

**render**

<b>COLLABORATORS</b>
----------------------

	<i>TITLE :</i> render		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY		May 24, 2025	

<b>REVISION HISTORY</b>
-------------------------

NUMBER	DATE	DESCRIPTION	NAME

# Contents

<b>1</b>	<b>render</b>	<b>1</b>
1.1	render.doc . . . . .	1
1.2	render.library/AddChunkyImageA . . . . .	2
1.3	render.library/AddHistogramA . . . . .	3
1.4	render.library/AddRGB . . . . .	4
1.5	render.library/AddRGBImageA . . . . .	4
1.6	render.library/AllocRenderMem . . . . .	5
1.7	render.library/AllocRenderVec . . . . .	6
1.8	render.library/AllocRenderVecClear . . . . .	6
1.9	render.library/ApplyAlphaChannelA . . . . .	7
1.10	render.library/BestPen . . . . .	8
1.11	render.library/Chunky2BitMapA . . . . .	9
1.12	render.library/Chunky2RGBA . . . . .	10
1.13	render.library/ChunkyArrayDiversityA . . . . .	11
1.14	render.library/ConvertChunkyA . . . . .	13
1.15	render.library/CountRGB . . . . .	15
1.16	render.library/CreateAlphaArrayA . . . . .	16
1.17	render.library/CreateHistogramA . . . . .	17
1.18	render.library/CreateMapEngineA . . . . .	18
1.19	render.library/CreatePaletteA . . . . .	19
1.20	render.library/CreatePenTableA . . . . .	20
1.21	render.library/CreateRMHandlerA . . . . .	21
1.22	render.library/CreateScaleEngineA . . . . .	22
1.23	render.library/DeleteHistogram . . . . .	23
1.24	render.library/DeleteMapEngine . . . . .	23
1.25	render.library/DeleteRMHandler . . . . .	24
1.26	render.library/DeletePalette . . . . .	24
1.27	render.library/DeleteScaleEngine . . . . .	25
1.28	render.library/ExportPaletteA . . . . .	25
1.29	render.library/ExtractAlphaChannelA . . . . .	26

---

1.30	render.library/ExtractPaletteA . . . . .	27
1.31	render.library/FlushPalette . . . . .	28
1.32	render.library/FreeRenderMem . . . . .	29
1.33	render.library/FreeRenderVec . . . . .	29
1.34	render.library/ImportPaletteA . . . . .	30
1.35	render.library/InsertAlphaChannelA . . . . .	31
1.36	render.library/MapChunkyArrayA . . . . .	32
1.37	render.library/MapRGBArrayA . . . . .	33
1.38	render.library/MixAlphaChannelA . . . . .	34
1.39	render.library/MixRGBArrayA . . . . .	35
1.40	render.library/Planar2ChunkyA . . . . .	36
1.41	render.library/QueryHistogram . . . . .	37
1.42	render.library/RenderA . . . . .	38
1.43	render.library/RGBArrayDiversityA . . . . .	40
1.44	render.library/ScaleA . . . . .	42
1.45	render.library/ScaleOrdinate . . . . .	43
1.46	render.library/SortPaletteA . . . . .	44
1.47	render.library/TintRGBArrayA . . . . .	45

# Chapter 1

## render

### 1.1 render.doc

```
AddChunkyImageA ()
AddHistogramA ()
AddRGB ()
AddRGBImageA ()
AllocRenderMem ()
AllocRenderVec ()
AllocRenderVecClear ()
ApplyAlphaChannelA ()
BestPen ()
Chunky2BitMapA ()
Chunky2RGBA ()
ChunkyArrayDiversityA ()
ConvertChunkyA ()
CountRGB ()
CreateAlphaArrayA ()
CreateHistogramA ()
CreateMapEngineA ()
CreatePaletteA ()
CreatePenTableA ()
CreateRMHandlerA ()
CreateScaleEngineA ()
DeleteHistogram ()
DeleteMapEngine ()
DeletePalette ()
DeleteRMHandler ()
DeleteScaleEngine ()
ExportPaletteA ()
ExtractAlphaChannelA ()
ExtractPaletteA ()
FlushPalette ()
FreeRenderMem ()
FreeRenderVec ()
ImportPaletteA ()
InsertAlphaChannelA ()
MapChunkyArrayA ()
MapRGBArrayA ()
MixAlphaChannelA ()
MixRGBArrayA ()
```

---

```

Planar2ChunkyA()
QueryHistogram()
RenderA()
RGBArrayDiversityA()
ScaleA()
ScaleOrdinate()
SortPaletteA()
TintRGBArrayA()

```

## 1.2 render.library/AddChunkyImageA

### NAME

AddChunkyImageA - add chunky bytes to a histogram.  
 AddChunkyImage - varargs stub for AddChunkyImageA.

### SYNOPSIS

```

success = AddChunkyImageA(histogram, chunky, width, height,
d0                a0          a1          d0          d1
                        palette, taglist)
                        a2          a3

```

```

ULONG AddChunkyImageA(APTR, UBYTE *, UWORD, UWORD,
                        APTR, struct TagItem *)

```

```

ULONG AddChunkyImage(APTR, UBYTE *, UWORD, UWORD,
                      APTR, tag, ..., TAG_DONE)

```

### FUNCTION

This function adds an array of chunky bytes to a histogram. The color information contained in the chunky array and its palette will be stored in the histogram.

### INPUTS

histogram	- pointer to a histogram
chunky	- pointer to an array of chunky bytes
width	- width to be added [pixels]
height	- lines to be added [rows]
palette	- pointer to a palette created with CreatePaletteA()
taglist	- pointer to an array of TagItems

### TAGS

RND\_SourceWidth (UWORD) - Total width of the chunky array [pixels].  
 Default - equals to the specified width.

RND\_ProgressHook (ULONG) - Pointer to a callback hook  
 structure for progress display operations. Refer to  
 render/renderhooks.h for further information.  
 Default - NULL.

### RESULTS

success - return value to indicate whether the operation succeeded.  
 You must at least check for ADDH\_SUCCESS. Adding data to  
 histograms may fail at any time and the histogram may  
 thereof get inaccurate.

## NOTES

- It is not possible with this function to directly add a chunky array that represents a HAM color scheme. In this case you have to convert it to an RGB array via `Chunky2RGBA()`, and then to call `AddRGBImageA()`.
- This function may call the progress callback Hook with the `PMSGTYPE_LINES_ADDED` message.

## SEE ALSO

`AddRGBImageA()`, `CreateHistogramA()`, `render/renderhooks.h`

## 1.3 render.library/AddHistogramA

## NAME

`AddHistogramA` - add a histogram to another histogram.

`AddHistogram` - varargs stub for `AddHistogramA`.

## SYNOPSIS

```
success = AddHistogramA(desthistogram, sourcehistogram, taglist)
d0              a0              a1              a2
```

```
ULONG AddHistogramA(APTR, APTR, struct TagItem *)
```

```
ULONG AddHistogram(APTR, APTR, tag, ..., TAG_DONE)
```

## FUNCTION

This function adds a histogram to another histogram, according to the following scheme:

`desthistogram + sourcehistogram -> desthistogram`

The color information contained in the source histogram will be added to the destination histogram.

## INPUTS

<code>desthistogram</code>	- pointer to destination histogram
<code>sourcehistogram</code>	- pointer to source histogram
<code>taglist</code>	- pointer to an array of <code>TagItems</code>

## TAGS

`RND_Weight (UWORD)` - multiplication factor for the number of representations in `sourcehistogram`

## RESULTS

`success` - return value to indicate whether the operation succeeded. You must at least check for `ADDH_SUCCESS`. Adding data to histograms may fail at any time and the histogram may thereof get inaccurate.

## SEE ALSO

`CreateHistogramA()`

---

## 1.4 render.library/AddRGB

### NAME

AddRGB - add a RGB value to a histogram.

### SYNOPSIS

```
success = AddRGB(histogram, RGB, count)
d0          a0          d0  d1
```

```
ULONG AddRGB(APTR, ULONG, ULONG)
```

### FUNCTION

This function adds a given number of representations for a single RGB value to a histogram.

### INPUTS

histogram	- pointer to a histogram
RGB	- RGB value to be added
count	- number of representations for that RGB value

### RESULTS

success - return value to indicate whether the operation succeeded. You must at least check for ADDH\_SUCCESS. Adding data to histograms may fail at any time and the histogram may thereof get inaccurate.

### SEE ALSO

AddRGBImageA(), CreateHistogramA()

## 1.5 render.library/AddRGBImageA

### NAME

AddRGBImageA - add an array of RGB data to a histogram.  
AddRGBImage - varargs stub for AddRGBImageA.

### SYNOPSIS

```
success = AddRGBImageA(histogram, rgb, width, height, taglist)
d0          a0          a1  d0    d1    a2
```

```
ULONG AddRGBImageA(APTR, ULONG *, UWORD, UWORD, struct TagItem *)
```

```
ULONG AddRGBImage(APTR, ULONG *, UWORD, UWORD, tag, ..., TAG_DONE)
```

### FUNCTION

This function adds an array of RGB pixels to a histogram. The color information contained in the RGB array gets stored in the histogram.

### INPUTS



histogram - pointer to a histogram  
 rgb - pointer to an array of RGB data  
 width - width to be added [pixels]  
 height - lines to be added [rows]  
 taglist - pointer to an array of TagItems

## TAGS

RND\_SourceWidth (UWORD) - Total width of the RGB array [pixels].  
 Default - equals to the specified width.

RND\_ProgressHook (ULONG) - Pointer to a callback hook  
 structure for progress display operations. Refer to  
 render/renderhooks.h for further information.  
 Default - NULL.

## RESULTS

success - return value to indicate whether the operation succeeded.  
 You must at least check for ADDH\_SUCCESS. Adding data to  
 histograms may fail at any time and the histogram may  
 thereof get inaccurate.

## NOTES

- This function may call the progress callback Hook with  
 the PMSGTYPE\_LINES\_ADDED message.

## SEE ALSO

AddRGB(), CreateHistogramA(), render/renderhooks.h

## 1.6 render.library/AllocRenderMem

## NAME

AllocRenderMem - allocate memory from a render-memhandler.

## SYNOPSIS

```

mem = AllocRenderMem(rendermemhandler, size)
d0                                a0                                d0

```

```

APTR AllocRenderMem(APTR, ULONG)

```

## FUNCTION

AllocRenderMem will allocate a memory block from a  
 render-memhandler, or from public memory. If there is no memory  
 block of the requested size available, NULL will be returned.  
 You must check this return value. The request may fail at any  
 time. Every call to this function must be followed by a call to  
 FreeRenderMem.

## INPUTS

rendermemhandler - pointer to a render-memhandler, or NULL.  
 If you pass NULL, MEMF\_ANY will be used.  
 size - the size of the desired block in bytes

## RESULTS

mem - pointer to a block of memory, or NULL if the

allocation failed.

#### NOTES

There is no real need for this function being available to you, except for helping you to create a smart, lean and sexy memory management - the idea is to enable your application and the library to share a particular memory pool.

#### SEE ALSO

`FreeRenderMem()`, `AllocRenderVec()`, `CreateRMHandlerA()`

## 1.7 render.library/AllocRenderVec

#### NAME

`AllocRenderVec` - allocate memory from a render-memhandler, and keep track of the allocated size and the memhandler itself.

#### SYNOPSIS

```
mem = AllocRenderVec(rendermemhandler, size)
d0                                     a0                                     d0
```

`APTR AllocRenderVec(APTR, ULONG)`

#### FUNCTION

`AllocRenderVec` will allocate a memory block from a render-memhandler, or from public memory. If there is no memory block of the requested size available, `NULL` will be returned. You must check this return value. The request may fail at any time. Any call to this function must be followed by a call to `FreeRenderVec()`. `AllocRenderVec()` keeps track of the allocated size and the memhandler itself.

#### INPUTS

`rendermemhandler` - pointer to a render-memhandler, or `NULL`.  
If you pass `NULL`, `MEMF_ANY` will be used.  
`size` - the size of the desired block in bytes

#### RESULTS

`mem` - pointer to a block of memory, or `NULL` if the allocation failed.

#### SEE ALSO

`FreeRenderVec()`, `AllocRenderVecClear()`, `AllocRenderMem()`, `CreateRMHandlerA()`

## 1.8 render.library/AllocRenderVecClear

#### NAME

`AllocRenderVecClear` - allocate a clear block of memory from a

---

## render-memhandler

## SYNOPSIS

```
mem = AllocRenderVecClear(rendermemhandler, size)
d0          a0          d0
```

```
APTR AllocRenderVecClear(APTR, ULONG)
```

## FUNCTION

AllocRenderVecClear will allocate a clear memory block from a render-memhandler, or from public memory. If there is no memory block of the requested size available, NULL will be returned. You must check this return value. The request may fail at any time. Any call to this function must be followed by a call to FreeRenderVec(). AllocRenderVecClear() keeps track of the allocated size and the memhandler itself.

## INPUTS

rendermemhandler - pointer to a render-memhandler, or NULL.  
If you pass NULL, MEMF\_ANY will be used.  
size - the size of the desired block in bytes

## RESULTS

mem - pointer to a block of memory filled with NULL bytes, or NULL if the allocation failed.

## SEE ALSO

FreeRenderVec(), AllocRenderVec(), AllocRenderMem(),  
CreateRMHandlerA()

## 1.9 render.library/ApplyAlphaChannelA

## NAME

ApplyAlphaChannelA - compose RGB array via alpha-channel.  
ApplyAlphaChannel - varargs stub for ApplyAlphaChannelA.

## SYNOPSIS

```
ApplyAlphaChannelA(sourcearray, width, height, destarray, tags)
a0          d0      d1      a1      a2
```

```
void ApplyAlphaChannelA(ULONG *, UWORD, UWORD, ULONG *,
                        struct TagItem *)
```

```
void ApplyAlphaChannel(ULONG *, UWORD, UWORD, ULONG *,
                        tag, ..., TAG_DONE)
```

## FUNCTION

This function uses the alpha-channel information (by default taken from the source array) to perform a composition of the source and destination array. The result will be written to the destination array.

## INPUTS

sourcearray - pointer to an ARGB array

width	- width [pixels]
height	- height [rows]
destarray	- pointer to a RGB array
taglist	- pointer to an array of TagItems

## TAGS

RND\_SourceWidth (UWORD) - Total width of the source array [pixels].  
Default - equals to the specified width.

RND\_DestWidth (UWORD) - Total width of the dest array [pixels].  
Default - equals to the specified width.

RND\_AlphaChannel (ULONG) - Pointer to the alpha-channel array.  
Default - equals to the pointer to the source array.

RND\_AlphaModulo (UWORD) - Pixel offset that will be used when  
proceeding from one alpha-channel value to the next [bytes].  
Default - 4.

RND\_AlphaWidth (ULONG) - Total width of the alpha-channel array  
[pixels]. Default - equals to the specified width.

## RESULTS

none

## SEE ALSO

InsertAlphaChannelA(), ExtractAlphaChannelA(), MixAlphaChannelA()

## 1.10 render.library/BestPen

## NAME

BestPen - find the best matching pen.

## SYNOPSIS

```
pen = BestPen(palette, RGB)
d0          a0          d0
```

```
LONG BestPen(APTR, ULONG)
```

## FUNCTION

Determine a palette's pen number that matches best a given RGB value.

## INPUTS

palette	- palette created with CreatePaletteA()
RGB	- RGB value to find a match for

## RESULTS

pen	- pen number or -1 if no pen could be found. (Usually this occurs when the palette is empty.)
-----	--

## 1.11 render.library/Chunky2BitMapA

### NAME

Chunky2BitMapA - convert chunky data to bitplanes.  
 Chunky2BitMap - varargs stub for Chunky2BitMapA.

### SYNOPSIS

```
Chunky2BitMapA(chunky, sx, sy, width, height, bitmap, dx, dy, taglist)
               a0      d0 d1 d2      d3      a1      d4 d5 a2

void Chunky2BitMapA(UBYTE *, UWORD, UWORD, UWORD, UWORD,
                   struct BitMap *, UWORD, UWORD, struct TagItem *)

void Chunky2BitMap(UBYTE *, UWORD, UWORD, UWORD, UWORD,
                  struct BitMap *, UWORD, UWORD, tag, ..., TAG_DONE)
```

### FUNCTION

Converts an array of chunky bytes to the bitplanes associated with a BitMap structure. You can specify clip areas both inside the chunky array and the BitMap. This function merges the data into the destination BitMap if required. BMF\_INTERLEAVED is also handled.

You may only process BMF\_STANDARD bitmaps with this function.

### INPUTS

chunky	- pointer to an array of chunky bytes
sx	- left edge inside the chunky array [pixels]
sy	- top edge inside the chunky array [rows]
width	- width [pixels]
height	- height [rows]
bitmap	- pointer to an initialized BitMap structure
dx	- destination left edge inside BitMap [pixels]
dy	- destination top edge inside BitMap [rows]
taglist	- pointer to an array of TagItems

### TAGS

RND\_SourceWidth (UWORD) - Total width of the chunky array [pixels].  
 Default - equals to the specified width.

RND\_PenTable (ULONG) - Pointer to a table of 256 UBYTES  
 for a secondary conversion of the pen numbers.  
 Default - NULL.

### RESULTS

none

### IMPORTANT NOTES

Starting with v39, you are not allowed to assume foreign BitMap structures being of a planar type. You may pass a BitMap structure to this function only if the BMF\_STANDARD flag is set.

Also remember to set-up your own BitMap structure with the BMF\_STANDARD flag if you wish to convert it with this function. Consider Chunky2BitMapA() being low-level. The BitMap structure involved here is intended to hold planar information. Since v39,

this is different to what `graphics.library` might associate with a `BitMap` structure.

Do not use this function with `BitMap` structures that are actually being displayed. If you wish to transfer chunky bytes to visible areas, use `graphics.library` functions, such as `WriteChunkyPixels()`, or `WritePixelFormat8()`. You may also use `Chunky2BitMapA()` followed by `BltBitMapRastPort()` etc.

With a graphics card supplied, `WriteChunkyPixels()` can be hundreds times faster than `Chunky2BitMapA()` followed by `BltBitMapRastPort()`. If you have to provide backward compatibility, it is worth the effort to differentiate between Kick 2.x and OS 3.x. Use `Chunky2BitMapA()` in the first case, `WritePixelFormat8()` in the second case, and `WriteChunkyPixels()` if the system runs under OS3.1 and is supplied with a graphics card.

SEE ALSO

`Planar2ChunkyA()`, `graphics/gfx.h`,  
`graphics.library/WriteChunkyPixels()`

## 1.12 render.library/Chunky2RGBA

NAME

`Chunky2RGBA` - convert an array of chunky bytes to RGB data.  
`Chunky2RGB` - varargs stub for `Chunky2RGBA`.

SYNOPSIS

```
success = Chunky2RGBA(chunky,width,height,rgb,palette,taglist)
d0              a0      d0      d1      a1  a2      a3
```

```
ULONG Chunky2RGBA(UBYTE *,UWORD,UWORD,ULONG *,APTR,struct TagItem *)
```

```
ULONG Chunky2RGB(UBYTE *,UWORD,UWORD,ULONG *,APTR,tag,...,TAG_DONE)
```

FUNCTION

This function converts an array of chunky bytes to RGB data.

INPUTS

<code>chunky</code>	- pointer to an array of chunky bytes
<code>width</code>	- width to be converted [pixels]
<code>height</code>	- height to be converted [rows]
<code>rgb</code>	- pointer to RGB destination buffer
<code>palette</code>	- pointer to a palette created with <code>CreatePaletteA()</code>
<code>taglist</code>	- pointer to an array of <code>TagItems</code>

TAGS

`RND_SourceWidth` (UWORD) - Total width of the chunky array [pixels].  
 Default - equals to the specified width.

`RND_DestWidth` (UWORD) - Total width of the RGB array [pixels].  
 Default - equals to the specified width.

---

RND\_ColorMode (ULONG) - Color mode that defines how to determine a pixel's actual color. Valid types:

COLORMODE_CLUT	- normal palette lookup
COLORMODE_HAM8	- HAM8 mode palette lookup
COLORMODE_HAM6	- HAM6 mode palette lookup

Default - COLORMODE\_CLUT.

RND\_LeftEdge (UWORD) - Horizontal starting position inside the chunky array [pixels]. This is mainly intended to allow the conversion of HAM clip areas.

Default - 0.

RND\_ProgressHook (ULONG) - Pointer to a callback hook structure for progress display operations. Refer to render/renderhooks.h for further information.

Default - NULL.

RND\_PenTable (ULONG) - Pointer to a table of 256 UBYTES for an additional conversion of the pen numbers.

Default - NULL.

Note: This is not defined for HAM modes.

RND\_LineHook (ULONG) - Pointer to a callback hook structure for line-related operations during conversion. This hook is executed once before a line is processed and once after it has been completed. Refer to render/renderhooks.h for further information.

Default - NULL.

#### RESULTS

success - CONV\_SUCCESS to indicate that the operation succeeded. Currently, the only reason for this function to fail is CONV\_CALLBACK\_ABORTED.

#### NOTES

- This function may call the progress callback Hook with the PMSGTYPE\_LINES\_CONVERTED message type.

#### SEE ALSO

render/render.h

## 1.13 render.library/ChunkyArrayDiversityA

#### NAME

ChunkyArrayDiversityA - calculate chunky array adaptibility  
 ChunkyArrayDiversity - varargs stub for ChunkyArrayDiversityA.

#### SYNOPSIS

```
diversity = ChunkyArrayDiversityA(chunkyarray,palette,
d0                                a0                a1
                                width,height,taglist)
```

d0      d1      a2

```
LONG ChunkyArrayDiversityA(UBYTE *,APTR,UWORD,UWORD,
                           struct TagItem *)
```

```
LONG ChunkyArrayDiversity(UBYTE *,UWORD,UWORD,
                           tag,...,TAG_DONE)
```

#### FUNCTION

Calculate a chunky array's adaptibility to a palette or mapping-engine.

#### INPUTS

chunkyarray - pointer to an array of chunky bytes  
 palette     - the chunky array's palette  
 width       - width to be processed [pixels]  
 height      - height to be processed [pixels]  
 taglist     - pointer to an array of tagitems

#### TAGS

RND\_Palette (ULONG) - pointer to a palette. Either this or the RND\_MapEngine argument is obligatory!

RND\_MapEngine (ULONG) - pointer to a mapping-engine. Either this or the RND\_Palette argument is obligatory!

RND\_SourceWidth (UWORD) - Total width of the RGB array [pixels].  
 Default - equals to the specified width.

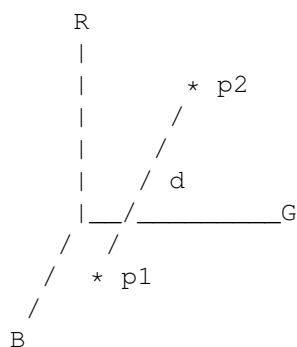
RND\_Interleave (ULONG) - number of pixels to skip horizontally.  
 Default - 0.

#### RESULTS

diversity - an indicator for the chunky array's adaptibility to the given palette or mapping-engine. 0 indicates perfect adaptibility, 195075 indicates the worst case.

#### IMPLEMENTATION

Every color in the source array is adapted to the specified palette or mapping-engine. Now assume p2 is the RGB of the palette entry that matches best with the requested color p1.



This function sums up the diversity  $d^2$  for every color in the source array, and divides it by the number of pixels processed:



$$D = \frac{1}{\text{pixels}} \cdot \frac{\sum_{k=1}^{\text{pixels}} d(k)^2}{\text{pixels}}$$

D is returned, so what you get is the average  $d^2$  for all RGB values that appear in the array.  $d^2$  is summed up instead of  $d$  because it is a better indicator for most purposes.

Best case - all pixels match perfectly:  $D(\min) = 0$ .

Worst case - all pixels are white, and the only available color in the palette is black:  $D(\max) = 255^2 + 255^2 + 255^2 = 195075$ .

#### EXAMPLE

D indicates a palette's applicability for the given chunky array. You can e.g. use it to decide whether error diffusion should be enabled for `ConvertChunkyA()`.

```
dithermode = DITHERMODE_NONE;
```

```
D = ChunkyArrayDiversity(sourcearray, palette, width, height,
    RND_Palette, destpalette, RND_Interleave, 4, TAG_DONE);
```

```
if (D > threshold)
{
    dithermode = DITHERMODE_FS;
}
```

```
ConvertChunky(sourcearray, palette, width, height, destarray,
    destpalette, RND_DitherMode, dithermode, TAG_DONE);
```

The interleave factor increases speed drastically. You gain a lot of performance if only dithering can be avoided sometimes.

#### SEE ALSO

`RGBArrayDiversityA()`

## 1.14 render.library/ConvertChunkyA

#### NAME

`ConvertChunkyA` - convert an array of chunky bytes to a new palette.  
`ConvertChunky` - varargs stub for `ConvertChunkyA`.

#### SYNOPSIS

```
success = ConvertChunkyA(source, sourcepalette, width, height, dest,
    d0          a0      a1          d0      d1      a2
    destpalette, taglist)
    a3          a4
```

```
ULONG ConvertChunkyA(UBYTE *, APTR, UWORD, UWORD, UBYTE *,
    APTR, struct TagItem *)
```

```
ULONG ConvertChunky(UBYTE *,APTR,UWORD,UWORD,UBYTE *,
                    APTR,tag,...,TAG_DONE)
```

#### FUNCTION

This function converts a source array of chunky bytes to another chunky array, and adapts it to a new palette.

#### INPUTS

source	- pointer to a source array of chunky bytes
sourcepalette	- pointer to the source array's palette
width	- width to be converted [pixels]
height	- height to be converted [rows]
dest	- pointer to the destination chunky array
destpalette	- pointer to the destination array's palette
taglist	- pointer to an array of TagItems

#### TAGS

RND\_SourceWidth (UWORD) - Total width of source array [pixels].  
Default - equals to the specified width.

RND\_DestWidth (UWORD) - Total width of dest array [pixels].  
Default - equals to the specified width.

RND\_DitherMode (UWORD) - Error diffusion mode. Valid types:

DITHERMODE_NONE	- no error diffusion
DITHERMODE_FS	- Floyd-Steinberg Does not handle a dither amount.
DITHERMODE_RANDOM	- Random dithering Handles RND_DitherAmount.
DITHERMODE_EDD	- EDD dithering Does not handle a dither amount.

Default - DITHERMODE\_NONE.

RND\_DitherAmount (UWORD) - Dither intensity (0-255).  
Only valid with certain dither modes (see above).  
Default - 128.

RND\_LineHook (ULONG) - Pointer to a callback hook structure for line-related operations during conversion. This hook is executed once before a line is rendered and once after a line has been completed. Refer to render/renderhooks.h for further information.  
Default - NULL.

RND\_OffsetColorZero (UWORD) - First pen number to appear in the rendered chunky image. This offset will be added to the palette's indices. Default - 0.  
Note: RND\_PenTable overrides RND\_OffsetColorZero.

RND\_PenTable (ULONG) - Pointer to a table of 256 UBYTES for a secondary conversion of the pen numbers.  
Default - NULL.

---

Note: RND\_PenTable overrides RND\_OffsetColorZero.

RND\_ProgressHook (ULONG) - Pointer to a callback hook structure for progress display operations. Refer to render/renderhooks.h for further information.  
Default - NULL.

RND\_ScaleEngine (ULONG) - Pointer to a scaling-engine. This scaling-engine will be used to scale the source data stream before it is being rendered.  
Default - NULL.

Notes:

- The scaling engine's destination width and height specifications override the width and height arguments for RenderA().
- The scaling-engine must have been created with PIXFMT\_CHUNKY\_CLUT.

RND\_MapEngine (ULONG) - Pointer to a mapping-engine. This can greatly improve performance.  
Default - NULL.

Notes:

- The mapping-engine must have been created for the destination palette that is specified for ConvertChunkyA().
- With dithering enabled, you better do not specify a mapping-engine which was created with a link to a histogram. Accurate conversion could not be guaranteed in this case.

#### RESULTS

success - returncode to indicate whether the operation succeeded. You must at least check for CONV\_SUCCESS.

#### NOTES

- This function may call the progress callback Hook with the PMSGTYPE\_LINES\_CONVERTED message type.

#### SEE ALSO

MapChunkyArrayA(), RenderA(), CreatePenTableA(), CreatePaletteA(), render/render.h, render/renderhooks.h

## 1.15 render.library/CountRGB

#### NAME

CountRGB - count a RGB value in a histogram.

#### SYNOPSIS

```
count = CountRGB(histogram, RGB)
d0          a0          d0
```

ULONG CountRGB(APTR,ULONG)

#### FUNCTION

Counts the number of occurrences for a given RGB value. The result may depend on the histogram's accuracy.

#### INPUTS

histogram - pointer to a histogram

#### RESULTS

count - number of representations for the specified RGB value.

#### NOTE

You only get the exact result for 24bit histograms. The lower the resolution, the more colors are actually put together into one 'category' of similar colors. A 24bit histogram differentiates 16,7 million colors, a 15bit histogram, for instance, only 32768.

## 1.16 render.library/CreateAlphaArrayA

#### NAME

CreateAlphaArrayA - create an alpha-channel for a RGB array.  
CreateAlphaArray - varargs stub for CreateAlphaArrayA.

#### SYNOPSIS

```
CreateAlphaArrayA(rgbarray,width,height,tags)
                  a0      d0      d1      a1
```

```
void CreateAlphaArrayA(ULONG *,UWORD,UWORD,struct TagItem *)
```

```
void CreateAlphaArray(ULONG *,UWORD,UWORD,tag,...,TAG_DONE)
```

#### FUNCTION

This function creates an alpha-channel for the given RGB array. The alpha-channel will be the difference in brightness towards black (0x000000), or optionally towards another RGB value.

#### INPUTS

rgbarray	- pointer to a RGB array
width	- width [pixels]
height	- height [rows]
taglist	- pointer to an array of TagItems

#### TAGS

RND\_SourceWidth (UWORD) - Total width of the RGB array [pixels].  
Default - equals to the specified width.

RND\_AlphaChannel (ULONG) - Alpha-channel destination array.  
Default - equals to rgbarray. the upmost byte in the RGB array is a good place for an alpha-channel, usually.

RND\_AlphaWidth (UWORD) - Total width of the alpha-channel array

[pixels]. Default - equals to the specified width.

RND\_AlphaModulo (UWORD) - Alpha-channel pixel modulo [bytes].  
Default - 4

RND\_MaskRGB (ULONG) - The RGB value to compute the alpha-channel  
towards. Default - 0x000000

#### RESULTS

none

## 1.17 render.library/CreateHistogramA

#### NAME

CreateHistogramA - create and set up a histogram.  
CreateHistogram - varargs stub for CreateHistogramA.

#### SYNOPSIS

```
hst = CreateHistogramA(taglist)
d0                                a1
```

```
APTR CreateHistogramA(struct TagItem *)
```

```
APTR CreateHistogram(tag,...,TAG_DONE)
```

#### FUNCTION

Allocates and initializes a histogram.

#### INPUTS

taglist - pointer to an array of TagItems

#### TAGS

RND\_RMHandler (ULONG) - pointer to a render-memhandler  
created with CreateRMHandlerA().  
Default - NULL.

RND\_HSType (UWORD) - Type of histogram. Valid types:

HSTYPE_12BIT	- 12bit dynamic histogram
HSTYPE_15BIT	- 15bit dynamic histogram
HSTYPE_18BIT	- 18bit dynamic histogram
HSTYPE_21BIT	- 21bit dynamic histogram
HSTYPE_24BIT	- 24bit dynamic histogram
HSTYPE_12BIT_TURBO	- 12bit tabular histogram
HSTYPE_15BIT_TURBO	- 15bit tabular histogram
HSTYPE_18BIT_TURBO	- 18bit tabular histogram

Default - HSTYPE\_15BIT\_TURBO.

#### RESULTS

histogram - pointer to a histogram ready for usage,  
or NULL if something went wrong.

#### SEE ALSO

```
DeleteHistogram(), QueryHistogram(), render/render.h
```

## 1.18 render.library/CreateMapEngineA

### NAME

CreateMapEngineA - create a mapping-engine.  
 CreateMapEngine - varargs stub for CreateMapEngineA.

### SYNOPSIS

```
engine = CreateMapEngineA(palette,taglist)
d0                      a0          a1
```

```
ULONG CreateMapEngineA(APTR,struct TagItem *)
```

```
ULONG CreateMapEngine(APTR,tag,...,TAG_DONE)
```

### FUNCTION

This function will create and prepare a mapping-engine for usage. Mapping-engines are highly optimized low-level conversion units for rendering to a specific palette.

Mapping-engines are immediately dependent from a palette, and they may be additionally coupled with a histogram. In the latter case, mapping-engines adapt only RGB values that can be found in the histogram.

Mapping-engines do not incorporate the contents of their palettes and histograms. They are getting notified, and they automatically update themselves when needed. This update takes place only on demand, i.e. whenever you pass a mapping-engine to a function. The mapping-engine checks if any changes applied to the palette or histogram, and it reconstructs itself. The palette's pen-adaption buffers are updated as well - palettes and mapping-engines are just good friends.

### INPUTS

```
palette      - pointer to a palette
taglist      - pointer to an array of TagItems
```

### TAGS

```
RND_RMHandler (ULONG) - Pointer to a render-memhandler.
    This will be used for the mapping-engine's
    buffers. Default - The palette's memhandler.
```

```
RND_Histogram (ULONG) - Pointer to a histogram.
    The histogram must be of a HSTYPE_..._TURBO type,
    and have the same resolution as the palette.
    Default - none.
```

### RESULTS

```
engine - pointer to a mapping-engine ready for usage,
        or NULL if something went wrong.
```

### NOTES

You must free a mapping-engine before you DeletePalette() the palette it is dependent from. The same applies for the histogram.

SEE ALSO

DeleteMapEngine(), MapRGBArrayA(), CreateRMHandlerA()

## 1.19 render.library/CreatePaletteA

NAME

CreatePaletteA - create a palette.  
CreatePalette - vararg stub for CreatePaletteA.

SYNOPSIS

```
palette = CreatePaletteA(taglist)
d0          a1

APTR CreatePaletteA(struct TagItem *)

APTR CreatePalette(tag, ..., TAG_DONE)
```

FUNCTION

This function creates and initializes a palette that can hold up to 256 color entries.

INPUTS

taglist - pointer to an array of TagItems

TAGS

RND\_HSType (ULONG) - the palette's resolution. Palette adaption accuracy and memory consumption depend on this constant. A palette's resolution is specified analogously to a histogram's resolution. Valid types:

HSTYPE\_12BIT  
HSTYPE\_15BIT  
HSTYPE\_18BIT

Default - HSTYPE\_15BIT.

RND\_RMHandler (ULONG) - pointer to a render-memhandler that was created with CreateRMHandler(). Default - NULL.

RESULTS

palette - a palette ready for usage,  
or NULL if something went wrong.

SEE ALSO

DeletePalette(), ImportPaletteA(), ExportPaletteA(),  
ExtractPaletteA(), FlushPalette(), render/render.h

---

## 1.20 render.library/CreatePenTableA

### NAME

CreatePenTableA - create a pen conversion table.  
 CreatePenTable - varargs stub for CreatePenTableA.

### SYNOPSIS

```
CreatePenTableA(chunky,oldpalette,width,height,newpalette,
                a0      a1          d0      d1      a2
                pentab,taglist)
                a3      a4
```

```
void CreatePenTableA(UBYTE *,APTR,UWORD,UWORD,APTR,
                    UBYTE *,struct TagItem *)
```

```
void CreatePenTable(UBYTE *,APTR,UWORD,UWORD,APTR,
                   UBYTE *,tag,...,TAG_DONE)
```

### FUNCTION

This function creates a table for the conversion of a particular array of chunky bytes. It scans through the chunky array, adapts the found palette entries to a new palette, and generates an output table of 256 UBYTES. The resulting table is handled as follows:

```
new_pen_number = pentab[old_pen_number]
```

### INPUTS

```
chunky      - pointer to an array of chunky bytes
oldpalette  - pointer to the original palette
width       - width to be processed [pixels]
height      - height to be processed [rows]
newpalette  - pointer to a palette to be adapted to
pentab      - pointer to the destination table
taglist     - pointer to an array of TagItems
```

### TAGS

```
RND_SourceWidth (UWORD) - Total width of the chunky array [pixels].
    Default - equals to the specified width.
```

```
RND_PenTable (ULONG) - Pointer to a table of 256 UBYTES
    for a secondary conversion of the pen numbers.
    Default - NULL.
```

### RESULTS

```
none
```

### NOTES

The destination table is assumed to have 256 entries, with no respect to what color indices actually occur in the chunky array.

### SEE ALSO

```
ConvertChunkyA(), CreatePaletteA()
```



## 1.21 render.library/CreateRMHandlerA

### NAME

CreateRMHandlerA - Create and set up a memory handler.  
 CreateRMHandler - varargs stub for CreateRMHandlerA.

### SYNOPSIS

```
rmh = CreateRMHandlerA(taglist)
d0                                     a1

APTR CreateRMHandlerA(struct TagItem *)

APTR CreateRMHandler(tag, ..., TAG_DONE)
```

### FUNCTION

This function allocates and initializes a render-memhandler. This is a custom memory resource manager for histograms, rendering, palettes, and many other render.library objects and functions. You may use a render-memhandler for your own purposes, too.

A render-memhandler helps to avoid memory fragmentation as well as extreme stressing of the system's public memory lists. Private memory management is supported as well as v39 exec pools and common public memory. Future versions might provide more types of memory management.

### INPUTS

taglist - pointer to an array of TagItems

### TAGS

RND\_MemType (UWORD) - type of memory management. Valid types:

RMHTYPE_POOL	- v39 exec dynamic pool
RMHTYPE_PRIVATE	- you supply a private memory pool
RMHTYPE_PUBLIC	- use common public memory

Default - RMHTYPE\_PUBLIC.

RND\_MemBlock (ULONG) - pointer to a block of memory used for private memory management. This tag is obligatory if you specify RMHTYPE\_PRIVATE and is ignored otherwise.  
 Default - none.

RND\_MemSize (ULONG) - size of the memory block used for private memory management. This tag is obligatory if you specify RMHTYPE\_PRIVATE and is ignored otherwise.  
 Default - none.

RND\_MemFlags (ULONG) - memory flags, as defined in exec/memory.h. These are ignored for RMHTYPE\_PRIVATE. Default - MEMF\_ANY.

### RESULTS

rmh - a render-memhandler ready for usage, or NULL if something went wrong.

#### NOTES

You must check for the presence of exec v39 before you create a memhandler with RMHTYPE\_POOL specified.

#### SEE ALSO

AllocRenderMem(), AllocRenderVec(), DeleteRMHandler(),  
exec.library/CreatePool(), render/render.h, exec/memory.h

## 1.22 render.library/CreateScaleEngineA

#### NAME

CreateScaleEngineA - Create a scaling-engine.  
CreateScaleEngine - varargs stub for CreateScaleEngineA.

#### SYNOPSIS

```
scaleengine = CreateScaleEngineA(sourcewidth,sourceheight,
d0                                d0            d1
                                destwidth,destheight,taglist)
                                d2            d3            a1
```

```
APTR CreateScaleEngineA(UWORD,UWORD,UWORD,
                        UWORD,struct TagItem *)
```

```
APTR CreateScaleEngine(UWORD,UWORD,UWORD,
                        UWORD,tag,...,TAG_DONE)
```

#### FUNCTION

Allocates and initializes a scaling-engine for a specific set of scaling parameters. Once set up, a scaling-engine is highly optimized for its particular parameter specifications.

#### INPUTS

```
sourcewidth - source width [pixels]
sourceheight - source height [rows]
destwidth    - destination width [pixels]
destheight   - destination height [rows]
taglist      - pointer to an array of tag items
```

#### TAGS

RND\_RMHandler (ULONG) - pointer to a render-memhandler,  
such as created with CreateRMHandlerA().  
Default - NULL.

RND\_PixelFormat (ULONG) - Type of data to process.  
Currently defined are PIXFMT\_CHUNKY\_CLUT  
and PIXFMT\_ORGB\_32. Default - PIXFMT\_CHUNKY\_CLUT.

RND\_Coordinates (ULONG) - Pointer to an array of coordinates  
for texturemapped scaling. This array consists of  
4 WORD pairs of x/y coordinates each. The array forms  
a trapezoid inside the destination buffer for the source

array being mapped to. Any parameters are valid, since clipping is fully implemented. Default - NULL.

#### RESULTS

engine - a scaling-engine ready for usage, or NULL if something went wrong.

#### SEE ALSO

DeleteScaleEngine(), ScaleA()

## 1.23 render.library/DeleteHistogram

#### NAME

DeleteHistogram - dispose a histogram.

#### SYNOPSIS

```
DeleteHistogram(histogram)
                a0
```

```
void DeleteHistogram(APTR)
```

#### FUNCTION

Removes a histogram and frees all associated memory.

#### INPUTS

histogram - pointer to a histogram  
Passing a NULL pointer is safe.

#### RESULTS

none

## 1.24 render.library/DeleteMapEngine

#### NAME

DeleteMapEngine - dispose a mapping-engine.

#### SYNOPSIS

```
DeleteMapEngine(engine)
                a0
```

```
void DeleteMapEngine(APTR)
```

#### FUNCTION

Removes a mapping-engine and frees all associated memory.

#### INPUTS

engine - pointer to a mapping-engine

#### NOTES

You must free mapping-engines before you delete the

---

palettes and histograms they're dependent from.

#### RESULTS

none

#### SEE ALSO

CreateMapEngine()

## 1.25 render.library/DeleteRMHandler

#### NAME

DeleteRMHandler - free a render-memhandler.

#### SYNOPSIS

```
DeleteRMHandler(rendermemhandler)
                a0
```

```
void DeleteRMHandler(APTR)
```

#### FUNCTION

DeleteRMHandler() will remove and free a previously created render-memhandler. That does not imply that any outstanding memory will be returned to the system or to whatever memory resources. You are responsible for freeing each memory block that you have allocated from a render-memhandler.

#### INPUTS

render-memhandler - a render-memhandler to be deleted.

#### RESULTS

none

#### NOTES

You are not allowed to call DeleteRMHandler() before every single byte has been returned to the memhandler.

#### SEE ALSO

CreateRMHandlerA()

## 1.26 render.library/DeletePalette

#### NAME

DeletePalette - dispose a palette.

#### SYNOPSIS

```
DeletePalette(palette)
                a0
```

```
void DeletePalette(APTR)
```

---

**FUNCTION**

This function deletes a palette and frees all associated memory.

**INPUTS**

palette - pointer to a palette created with CreatePaletteA()

**RESULTS**

none

**SEE ALSO**

CreatePaletteA(), FlushPalette()

## 1.27 render.library/DeleteScaleEngine

**NAME**

DeleteScaleEngine - dispose a scaling-engine.

**SYNOPSIS**

```
DeleteScaleEngine(engine)
                    a0
```

```
void DeleteScaleEngine(APTR)
```

**FUNCTION**

Deletes a scaling-engine and frees all associated memory.

**INPUTS**

engine - a scaling-engine to be removed

**RESULTS**

none

**SEE ALSO**

CreateScaleEngineA()

## 1.28 render.library/ExportPaletteA

**NAME**

ExportPaletteA - export a palette.

ExportPalette - varargs stub for ExportPaletteA.

**SYNOPSIS**

```
ExportPaletteA(palette,buffer,taglist)
               a0      a1      a2
```

```
ExportPaletteA(APTR,APTR,struct TagItem *)
```

```
ExportPalette(APTR,APTR,tag,...,TAG_DONE)
```

---

## FUNCTION

This function exports a palette (or a part of it) to a colortable.

## INPUTS

palette - pointer to a palette created with CreatePaletteA()  
 buffer - pointer to a destination buffer  
 taglist - pointer to an array of tag items

## TAGS

RND\_PaletteFormat (ULONG) - format of the color table to be exported. Valid types:

PALFMT\_RGB32 - ULONG red,green,blue  
 PALFMT\_RGB8 - ULONG 0x00rrggbb  
 PALFMT\_RGB4 - UWORD 0xrgb

Default - PALFMT\_RGB8.

RND\_FirstColor (ULONG) - first color entry to export.  
 Default - 0.

RND\_NumColors (ULONG) - number of colors to export.  
 Default - the number of colors inside the palette.

## RESULTS

none

## SEE ALSO

CreatePaletteA(), ImportPaletteA()

## 1.29 render.library/ExtractAlphaChannelA

## NAME

ExtractAlphaChannelA - extract alpha-channel from an ARGB array.  
 ExtractAlphaChannel - varargs stub for ExtractAlphaChannelA.

## SYNOPSIS

```
ExtractAlphaChannelA(argbarray,width,height,chunkyarray,tags)
                    a0          d0      d1      a1          a2
```

```
void ExtractAlphaChannelA(ULONG *,UWORD,UWORD,UBYTE *,
                        struct TagItem *)
```

```
void ExtractAlphaChannel(ULONG *,UWORD,UWORD,UBYTE *,
                        tag,...,TAG_DONE)
```

## FUNCTION

This function extracts the alpha-channel mask from an ARGB array and writes it to an array of chunky bytes.

## INPUTS

argbarray	- pointer to an ARGB array
width	- width [pixels]
height	- height [rows]
chunkyarray	- pointer to an array of chunky bytes
taglist	- pointer to an array of TagItems

## TAGS

RND\_SourceWidth (UWORD) - Total width of the ARGB array [pixels].  
Default - equals to the specified width.

RND\_DestWidth (UWORD) - Total width of the chunky array [pixels].  
Default - equals to the specified width.

## RESULTS

none

## SEE ALSO

InsertAlphaChannelA(), ApplyAlphaChannelA()

## 1.30 render.library/ExtractPaletteA

## NAME

ExtractPaletteA - extract a palette from a histogram.  
ExtractPalette - varargs stub for ExtractPaletteA.

## SYNOPSIS

```
success = ExtractPaletteA(histogram,palette,numcolors,taglist)
d0          a0          a1          d0          a2
```

```
ULONG ExtractPaletteA(APTR,ULONG *,UWORD,struct TagItem *)
```

```
ULONG ExtractPalettA(APTR,ULONG *,UWORD,tag,...,TAG_DONE)
```

## FUNCTION

This function extracts a given number of colors from a histogram and stores it in a palette.

## INPUTS

histogram	- pointer to a histogram
palette	- pointer to a palette created with CreatePaletteA()
numcolors	- number of entries to extract
taglist	- pointer to an array of TagItems

## TAGS

RND\_RMHandler (ULONG) - Custom memory handler created with CreateRMHandlerA(). This is used to handle intermediate buffers during quantization.  
Default - The histogram's memory handler.

RND\_ProgressHook (ULONG) - Pointer to a callback hook structure for progress display operations. Refer to render/renderhooks.h for further information.  
Default - NULL.

**RND\_RGBWeight** (ULONG) - R/G/B quantization factors.  
 They form a relative measurement between the R/G/B components, defining what color components should be preferred when the histogram gets decomposed.  
 Default - 0x010101 (all components are treat equally).

**RND\_ColorMode** (ULONG) - Color mode that defines how to determine a pixel's actual color. Currently this tag should only be set to **COLORMODE\_HAM6** if you extract a palette for the use with a HAM6 image.  
 Default - **COLORMODE\_CLUT**.

**RND\_FirstColor** (ULONG) - first color entry inside the palette that will be used for the extracted colors. See also **ImportPaletteA()** for further details. Default - 0.

**RND\_NewPalette** (ULONG) - if set to **TRUE**, this flag indicates that you want to dispose the current palette and create a new one. If set to **FALSE**, the new color entries are merged to the existing palette. Default - **TRUE**.

#### RESULTS

**success** - returncode to indicate whether the operation succeeded. You must at least check for **EXTP\_SUCCESS**.

#### NOTES

- This function may call the progress callback Hook with the **PMSGTYPE\_COLORS\_CHOSEN** message type.

#### SEE ALSO

**CreateHistogramA()**, **CreatePaletteA()**, **ImportPaletteA()**, **render/render.h**, **render/renderhooks.h**

## 1.31 render.library/FlushPalette

#### NAME

**FlushPalette** - flush all buffers from a palette.

#### SYNOPSIS

```
FlushPalette(palette)
           a0
```

```
void FlushPalette(APTR)
```

#### FUNCTION

This function flushes all buffers that might be associated with a palette.

#### INPUTS

**palette** - pointer to a palette that was created with **CreatePaletteA()**



RESULTS  
    none

SEE ALSO  
    DeletePalette()

## 1.32 render.library/FreeRenderMem

NAME  
    FreeRenderMem - return memory to a render-memhandler.

SYNOPSIS  
    FreeRenderMem(rendermemhandler,mem,size)  
                    a0                    a1  d0

    void FreeRenderMem(APTR,APTR,ULONG)

FUNCTION  
    Free a block of memory that was allocated with  
    AllocRenderMem().

INPUTS  
    rendermemhandler - pointer to the render-memhandler the block  
                            has been allocated from  
    mem                    - pointer to the memory block to be returned  
    size                    - size of that memory block [bytes]

RESULTS  
    NONE

SEE ALSO  
    AllocRenderMem(), CreateRMHandlerA()

## 1.33 render.library/FreeRenderVec

NAME  
    FreeRenderVec - return memory to a render-memhandler.

SYNOPSIS  
    FreeRenderVec(mem)  
                    a0

    void FreeRenderVec(APTR)

FUNCTION  
    Free a block of memory that was allocated with AllocRenderVec().

INPUTS  
    mem - pointer to a memory block  
            to be returned to its render-memhandler.

---

Passing a NULL pointer is safe.

#### RESULTS

NONE

#### SEE ALSO

`AllocRenderVec()`, `DeleteRMHandler()`

## 1.34 render.library/ImportPaletteA

#### NAME

`ImportPaletteA` - import a palette.  
`ImportPalette` - varargs stub for `ImportPaletteA`.

#### SYNOPSIS

```
ImportPaletteA(palette, table, entries, taglist)
               a0      a1      d0      a2

ImportPaletteA(APTR, APTR, UWORD, struct TagItem *)

ImportPalette(APTR, APTR, UWORD, tag, ..., TAG_DONE)
```

#### FUNCTION

This function imports entries from a color table (or from a palette) to a palette. You are allowed to import multiple times. When doing so, entries will be overwritten (not inserted). The palette automatically grows to the required number of entries. Remember to neither import more than 256 entries nor beyond the 256th entry.

#### INPUTS

`palette` - pointer to a palette created with `CreatePaletteA()`  
`table` - pointer to a source color table (or a `render.library` palette)  
`entries` - number of color entries to import  
`taglist` - pointer to an array of tag items

#### TAGS

`RND_PaletteFormat` (ULONG) - format of the color table to be imported. Valid types:

```
PALFMT_RGB32 - ULONG red, green, blue
PALFMT_RGB8  - ULONG 0x00rrggbb
PALFMT_RGB4  - UWORD 0xrgb
PALFMT_PALETTE - another palette
```

Default - `PALFMT_RGB8`.

`RND_FirstColor` (ULONG) - first color entry inside the palette to import to. Default - 0.

`RND_EHBPalette` (ULONG) - tag to indicate whether the imported colors should be interpreted as for an

Extra-Halfbrite picture. Default - FALSE.

RND\_NewPalette (ULONG) - if set to TRUE, this flag indicates that you want to dispose the current palette and import a new one. If set to FALSE, the palette is merged. Default - TRUE.

#### RESULTS

none

#### SEE ALSO

CreatePaletteA(), ExportPaletteA(), render/render.h

## 1.35 render.library/InsertAlphaChannelA

#### NAME

InsertAlphaChannelA - insert a alpha-channel mask to a RGB array.

InsertAlphaChannel - varargs stub for InsertAlphaChannelA.

#### SYNOPSIS

```
InsertAlphaChannelA(chunkyarray,width,height,rgbarray,tags)
                   a0          d0      d1      a1      a2
```

```
void InsertAlphaChannelA(UBYTE *,UWORD,UWORD,ULONG *,
                        struct TagItem *)
```

```
void InsertAlphaChannel(UBYTE *,UWORD,UWORD,ULONG *,
                        tag,...,TAG_DONE)
```

#### FUNCTION

This function inserts an array of chunky bytes to a RGB array. The resulting ARGB array can then be used for alpha-channel operations, such as provided with ApplyAlphaChannelA().

#### INPUTS

chunkyarray	- pointer to an array of chunky bytes
width	- width [pixels]
height	- height [rows]
rgbarray	- pointer to a RGB array
taglist	- pointer to an array of TagItems

#### TAGS

RND\_SourceWidth (UWORD) - Total width of the chunky array [pixels].  
Default - equals to the specified width.

RND\_DestWidth (UWORD) - Total width of the rgb array [pixels].  
Default - equals to the specified width.

#### RESULTS

none

#### SEE ALSO

ExtractAlphaChannelA(), ApplyAlphaChannelA()

## 1.36 render.library/MapChunkyArrayA

### NAME

MapChunkyArrayA - remap a chunky array.  
 MapChunkyArray - varargs stub for MapChunkyArrayA.

### SYNOPSIS

```
success = MapChunkyArrayA(engine,sourcearray,palette,width,
d0                a0      a1          a2      d0
                height,destarray,taglist)
                d1      a3          a4
```

```
ULONG MapChunkyArrayA(APTR,UBYTE *,APTR,UWORD,UWORD,UBYTE *,
                struct TagItem *)
```

```
ULONG MapChunkyArray (APTR,UBYTE *,APTR,UWORD,UWORD,UBYTE *,
                tag,...,TAG_DONE)
```

### FUNCTION

MapChunkyArrayA() maps an array of chunky bytes to another chunky array.

### INPUTS

engine - pointer to a mapping-engine created with CreateMapEngineA()  
 sourcearray - pointer to source array of chunky pixels  
 palette - pointer to source array's palette  
 width - width to be converted [pixels]  
 height - height to be converted [rows]  
 destarray - pointer to the destination chunky array  
 taglist - pointer to an array of TagItems

### TAGS

RND\_SourceWidth (UWORD) - Total width of the chunky array [pixels].  
 Default - equals to the specified width.

RND\_DestWidth (UWORD) - Total width of the chunky array [pixels].  
 Default - equals to the specified width.

RND\_PenTable (ULONG) - Pointer to a table of 256 UBYTES for a secondary conversion of the pen numbers.  
 Default - NULL.

### RESULTS

success - returncode to indicate whether the operation succeeded. You must at least check for CONV\_SUCCESS.

### NOTES

This function is considered low-level. Much more functionality is provided with ConvertChunkyA(). Mapping-engines may also be passed to ConvertChunkyA(), and MapChunkyArrayA() is only slightly faster in this case. ConvertChunkyA() is preferable under most

circumstances. If you need to remap lots of tiny images, you might prefer `MapChunkyArrayA()`, since it has got very few overhead.

SEE ALSO

`ConvertChunkyA()`, `MapRGBArrayA()`, `CreateMapEngineA()`

## 1.37 render.library/MapRGBArrayA

### NAME

`MapRGBArrayA` - map an RGB array to chunky bytes.  
`MapRGBArray` - varargs stub for `MapRGBArrayA`.

### SYNOPSIS

```
success = MapRGBArrayA(engine, rgbarray, width, height, chunky, taglist)
d0          a0          a1          d0          d1          a2          a3
```

```
ULONG MapRGBArrayA(APTR, ULONG *, UWORD, UWORD, UBYTE *, struct TagItem *)
```

```
ULONG MapRGBArray(APTR, ULONG *, UWORD, UWORD, UBYTE *, tag, ..., TAG_DONE)
```

### FUNCTION

`MapRGBArrayA()` maps an array of RGB data to a chunky array.

### INPUTS

<code>engine</code>	- pointer to a mapping-engine created with <code>CreateMapEngineA()</code>
<code>rgbarray</code>	- pointer to an array of RGB pixels
<code>width</code>	- width to be converted [pixels]
<code>height</code>	- height to be converted [rows]
<code>chunky</code>	- pointer to the destination array
<code>taglist</code>	- pointer to an array of <code>TagItems</code>

### TAGS

`RND_SourceWidth` (UWORD) - Total width of the RGB array [pixels].  
 Default - equals to the specified width.

`RND_DestWidth` (UWORD) - Total width of the chunky array [pixels].  
 Default - equals to the specified width.

`RND_PenTable` (ULONG) - Pointer to a table of 256 UBYTES for a secondary conversion of the pen numbers.  
 Default - NULL.

### RESULTS

`success` - returncode to indicate whether the operation succeeded. You must at least check for `CONV_SUCCESS`.

### NOTES

- Do not use `MapRGBArrayA()` with ARGB (A=alpha-channel) data. The upper byte must be set to zero!
- This function is considered low-level. Much more functionality is provided with `RenderA()`. Mapping-engines may also be passed to `RenderA()`, and `MapRGBArrayA()` is only slightly faster in this

case. `RenderA()` is preferable under most circumstances. If you need to render lots of tiny images, you might prefer `MapRGBArrayA()`, since it has got few overhead.

SEE ALSO

`RenderA()`, `MapChunkyArrayA()`, `CreateMapEngineA()`

## 1.38 render.library/MixAlphaChannelA

NAME

`MixAlphachannelA` - mix two arrays via alpha-channels.  
`MixAlphachannel` - varargs stub for `MixAlphachannelA`.

SYNOPSIS

```
MixAlphachannelA(sourcearray1,sourcearray2,width,height,destarray,
                  a0          a1          d0    d1    a2
                  tags)
                  a3
```

```
void MixAlphachannelA(ULONG *,ULONG *,UWORD,UWORD,ULONG *,
                     struct TagItem *)
```

```
void MixAlphachannel(ULONG *,ULONG*,UWORD,UWORD,ULONG *,
                    tag,...,TAG_DONE)
```

FUNCTION

This function does a 'weighted' alpha-channel composition of two RGB arrays and writes the result to a third RGB array. You may optionally disable one or both alpha-channel sources, in the latter case a 1:1 mix (without alpha-channel) will be performed.

INPUTS

<code>sourcearray1</code>	- pointer to an RGB array, source
<code>sourcearray2</code>	- pointer to an RGB array, source
<code>width</code>	- width [pixels]
<code>height</code>	- height [rows]
<code>destarray</code>	- pointer to an RGB array, destination
<code>taglist</code>	- pointer to an array of <code>TagItems</code>

TAGS

`RND_SourceWidth` (UWORD) - Total width of `sourcearray1` [pixels].  
 Default - equals to the specified width.

`RND_SourceWidth2` (UWORD) - Total width of `sourcearray2` [pixels].  
 Default - equals to the specified width.

`RND_DestWidth` (UWORD) - Total width of `destarray` [pixels].  
 Default - equals to the specified width.

`RND_AlphaChannel` (ULONG) - Pointer to an alpha-channel array for `sourcearray1`. This may be NULL. Default - equals to `sourcearray1`.

RND\_AlphaWidth (UWORD) - Total width of the first alpha-channel array [pixels]. Default - equals to RND\_SourceWidth.

RND\_AlphaModulo (UWORD) - Alpha-channel pixel modulo for the first alpha-channel array [bytes]. Default - 4

RND\_AlphaChannel2 (ULONG) - Pointer to an alpha-channel array for sourcearray2. This may be NULL. Default - equals to sourcearray2.

RND\_AlphaWidth2 (UWORD) - Total width of the second alpha-channel array [pixels]. Default - equals to RND\_SourceWidth2.

RND\_AlphaModulo2 (UWORD) - Alpha-channel pixel modulo for the second alpha-channel array [bytes]. Default - 4

#### RESULTS

none

#### NOTES

this function is much more generalized than ApplyAlphaChannelA() and MixRGBArrayA(), but it has got more overhead and might be noticeably slower. read the appropriate autodoc sections to see if one of the above mentioned functions suffice for your particular purpose.

#### SEE ALSO

ApplyAlphaChannelA(), MixRGBArrayA(), InsertAlphaChannelA(), ExtractAlphaChannelA()

## 1.39 render.library/MixRGBArrayA

#### NAME

MixRGBArrayA - mix two RGB arrays.  
MixRGBArray - varargs stub for MixRGBArrayA.

#### SYNOPSIS

```
MixRGBArrayA(sourcearray,width,height,destarray,ratio,tags)
               a0           d0      d1      a1           d2      a2
```

```
void MixRGBArrayA(ULONG *,UWORD,UWORD,ULONG *,UWORD,
                  struct TagItem *)
```

```
void MixRGBArray(ULONG *,UWORD,UWORD,ULONG *,UWORD,
                 tag,...,TAG_DONE)
```

#### FUNCTION

This function mixes sourcearray and destarray and writes the result to destarray. Unlike ApplyAlphaChannelA(), this function does not use any alpha-channel information.

#### INPUTS

sourcearray - pointer to an ARGB array  
width - width [pixels]

height - height [rows]  
 destarray - pointer to a RGB array  
 ratio - mix ratio (0...255)  
 taglist - pointer to an array of TagItems

## TAGS

RND\_SourceWidth (UWORD) - Total width of the source array [pixels].  
 Default - equals to the specified width.

RND\_DestWidth (UWORD) - Total width of the dest array [pixels].  
 Default - equals to the specified width.

## RESULTS

none

## SEE ALSO

ApplyAlphaChannelA()

## 1.40 render.library/Planar2ChunkyA

## NAME

Planar2ChunkyA - convert bitplane data to chunky bytes.  
 Planar2Chunky - varargs stub for Planar2ChunkyA.

## SYNOPSIS

```
Planar2ChunkyA(planetab,bytewidth,rows,depth,bytesperrow,
               a0      d0      d1      d2      d3
               chunkybuffer,taglist)
               a1      a2
```

```
void Planar2ChunkyA(PLANEPTR *,UWORD,UWORD,UWORD,UWORD,
                   UBYTE *,struct TagItem *)
```

```
void Planar2Chunky(PLANEPTR *,UWORD,UWORD,UWORD,UWORD
                   UBYTE *,tag,...,TAG_DONE)
```

## FUNCTION

This function converts raw bitplane-oriented (planar) graphics to an array of chunky bytes.

## INPUTS

planetab - pointer to a table of planepointers  
 bytewidth - width [bytes]. This must be an even number.  
 rows - height [rows]  
 depth - number of bitplanes in planetab  
 bytesperrow - total bytes per row in the source bitplanes.  
 This must be an even number. If you convert interleaved bitplanes, multiply by depth.  
 chunky - pointer to the destination chunky buffer  
 taglist - pointer to an array of TagItems

## TAGS

RND\_DestWidth (UWORD) - Total width of chunky array [pixels].  
 Default - equals to bytewidth \* 8.



You are explicitly allowed to use a destwidth that is smaller than bytewidth \* 8 pixels.

Important note:

If you specify this tag, you must still supply a chunky buffer of at least bytewidth \* 8 \* rows bytes.

#### NOTES

Starting with v39, you are not allowed to assume foreign BitMap structures being of a planar type. Before you grab a table of plane pointers out of an unknown BitMap structure and pass it to this function, you have to check for the presence of the BMF\_STANDARD flag.

#### RESULTS

none

#### SEE ALSO

Chunky2BitMapA(), graphics/gfx.h

## 1.41 render.library/QueryHistogram

#### NAME

QueryHistogram - query a histogram parameter.

#### SYNOPSIS

```
value = QueryHistogram(histogram,tag)
d0                a0                d0
```

```
ULONG QueryHistogram(APTR,Tag)
```

#### FUNCTION

Query one of a histogram's specifications via Tag parameter.

#### INPUTS

histogram - pointer to a histogram  
tag - Tag to be queried

#### TAGS

RND\_NumColors (ULONG) -  
the number of different colors inside the histogram

RND\_NumPixels (ULONG) -  
the number of pixels that have been added to the histogram

RND\_RMHandler (ULONG) -  
the histogram's render-memhandler

RND\_HSType (UWORD) -  
the histogram's type

---

## RESULTS

value - the queried parameter

SEE ALSO

CreateHistogramA()

### 1.42 render.library/RenderA

NAME

RenderA - render an array of RGB data to chunky bytes.  
Render - varargs stub for RenderA.

```

SYNOPSIS
    success = RenderA(rgba,width,height,chunky,palette,taglist)
    d0          a0  d0    d1    a1    a2    a3

    ULONG RenderA(ULONG *,UWORD,UWORD,UBYTE *,APTR,struct TagItem *)

    ULONG Render(ULONG *,UWORD,UWORD,UBYTE *,APTR,tag,...,TAG_DONE)

```

FUNCTION

Render an array of RGB data to an array of chunky bytes.

```

INPUTS
    rgb          - pointer to an array of RGB pixels
    width        - width to be converted [pixels]
    height       - height to be converted [rows]
    chunky       - pointer to the destination array
    palette      - pointer to a palette to be rendered to
    taglist      - pointer to an array of TagItems

```

TAGS

RND\_SourceWidth (UWORD) - Total width of the RGB array [pixels].  
Default - equals to the specified width.

RND\_DestWidth (UWORD) - Total width of the chunky array [pixels].  
Default - equals to the specified width.

RND\_ColorMode (ULONG) - Color mode that defines how to determine  
a pixel's actual color. Valid types:

COLORMODE_CLUT	- normal palette lookup
COLORMODE_HAM8	- HAM8 mode palette lookup
COLORMODE_HAM6	- HAM6 mode palette lookup

Default - COLORMODE\_CLUT.

RND\_DitherMode (UWORD) - Error diffusion mode. Valid types:

DITHERMODE_NONE	- no error diffusion
DITHERMODE_FS	- Floyd-Steinberg Does not handle a dither amount.

DITHERMODE\_RANDOM - Random dithering  
Handles RND\_DitherAmount.

DITHERMODE\_EDD - EDD dithering  
Does not handle a dither amount.

Default - DITHERMODE\_NONE.

RND\_ProgressHook (ULONG) - Pointer to a callback hook structure for progress display operations. Refer to render/renderhooks.h for further information.  
Default - NULL.

RND\_OffsetColorZero (UWORD) - First pen number to appear in the rendered chunky image. This offset will be added to the palette's indices. Default - 0.  
Note: RND\_PenTable overrides RND\_OffsetColorZero.

RND\_PenTable (ULONG) - Pointer to a table of 256 UBYTES for a secondary conversion of the pen numbers.  
Default - NULL.  
Note: RND\_PenTable overrides RND\_OffsetColorZero.

RND\_LineHook (ULONG) - Pointer to a callback hook structure for line-related operations during render. This hook is executed once before a line is rendered and once after it has been completed. Refer to render/renderhooks.h for further information.  
Default - NULL.

RND\_DitherAmount (UWORD) - Dither intensity (0-255).  
Only valid with certain dither modes (see above).  
Default - 128.

RND\_ScaleEngine (ULONG) - Pointer to a scaling-engine. This scaling-engine will be used to scale the source data stream before it is being rendered.  
Default - NULL.

Notes:

- The scaling engine's destination width and height specifications override the width and height arguments for RenderA().
- The scaling-engine must have been created with PIXFMT\_ORGB\_32.

RND\_MapEngine (ULONG) - Pointer to a mapping-engine. This can greatly improve render performance.  
Default - NULL.

Notes:

- Mapping-engines are not applied for rendering to HAM modes.
  - With dithering enabled, you better do not specify a mapping-engine which was created with a link to a histogram. Accurate conversion could not be guaranteed in this case.
-

## RESULTS

success - returncode to indicate whether the operation succeeded. You must at least check for `REND_SUCCESS`.

## NOTES

- This function may call the progress callback Hook with the `PMSGTYPE_LINES_RENDERED` message type.

## SEE ALSO

`MapRGBArrayA()`, `ConvertChunkyA()`, `CreateMapEngineA()`, `CreateScaleEngineA()`, `render/render.h`, `render/renderhooks.h`

## 1.43 render.library/RGBArrayDiversityA

## NAME

`RGBArrayDiversityA` - calculate RGB array adaptibility  
`RGBArrayDiversity` - varargs stub for `RGBArrayDiversityA`

## SYNOPSIS

```
diversity = RGBArrayDiversityA(rgbarray,width,height,taglist)
d0          a0          d0    d1    a1
```

```
LONG RGBArrayDiversityA(ULONG *,UWORD,UWORD,struct TagItem *)
```

```
LONG RGBArrayDiversity(ULONG *,UWORD,UWORD,tag,...,TAG_DONE)
```

## FUNCTION

This function calculates a RGB array's adaptibility to a palette or mapping-engine.

## INPUTS

`rgbarray` - pointer to an array of RGB data  
`width` - width of the RGB array [pixels]  
`height` - height of the RGB array [pixels]  
`taglist` - pointer to an array of tagitems

## TAGS

`RND_Palette` (ULONG) - pointer to a palette. Either this or the `RND_MapEngine` argument is obligatory!

`RND_MapEngine` (ULONG) - pointer to a mapping-engine. Either this or the `RND_Palette` argument is obligatory!

`RND_SourceWidth` (UWORD) - Total width of the RGB array [pixels].  
 Default - equals to the specified width.

`RND_Interleave` (ULONG) - number of pixels to skip horizontally.  
 Default - 0.

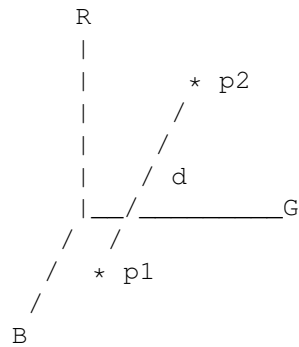
## RESULTS

`diversity` - an indicator for the RGB array's adaptibility to the given palette or mapping-engine. 0 indicates

perfect adaptibility, 195075 indicates the worst case.

#### IMPLEMENTATION

Every RGB in the source array is adapted to the specified palette or mapping-engine. Now assume p2 is the RGB of the palette entry that matches best with the requested RGB p1.



This function sums up the diversity  $d^2$  for every RGB in the source array, and divides it by the number of pixels processed:

$$D = \frac{1}{\text{pixels}} \cdot \sum_{k=1}^{k=\text{pixels}} d(k)^2$$

D is returned, so what you get is the average  $d^2$  for all RGB values that appear in the array.  $d^2$  is summed up instead of d because it is a better indicator for most purposes.

Best case - all pixels match perfectly:  $D(\text{min}) = 0$ .

Worst case - all pixels are white, and the only available color in the palette is black:  $D(\text{max}) = 255^2 + 255^2 + 255^2 = 195075$ .

#### EXAMPLE

D indicates a palette's applicability for the given RGB array. You can e.g. use it to decide whether error diffusion should be enabled or not.

```
dithermode = DITHERMODE_NONE;
```

```
D = RGBArrayDiversity(rgbarray, width, height,
    RND_Palette, destpalette, RND_Interleave, 4, TAG_DONE);
```

```
if (D > threshold)
{
    dithermode = DITHERMODE_FS;
}
```

```
Render(rgbarray, width, height, chunkyarray, destpalette,
    RND_DitherMode, dithermode, TAG_DONE);
```

The interleave factor increases speed drastically. Nonetheless,

`RGBArrayDiversity()` is yet very fast. You gain a lot of performance if only dithering can be avoided sometimes.

Applying `RGBArrayDiversity()` to a mapping-engine is even more sophisticated. Use this if a histogram is available for the RGB array.

```
me = CreateMapEngine(palette,RND_Histogram,histo,TAG_DONE);

D = RGBArrayDiversity(rgbarray, width, height,
    RND_MapEngine, me, RND_Interleave, 4, TAG_DONE);

if (D > threshold)
{
    Render(rgbarray, width, height, chunkyarray, destpalette,
        RND_DitherMode, DITHERMODE_FS, TAG_DONE);
}
else
{
    MapRGBArray(me, rgbarray, width, height, chunkyarray, NULL);
}
```

This example provides even higher performance. `MapRGBArrayA()` is faster than `RenderA()`, especially when the mapping-engine is coupled with a histogram. Note: That mapping-engine's histogram has to be of a `HSTYPE_..._TURBO` type with the same resolution as the palette.

SEE ALSO

`ChunkyArrayDiversityA()`

## 1.44 render.library/ScaleA

### NAME

`ScaleA` - scale an image.  
`Scale` - varargs stub for `ScaleA`.

### SYNOPSIS

```
success = ScaleA(engine,source,dest,taglist)
                a0      a1      a2      a3
```

```
ULONG ScaleA(APTR,APTR,APTR,struct TagItem *)
```

```
ULONG Scale(APTR,APTR,APTR,tag,...,TAG_DONE)
```

### FUNCTION

This function scales a source array of pixels to another array.

### INPUTS

`engine` - pointer to a scaling-engine created  
           with `CreateScaleEngineA()`  
`source` - pointer to source array of pixels  
`dest` - pointer to destination array

taglist - pointer to an array of TagItems

#### TAGS

RND\_SourceWidth (UWORD) - Total width of the source array [pixels].  
Default - equals to the source width the scaling-engine was created with.

RND\_DestWidth (UWORD) - Total width of the dest array [pixels].  
Default - equals to the destination width the scaling-engine was created with.

RND\_LineHook (ULONG) - Pointer to a callback hook structure for line-related operations during conversion. This hook is executed once before a line is processed and once after it has been completed. Refer to render/renderhooks.h for further information.  
Default - NULL.

#### RESULTS

success - CONV\_SUCCESS to indicate that the operation succeeded. Currently, the only reason for this function to fail is CONV\_CALLBACK\_ABORTED.

#### NOTES

- It is possible to downscale (shrink) an array over itself, i.e. source and destination buffers may be the same in this case.

#### SEE ALSO

CreateScaleEngineA(), RenderA(), ConvertChunkyA()

## 1.45 render.library/ScaleOrdinate

#### NAME

ScaleOrdinate - scale a single ordinate.

#### SYNOPSIS

```
scaled_ordinate = ScaleOrdinate(start,dest,ordinate)
d0                d0    d1    d2
```

```
UWORD ScaleOrdinate(UWORD,UWORD,UWORD)
```

#### FUNCTION

This function scales a single ordinate. The algorithm used here is identical to what scaling-engines are created with.

#### INPUTS

start	- original value (e.g. width or height) e.g. the original width of an image. This value usually corresponds to a start value with CreateScaleEngineA(). Must not be 0.
dest	- destination value (e.g. width or height) e.g. the scaled width of an image.

This value usually corresponds to a  
 dest value with `CreateScaleEngineA()`.  
 Must not be 0.  
 ordinate - a single ordinate (e.g. of a pixel).  
 Must be less than <start>.

#### RESULTS

scaled\_ordinate - the new ordinate (after scaling)

#### EXAMPLE

Assume you have a specific pair of coordinates that represent a particular pixel inside an image. You can use this function to determine the pixel's new coordinates after the image has been scaled:

```
new_pixel_x = ScaleOrdinate(picwidth,newwidth,pixel_x);
new_pixel_y = ScaleOrdinate(picheight,newheight,pixel_y);
```

#### SEE ALSO

`CreateScaleEngineA()`

## 1.46 render.library/SortPaletteA

#### NAME

SortPaletteA - sort a palette.  
 SortPalette - varargs stub for SortPaletteA.

#### SYNOPSIS

```
success = SortPaletteA(palette,mode,taglist)
d0          a0          d0    a1
```

```
ULONG SortPaletteA(APTR,ULONG,struct TagItem *)
```

```
ULONG SortPalette(APTR,ULONG,tag,...,TAG_DONE)
```

#### FUNCTION

Sorts a palette according to a sort mode. Some sort modes apply to palettes solely, some others additionally require a histogram.

#### INPUTS

palette - pointer to a palette created with `CreatePaletteA()`.

mode - sort mode. Currently defined are:

PALMODE\_BRIGHTNESS - sort the palette entries  
 by brightness.

PALMODE\_SATURATION - sort the palette entries  
 by their color intensity.

PALMODE\_POPULARITY - sort the palette entries  
 by the number of pixels that they



represent. You must specify the RND\_Histogram taglist argument.

PALMODE\_REPRESENTATION - sort the palette entries by the number of histogram entries that they represent. You must specify the RND\_Histogram taglist argument.

PALMODE\_SIGNIFICANCE - sort the palette entries by their optical significance for the human eye. Implementation is unknown to you and may change. You must supply the RND\_Histogram taglist argument.

PALMODE\_ASCENDING - by default, sort direction is descending, i.e. precedence is 'more-to-less' of the given effect. Combine with this flag to invert the sort direction.

taglist - pointer to an array of tagitems.

#### TAGS

RND\_Histogram (ULONG) - pointer to a histogram. This taglist argument is obligatory for some sort modes. (See above)

#### RESULTS

success - return value to indicate whether the operation succeeded. You must at least check for SORTP\_SUCCESS.

#### NOTES

SortPaletteA() can be extremely useful when a palette is to be allocated from a ColorMap via ObtainBestPenA(). After sorting with PALMODE\_SIGNIFICANCE, important colors will be treated first. This increases probability for ObtainBestPenA() to allocate new pens for the most significant entries in the palette.

## 1.47 render.library/TintRGBArrayA

#### NAME

TintRGBArrayA - tint an RGB array.  
TintRGBArray - varargs stub for TintRGBArrayA.

#### SYNOPSIS

```
TintRGBArrayA(sourcearray,width,height,RGB,ratio,destarray,tags)
               a0          d0      d1      d2  d3      a1      a2
```

```
void TintRGBArrayA(ULONG *,UWORD,UWORD,ULONG,UWORD,ULONG *,
                  struct TagItem *)
```

```
void TintRGBArray(ULONG *,UWORD,UWORD,ULONG,UWORD,ULONG *,
                  tag,...,TAG_DONE)
```

## FUNCTION

This function mixes an RGB array with the specified RGB value and a mixing ratio. The result is written to another array, which may be the same as the source array.

## INPUTS

sourcearray	- pointer to an ARGB array
width	- width [pixels]
height	- height [rows]
RGB	- tint RGB
ratio	- tint ratio (0...255)
destarray	- pointer to an ARGB array
taglist	- pointer to an array of TagItems

## TAGS

RND\_SourceWidth (UWORD) - Total width of the source array [pixels].  
Default - equals to the specified width.

RND\_DestWidth (UWORD) - Total width of the dest array [pixels].  
Default - equals to the specified width.

## RESULTS

none