w = Green(ColourIndex)
```

FUNCTION

Return the Green value of the colour found in the current palette. Returned value is always between 0 and 255.

1.10 blue

SYNTAX

```
Blue.w = Blue(ColourIndex)
```

FUNCTION

Return the Blue value of the colour found in the current palette.
Returned value is always between 0 and 255.

1.11 createpalette

SYNTAX

```
res.l = CreatePalette(#Palette, NbColour)
```

COMMAND

Tries to create a new palette with given argument. The size, in memory, taken by a palette object can be calculated like this:

```
Size (in bytes) = NbColours * 12 + 12
```

The created palette is ready to use and filled with colour 0.

1.12 rgb

SYNTAX

```
Rgb(ScreenID, ColourIndex, R, G, B)
```

STATEMENT

Change directly the RGB value of a colour in the given Screen.

1.13 nbcolour

SYNTAX

```
Result.l = NbColour
```

STATEMENT

Returns the number of colour of currently used palette.

1.14 palrgb

SYNTAX

```
PalRgb(ColourIndex, R, G, B)
```

STATEMENT

Change the RGB value of a colour in the current palette.

1.15 getscreenpalette

SYNTAX

```
res.l = GetScreenPalette(#Palette, ScreenID)
```

COMMAND

Tries to create a new palette and fill it with screen colour information.
If res = 0 the palette could not be created.

1.16 getpicturepalette

SYNTAX

```
res.l = GetPicturePalette(#Palette, PictureID)
```

COMMAND

Tries to create a new palette and fill it with picture colour information.
If res = 0 the palette could not be created.

PictureID is a pointer to an IFF/ILBM file in memory.

1.17 displaypalette

SYNTAX

```
DisplayPalette(#Palette, ScreenID)
```

STATEMENT

Display the given #Palette on the screen.

1.18 usepalette

SYNTAX

```
UsePalette(#Palette)
```

STATEMENT

Change the current Palette to the given #Palette.

1.19 fadeout

SYNTAX

```
FadeOut(#Palette, Step, NbLoop, ScreenID)
```

STATEMENT

It will display a very nice fade out from the given palette.
The palette WILL be modified (at the end of the fading, the palette will be completely black). The fadeout speed can be controlled with the 'Step' parameter.

If Step = 1 then the fading will be smooth and take 1 vwait
before fading the next frame
If Step = 2 fading will be 2 times faster than Step 1 ...

NbLoop is used to fade partially a screen:

If NbLoop = 255, the whole screen will be black at end, because
with 255 loops, the fadeout is complete

If NbLoop = 50, after 50 loop the FadeOut will stop. Test it
to understand better :)

This routine is optimized for speed and gives excellent results
even on small Amiga. And more, it's fully system-friendly (no
hardware bang...) so works on GFX card too ! It's better to use
this routine than the Fade() to do standard Fade Out..