

render

COLLABORATORS

	<i>TITLE :</i> render		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY		January 18, 2023	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1	render	1
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	5
1.6	render.library/AllocRenderMem	6
1.7	render.library/AllocRenderVec	6
1.8	render.library/BestPen	7
1.9	render.library/Chunky2BitMapA	8
1.10	render.library/Chunky2RGBA	9
1.11	render.library/ConvertChunkyA	11
1.12	render.library/CreateHistogramA	12
1.13	render.library/CountRGB	13
1.14	render.library/CreatePaletteA	13
1.15	render.library/CreatePenTableA	14
1.16	render.library/CreateRMHandlerA	15
1.17	render.library/CreateScaleEngineA	17
1.18	render.library/DeleteHistogram	18
1.19	render.library/DeleteRMHandler	18
1.20	render.library/DeletePalette	19
1.21	render.library/DeleteScaleEngine	19
1.22	render.library/ExportPaletteA	20
1.23	render.library/ExtractPaletteA	21
1.24	render.library/FlushPalette	22
1.25	render.library/FreeRenderMem	23
1.26	render.library/FreeRenderVec	23
1.27	render.library/ImportPaletteA	24
1.28	render.library/Planar2ChunkyA	25
1.29	render.library/QueryHistogram	26

1.30	render.library/RenderA	27
1.31	render.library/ScaleA	29
1.32	render.library/ScaleOrdinate	29
1.33	render.library/SortPaletteA	30

Chapter 1

render

1.1 render.doc

AddChunkyImageA ()

AddHistogramA ()

AddRGB ()

AddRGBImageA ()

AllocRenderMem ()

AllocRenderVec ()

BestPen ()

Chunky2BitMapA ()

Chunky2RGBA ()

ConvertChunkyA ()

CountRGB ()

CreateHistogramA ()

CreatePaletteA ()

CreatePenTableA ()

CreateRMHandlerA ()

CreateScaleEngineA ()

DeleteHistogram ()

DeletePalette ()

DeleteRMHandler ()

```
DeleteScaleEngine()  
ExportPaletteA()  
ExtractPaletteA()  
FlushPalette()  
FreeRenderMem()  
FreeRenderVec()  
ImportPaletteA()  
Planar2ChunkyA()  
QueryHistogram()  
RenderA()  
ScaleA()  
ScaleOrdinate()  
SortPaletteA()
```

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 AddChunkyImageA(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 gets 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 object
 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 AddHistogramA(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

desthistogram - pointer to destination histogram
sourcehistogram - pointer to source histogram
taglist - pointer to an array of TagItems

TAGS

None defined. Pass 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.

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 single RGB value plus the number of its representations 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, keeping 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()
,
AllocRenderMem()
,
CreateRMHandlerA()
```

1.8 render.library/BestPen

NAME

BestPen - find the best matching pen.

SYNOPSIS

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

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

FUNCTION

Calculate 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.)

1.9 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 an additional 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()

1.10 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

chunky - pointer to an array of chunky bytes
width - width to be converted [pixels]
height - height to be converted [rows]
rgb - pointer to RGB destination buffer
palette - pointer to a palette object 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_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.

RESULTS

result - 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

1.11 render.library/ConvertChunkyA

NAME

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

SYNOPSIS

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

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

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

FUNCTION

This function converts a source array of chunky bytes to another array, and adapts it to a new palette. Optionally, a secondary pen conversion is performed.

INPUTS

source - pointer to the source array of chunky data
 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 width specified.

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

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

RESULTS

none

SEE ALSO

```
CreatePenTableA()
,
CreatePaletteA()
```

1.12 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
```


1.13 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 particular 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.

1.14 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 are in the range from 12 to 18 bit:

```
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
```

1.15 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.

This function is equivalent to

```
ConvertChunkyA()
```

```
, except
```

for that it does not actually convert the chunky image, but instead creates the conversion table for that palette adaption.

INPUTS

```
chunky      - pointer to an array of chunky bytes
oldpalette  - pointer to the original palette
width       - width to be converted [pixels]
height      - height to be converted [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 image.

SEE ALSO

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

1.16 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, and many other operations. 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

1.17 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, this scaling-engine is highly optimized for these 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.
```

RESULTS

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

SEE ALSO

```
DeleteScaleEngine()
,
ScaleA()
```

1.18 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

RESULTS

none

1.19 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()
```

1.20 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, such as created
with
CreatePaletteA()

RESULTS

none

SEE ALSO

CreatePaletteA()

,
FlushPalette()

1.21 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()  
,  
ScaleA()
```

1.22 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()

```

1.23 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. This results in a color lookup table (also called 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 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

result - 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
```

1.24 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()

1.25 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 previously allocated with AllocRenderMem().

INPUTS

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

RESULTS

NONE

SEE ALSO

AllocRenderMem()
,
DeleteRMHandler()

1.26 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 previously allocated with

```
AllocRenderVec()
```

```
.
```

INPUTS

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

RESULTS

```
NONE
```

SEE ALSO

```
AllocRenderVec()
```

```
,
```

```
DeleteRMHandler()
```

1.27 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 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 the source color table
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 import. Valid types:
```

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

Default - PALFMT_RGB8.

RND_FirstColor (ULONG) - first destination color entry to import to. Default - 0.

RND_EHBPalette (ULONG) - tag to indicate whether the palette 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
```

1.28 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

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
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 planepointers 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

1.29 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()

1.30 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

```
rgba      - 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.

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 color index 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 a line 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.

RESULTS

result - 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

`render/render.h`, `render/renderhooks.h`

1.31 render.library/ScaleA

NAME

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

SYNOPSIS

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

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

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

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.

RESULTS

none

SEE ALSO

CreateScaleEngineA()

1.32 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.

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);
```

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)

SEE ALSO

`CreateScaleEngineA()`

1.33 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.

SEE ALSO

render/render.h
