layers

COLLABORATORS

	TITLE : layers	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		March 28, 2025	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

layers

1.1 layers.doc

```
 BeginUpdate()              MoveLayer()
 BehindLayer()              MoveLayerInFrontOf()
 CreateBehindLayer()        NewLayerInfo()
 CreateUpfrontLayer()       ScrollLayer()
 DeleteLayer()              SizeLayer()
 DisposeLayerInfo()         SwapBitsRastPortClipRect()
 EndUpdate()                ThinLayerInfo()
 FattenLayerInfo()          UnlockLayer()
 InitLayers()               UnlockLayerInfo()
 InstallClipRegion()        UnlockLayers()
 LockLayer()                UpfrontLayer()
 LockLayerInfo()            WhichLayer()
 LockLayers()
```

1.2 layers.library/BeginUpdate

```
NAME
    BeginUpdate -- Prepare to repair damaged layer.

SYNOPSIS
    result = BeginUpdate( l )
    d0                    a0

    BOOLEAN result;
    struct Layer *l;

FUNCTION
    Convert damage list to ClipRect list and swap in for
    programmer to redraw through. This routine simulates
    the ROM library environment. The idea is to only render in the
    "damaged" areas, saving time over redrawing all of the layer.
    The layer is locked against changes made by the layer library.

INPUTS
    l - pointer to a layer
```

```
RESULTS
    result - TRUE if damage list converted to ClipRect list sucessfully.
             FALSE if list conversion aborted. (probably out of memory)

BUGS
    If BeginUpdate returns FALSE, programmer must abort the attempt to
    refresh this layer and instead call EndUpdate( l, FALSE ) to restore
    original ClipRect and damage list.

SEE ALSO
    EndUpdate, graphics/layers.h, graphics/clip.h
```

1.3 layers.library/BehindLayer

```
NAME
    BehindLayer -- Put layer behind other layers.

SYNOPSIS
    result = BehindLayer( dummy, l )
    d0                    a0     a1

    BOOLEAN result;
    LONG dummy;
    struct Layer *l;

FUNCTION
    Move this layer to the most behind position swapping bits
    in and out of the display with other layers.
    If other layers are REFRESH then collect their damage lists and
    set the LAYERREFRESH bit in the Flags fields of those layers that
    may be revealed.  If this layer is a backdrop layer then
    put this layer behind all other backdrop layers.
    If this layer is NOT a backdrop layer then put in front of the
    top backdrop layer and behind all other layers.

    Note: this operation may generate refresh events in other layers
       associated with this layer's Layer_Info structure.

INPUTS
    dummy - unused
    l - pointer to a layer

RESULTS
    result - TRUE    if operation successful
             FALSE   if operation unsuccessful (probably out of memory)

BUGS

SEE ALSO
    graphics/layers.h, graphics/clip.h
```

1.4 layers.library/CreateBehindLayer

```
NAME
    CreateBehindLayer -- Create a new layer behind all existing layers.

SYNOPSIS
    result = CreateBehindLayer(li,bm,x0,y0,x1,y1,flags [,bm2])
    d0                         a0 a1 d0 d1 d2 d3   d4  [ a2 ]

    struct Layer *result;
    struct Layer_Info *li;
    struct BitMap *bm;
    LONG x0,y0,x1,y1;
    LONG flags;
    struct BitMap *bm2;

FUNCTION
    Create a new Layer of position and size (x0,y0)->(x1,y1)
    Make this layer of type found in flags.
    If SuperBitMap, use bm2 as pointer to real SuperBitMap,
    and copy contents of Superbitmap into display layer.
    If this layer is a backdrop layer then place it behind all
    other layers including other backdrop layers. If this is
    not a backdrop layer then place it behind all nonbackdrop
    layers.

    Note: when using SUPERBITMAP, you should also set LAYERSMART flag.

INPUTS
    li - pointer to LayerInfo structure
    bm - pointer to common BitMap used by all Layers
    x0,y0 - upper left hand corner of layer
    x1,y1 - lower right hand corner of layer
    flags - various types of layers supported as bit sets.
            (for bit definitions, see graphics/layers.h )
    bm2 - pointer to optional Super BitMap

RESULTS
    result - pointer to Layer structure if successful
             NULL if not successful

BUGS

SEE ALSO
    DeleteLayer, graphics/layers.h, graphics/clip.h, graphics/gfx.h
```

1.5 layers.library/CreateUpfrontLayer

```
NAME
    CreateUpfrontLayer -- Create a new layer on top of existing layers.

SYNOPSIS
    result = CreateUpfrontLayer(li,bm,x0,y0,x1,y1,flags [,bm2])
    d0                          a0 a1 d0 d1 d2 d3   d4  [ a2 ]
```

```
    struct Layer *result;
    struct Layer_Info *li;
    struct BitMap *bm;
    LONG x0,y0,x1,y1;
    LONG flags;
    struct BitMap *bm2;

FUNCTION
    Create a new Layer of position and size (x0,y0)->(x1,y1)
    and place it on top of all other layers.
    Make this layer of type found in flags
    if SuperBitMap, use bm2 as pointer to real SuperBitMap.
    and copy contents of Superbitmap into display layer.

    Note: when using SUPERBITMAP, you should also set LAYERSMART flag.

INPUTS
    li - pointer to LayerInfo structure
    bm - pointer to common BitMap used by all Layers
    x0,y0 - upper left hand corner of layer
    x1,y1 - lower right hand corner of layer
    flags - various types of layers supported as bit sets.
    bm2 - pointer to optional Super BitMap

RESULTS
    result - pointer to Layer structure if successful
             NULL if not successful

BUGS

SEE ALSO
    DeleteLayer, graphics/layers.h, graphics/clip.h, graphics/gfx.h
```

1.6 layers.library/DeleteLayer

```
NAME
    DeleteLayer -- delete layer from layer list.

SYNOPSIS
    result = DeleteLayer( dummy, l )
    d0                    a0,    a1

    BOOLEAN result;
    LONG dummy;
    struct Layer *l;

FUNCTION
    Remove this layer from the list of layers.  Release memory
    associated with it.  Restore other layers that may have been
    obscured by it.  Trigger refresh in those that may need it.
    If this is a superbitmap layer make sure SuperBitMap is current.
    The SuperBitMap is not removed from the system but is available
    for program use even though the rest of the layer information has
    been deallocated.
```

```
INPUTS
    dummy - unused
    l - pointer to a layer

RESULTS
    result - TRUE if this layer successfully deleted from the system
             FALSE if layer not deleted. (probably out of memory )

BUGS

SEE ALSO
    graphics/layers.h, graphics/clip.h
```

1.7 layers.library/DisposeLayerInfo

```
NAME
    DisposeLayerInfo -- Return all memory for LayerInfo to memory pool

SYNOPSIS
    DisposeLayerInfo(li)
                     a0

    struct Layer_Info *li;

FUNCTION
    return LayerInfo and any other memory attached to this LayerInfo
    to memory allocator.

    Note: if you wish to delete the layers associated with this Layer_Info
        structure, remember to call DeleteLayer() for each of the layers
        before calling DisposeLayerInfo().

INPUTS
    li - pointer to LayerInfo structure

EXAMPLE
    -- delete the layers associated this Layer_Info structure --

    DeleteLayer(li,simple_layer);
    DeleteLayer(li,smart_layer);

    - see documentation on DeleteLayer about deleting SuperBitMap layers -
    my_super_bitmap_ptr = super_layer->SuperBitMap;
    DeleteLayer(li,super_layer);

    -- now dispose of the Layer_Info structure itself --
    DisposeLayerInfo(li);

BUGS

SEE ALSO
    DeleteLayer, graphics/layers.h
```

1.8 layers.library/EndUpdate

```
NAME
    EndUpdate -- remove damage list and restore state of layer to normal.

SYNOPSIS
    EndUpdate( l, flag )
               a0  d0

    struct Layer *l;
    USHORT flag;

FUNCTION
    After the programmer has redrawn his picture he calls this
    routine to restore the ClipRects to point to his standard
    layer tiling. The layer is then unlocked for access by the
    layer library.

    Note: use flag = FALSE if you are only making a partial update.
        You may use the other region functions (graphics functions such as
        OrRectRegion, AndRectRegion, and XorRectRegion ) to clip adjust
        the DamageList to reflect a partial update.

INPUTS
    l - pointer to a layer
    flag - use TRUE if update was completed. The damage list is cleared.
           use FALSE if update not complete. The damage list is retained.

EXAMPLE
    -- begin update for first part of two-part refresh --
    BeginUpdate(my_layer);

    -- do some refresh, but not all --
    my_partial_refresh_routine(my_layer);

    -- end update, false (not completely done refreshing yet) --
    EndUpdate(my_layer, FALSE);

    -- begin update for last part of refresh --
    BeginUpdate(my_layer);

    -- do rest of refresh --
    my_complete_refresh_routine(my_layer);

    -- end update, true (completely done refreshing now) --
    EndUpdate(my_layer, TRUE);

BUGS

SEE ALSO
    BeginUpdate, graphics/layers.h, graphics/clip.h
```

1.9 layers.library/FattenLayerInfo

```
NAME
    FattenLayerInfo -- convert 1.0 LayerInfo to 1.1 LayerInfo
    OBSOLETE OBSOLETE OBSOLETE OBSOLETE OBSOLETE

SYNOPSIS
    OBSOLETE OBSOLETE OBSOLETE OBSOLETE OBSOLETE
    FattenLayerInfo(li)
                    a0

    struct Layer_Info *li;
    OBSOLETE OBSOLETE OBSOLETE OBSOLETE OBSOLETE

FUNCTION
    V1.1 software and any later releases need to have more info in the
    Layer_Info structure. To do this in a 1.0 supportable manner requires
    allocation and deallocation of the memory whenever most
    layer library functions are called.  To prevent unnecessary
    allocation/deallocation FattenLayerInfo will preallocate the
    necessary data structures and fake out the layer library into
    thinking it has a LayerInfo gotten from NewLayerInfo.
    NewLayerInfo is the approved method for getting this structure.
    When a program needs to give up the LayerInfo structure it
    must call ThinLayerInfo before freeing the memory. ThinLayerInfo
    is not necessary if New/DisposeLayerInfo are used however.

INPUTS
    li - pointer to LayerInfo structure

BUGS

SEE ALSO
    NewLayerInfo, ThinLayerInfo, DisposeLayerInfo, graphics/layers.h
```

1.10 layers.library/InitLayers

```
NAME
    InitLayers -- Initialize Layer_Info structure
    OBSOLETE OBSOLETE OBSOLETE OBSOLETE OBSOLETE

SYNOPSIS
    OBSOLETE OBSOLETE OBSOLETE OBSOLETE OBSOLETE
    InitLayers(li)
               a0

    struct Layer_Info *li;
    OBSOLETE OBSOLETE OBSOLETE OBSOLETE OBSOLETE

FUNCTION
    Initialize Layer_Info structure in preparation to use
    other layer operations on this list of layers.
    Make the Layers unlocked (open), available to layer operations.

INPUTS
    li - pointer to LayerInfo structure
```

```
BUGS

SEE ALSO
    NewLayerInfo, DisposeLayerInfo, graphics/layers.h
```

1.11 layers.library/InstallClipRegion

```
NAME
    InstallClipRegion -- Install clip region in layer

SYNOPSIS
    oldclipregion = InstallClipRegion( l,  region )
    d0                                 a0  a1

    struct Region *oldclipregion;
    struct Layer *l;
    struct Region *region;

FUNCTION
    Installs a transparent Clip region in the layer. All
    subsequent graphics calls will be clipped to this region.
    You MUST remember to call InstallClipRegion(l,NULL) before
    calling DeleteLayer(l) or the Intuition function CloseWindow()
    if you have installed a non-NULL ClipRegion in l.

INPUTS
    l - pointer to a layer
    region - pointer to a region

RESULTS
    oldclipregion - The pointer to the previous ClipRegion that
        was installed. Returns NULL if no previous ClipRegion installed.

        Note: If the system runs out of memory while computing the
            resulting ClipRects the LAYERS_CLIPRECTS_LOST bit will
            be set in l->Flags.

BUGS
    If the system runs out of memory during normal layer operations,
    the ClipRect list may get swept away and not restored.
    As soon as there is enough memory and the layer library
    gets called again the ClipRect list will be rebuilt.

SEE ALSO
    BeginUpdate EndUpdate,
    graphics/layers.h, graphics/clip.h, graphics/regions.h
```

1.12 layers.library/LockLayer

```
NAME
    LockLayer -- Lock layer to make changes to ClipRects.
```

```
SYNOPSIS
    LockLayer( dummy, l )
               a0     a1

    LONG dummy;
    struct Layer *l;

FUNCTION
    Make this layer unavailable for other tasks to use.
    If another task is already using this layer then wait for
    it to complete and then reserve the layer for your own use.
    (this function does the same thing as graphics.library/LockLayerRom)

    Note: if you wish to lock MORE THAN ONE layer at a time, you
        must call LockLayerInfo() before locking those layers and
        then call UnlockLayerInfo() when you have finished. This
        is to prevent system "deadlocks".

    Further Note: while you hold the lock on a layer, Intuition will block
        on operations such as windowsizing, dragging, menus, and depth
        arranging windows in this layer's screen.  It is recommended that
        YOU do not make Intuition function calls while the layer is locked.

INPUTS
    dummy - unused
    l - pointer to a layer

BUGS

SEE ALSO
    UnlockLayer, LockLayerInfo, UnlockLayerInfo,
    graphics.library/LockLayerRom, graphics/layers.h, graphics/clip.h
```

1.13 layers.library/LockLayerInfo

```
NAME
    LockLayerInfo -- Lock the LayerInfo structure.

SYNOPSIS
    LockLayerInfo( li )
                   a0

    struct Layer_Info *li;

FUNCTION
    Before doing an operation that requires the LayerInfo
    structure, make sure that no other task is also using the
    LayerInfo structure.  LockLayerInfo() returns when the
    LayerInfo belongs to this task.  There should be
    an UnlockLayerInfo for every LockLayerInfo.

    Note: All layer routines presently LockLayerInfo() when they
    start up and UnlockLayerInfo() as they exit.  Programmers
    will need to use these Lock/Unlock routines if they wish
```

```
    to do something with the LayerStructure that is not
    supported by the layer library.

INPUTS
    li - pointer to Layer_Info structure

BUGS

SEE ALSO
    UnlockLayerInfo, graphics/layers.h
```

1.14 layers.library/LockLayers

```
NAME
    LockLayers -- lock all layers from graphics output.

SYNOPSIS
    LockLayers( li )
                a0

    struct Layer_Info *li;

FUNCTION
    First calls LockLayerInfo()
    Make all layers in this layer list locked.

INPUTS
    li - pointer to Layer_Info structure

BUGS

SEE ALSO
    LockLayer, LockLayerInfo, graphics/layers.h
```

1.15 layers.library/MoveLayer

```
NAME
    MoveLayer -- Move layer to new position in BitMap.

SYNOPSIS
    result = MoveLayer( dummy, l, dx, dy )
    d0                  a0     a1 d0  d1

    BOOLEAN result;
    LONG dummmy;
    struct Layer *l;
    LONG dx,dy;

FUNCTION
    Move this layer to new position in shared BitMap.
    If any refresh layers become revealed, collect damage and
    set REFRESH bit in layer Flags.
```

```
INPUTS
    dummy - unused
    l - pointer to a nonbackdrop layer
    dx - delta to add to current x position
    dy - delta to add to current y position

RETURNS
    result - TRUE if operation successful
             FALSE if failed (out of memory)

BUGS
    May not handle (dx,dy) which attempts to move the layer ouside the
    layer's RastPort->BitMap bounds .

SEE ALSO
    graphics/layers.h, graphics/clip.h
```

1.16 layers.library/MoveLayerInFrontOf

```
NAME
    MoveLayerInFrontOf-- Put layer in front of another layer.

SYNOPSIS
    result = MoveLayerInFrontOf( layertomove, targetlayer )
                                 a0           a1
    BOOLEAN result;
    struct Layer *layertomove;
    struct Layer *targetlayer;

FUNCTION
    Move this layer in front of target layer, swapping bits
    in and out of the display with other layers.
    If this is a refresh layer then collect damage list and
    set the LAYERREFRESH bit in layer->Flags if redraw required.

    Note: this operation may generate refresh events in other layers
       associated with this layer's Layer_Info structure.

INPUTS
    layertomove - pointer to layer which should be moved
    targetlayer - pointer to target layer in front of which to move layer

RESULTS
    result = TRUE    if operation successful
             FALSE   if operation unsuccessful (probably out of memory)

BUGS

SEE ALSO
    graphics/layers.h
```

1.17 layers.library/NewLayerInfo

```
NAME
    NewLayerInfo -- Allocate and Initialize full Layer_Info structure.

SYNOPSIS
    result = NewLayerInfo()
    d0

    struct Layer_Info *result;

FUNCTION
    Allocate memory required for full Layer_Info structure.
    Initialize Layer_Info structure in preparation to use
    other layer operations on this list of layers.
    Make the Layer_Info unlocked (open).

INPUTS
    None

RESULT
    result- pointer to Layer_Info structure if successful
            NULL if not enough memory

BUGS

SEE ALSO
    graphics/layers.h
```

1.18 layers.library/ScrollLayer

```
NAME
    ScrollLayer -- Scroll around in a superbitmap, translate coordinates
                   in non-superbitmap layer.

SYNOPSIS
    ScrollLayer( dummy, l, dx, dy )
                 a0     a1 d0  d1

    LONG dummy;
    struct Layer *l;
    LONG dx,dy;

FUNCTION
    For a SuperBitMap Layer:
    Update the SuperBitMap from the layer display, then copy bits
    between Layer and SuperBitMap to reposition layer over different
    portion of SuperBitMap.
    For nonSuperBitMap layers, all (x,y) pairs are adjusted by
    the scroll(x,y) value in the layer.  To cause (0,0) to actually
    be drawn at (3,10) use ScrollLayer(-3,-10). This can be useful
    along with InstallClipRegion to simulate Intuition GZZWindows
    without the overhead of an extra layer.
```

```
INPUTS
    dummy - unused
    l - pointer to a layer
    dx - delta to add to current x scroll value
    dy - delta to add to current y scroll value

BUGS
    May not handle (dx,dy) which attempts to move the layer ouside the
    layer's SuperBitMap bounds.

SEE ALSO
    graphics/layers.h
```

1.19 layers.library/SizeLayer

```
NAME
    SizeLayer -- Change the size of this nonbackdrop layer.

SYNOPSIS
    result = SizeLayer( dummy, l, dx, dy )
    d0                  a0     a1 d0  d1

    BOOLEAN result;
    LONG dummy;
    struct Layer *l;
    LONG dx, dy;

FUNCTION
    Change the size of this layer by (dx,dy). The lower right hand
    corner is extended to make room for the larger layer.
    If there is SuperBitMap for this layer then copy pixels into
    or out of the layer depending on whether the layer increases or
    decreases in size.  Collect damage list for those layers that may
    need to be refreshed if damage occurred.

INPUTS
    dummy - unused
    l - pointer to a nonbackdrop layer
    dx - delta to add to current x size
    dy - delta to add to current y size

RESULTS
    result - TRUE if operation successful
             FALSE if failed (out of memory)

BUGS

SEE ALSO
    graphics/layers.h, graphics/clip.h
```

1.20 layers.library/SwapBitsRastPortClipRect

```
NAME
    SwapBitsRastPortClipRect -- Swap bits between common bitmap
                                and obscured ClipRect

SYNOPSIS
    SwapBitsRastPortClipRect( rp, cr )
                             a0  a1

    struct RastPort *rp;
    struct ClipRect *cr;

FUNCTION
    Support routine useful for those that need to do some
    operations not done by the layer library.  Allows programmer
    to swap the contents of a small BitMap with a subsection of
    the display. This is accomplished without using extra memory.
    The bits in the display RastPort are exchanged with the
    bits in the ClipRect's BitMap.

    Note: the ClipRect structures which the layer library allocates are
    actually a little bigger than those described in the graphics/clip.h
    include file.  So be warned that it is not a good idea to have
    instances of cliprects in your code.

INPUTS
    rp - pointer to rastport
    cr - pointer to cliprect to swap bits with

BUGS

SEE ALSO
    graphics/clip.h, graphics/rastport.h, graphics/clip.h
```

1.21 layers.library/ThinLayerInfo

```
NAME
    ThinLayerInfo -- convert 1.1 LayerInfo to 1.0 LayerInfo.
    OBSOLETE OBSOLETE OBSOLETE OBSOLETE OBSOLETE

SYNOPSIS
    OBSOLETE OBSOLETE OBSOLETE OBSOLETE OBSOLETE
    ThinLayerInfo(li)
                  a0

    struct Layer_Info *li;
    OBSOLETE OBSOLETE OBSOLETE OBSOLETE OBSOLETE

FUNCTION
    return the extra memory needed that was allocated with
    FattenLayerInfo. This is must be done prior to freeing
    the Layer_Info structure itself. V1.1 software should be
    using DisposeLayerInfo.

INPUTS
```

```
    li - pointer to LayerInfo structure

BUGS

SEE ALSO
    DisposeLayerInfo, FattenLayerInfo, graphics/layers.h
```

1.22 layers.library/UnlockLayer

```
NAME
    UnlockLayer -- Unlock layer and allow graphics routines to use it.

SYNOPSIS
    UnlockLayer( l )
                 a0

    struct Layer *l;

FUNCTION
    When finished changing the ClipRects or whatever you were
    doing with this layer you must call UnlockLayer() to allow
    other tasks to proceed with graphic output to the layer.

INPUTS
    l - pointer to a layer

BUGS

SEE ALSO
    graphics/layers.h, graphics/clip.h
```

1.23 layers.library/UnlockLayerInfo

```
NAME
    UnlockLayerInfo -- Unlock the LayerInfo structure.

SYNOPSIS
    UnlockLayerInfo( li )
                     a0

    struct Layer_Info *li;

FUNCTION
    After the operation is complete that required a LockLayerInfo,
    unlock the LayerInfo structure so that  other tasks may
    affect the layers.

INPUTS
    li - pointer to the Layer_Info structure

    BUGS
```

```
    SEE ALSO
    LockLayerInfo, graphics/layers.h
```

1.24 layers.library/UnlockLayers

```
NAME
    UnlockLayers -- Unlock all layers from graphics output.
            Restart graphics output to layers that have been waiting

SYNOPSIS
    UnlockLayers( li )
                  a0

    struct Layer_Info *li;

FUNCTION
    Make all layers in this layer list unlocked.
    Then call UnlockLayerInfo

INPUTS
    li - pointer to the Layer_Info structure

BUGS

SEE ALSO
    LockLayers, UnlockLayer, graphics/layers.h
```

1.25 layers.library/UpfrontLayer

```
NAME
    UpfrontLayer -- Put layer in front of all other layers.

SYNOPSIS
    result = UpfrontLayer( dummy, l )
    do                     a0  a1

    BOOLEAN result;
    LONG dummy;
    struct Layer *l;

FUNCTION
    Move this layer to the most upfront position swapping bits
    in and out of the display with other layers.
    If this is a refresh layer then collect damage list and
    set the LAYERREFRESH bit in layer->Flags if redraw required.
    By clearing the BACKDROP bit in the layers Flags you may
    bring a Backdrop layer up to the front of all other layers.

    Note: this operation may generate refresh events in other layers
       associated with this layer's Layer_Info structure.

INPUTS
```

```
    dummy - unused
    l - pointer to a nonbackdrop layer

RESULTS
    result - TRUE   if operation successful
             FALSE  if operation unsuccessful (probably out of memory)

BUGS

SEE ALSO
    graphics/layers.h
```

1.26 layers.library/WhichLayer

```
NAME
    WhichLayer -- Which Layer is this point in?

SYNOPSIS
    layer = WhichLayer( li, x, y )
    d0                  a0  d0 d1

FUNCTION
    Starting at the topmost layer check to see if this point (x,y)
        occurs in this layer.  If it does return the pointer to this
        layer. Return NULL if there is no layer at this point.

INPUTS
    li  = pointer to LayerInfo structure
    (x,y)   = coordinate in the BitMap

RESULTS
    layer - pointer to the topmost layer that this point is in
            NULL if this point is not in a layer

SEE ALSO
    graphics/layers.h
```