SeedCFill Calculate a mask for use in CopyMask or CopyBits

#include <Quickdraw.h> **Color Quickdraw**

```
void        SeedCFill( srcMap, destMap, srcRect,destRect, seedH, seedV,
            matchProc, matchData );
BitMap      *srcMap ;       addr in a BitMap or PixMap to start calculating
BitMap      *destMap;       addr within a BitMap or PixMap to store 1s and 0s
Rect        *srcRect ;      position and size of source image
Rect        *destRect ;     position and size of destination image
short       seedH ;         horizontal offset, in pixels, to start pouring
short       seedV ;         vertical offset, in pixels, to start pouring
ColorSearchProcPtr
            matchProc ;     pointer to match seed RGB values
long        matchData ;     value passed by application
```

SeedCFill examines a portion of a source bitMap or pixMap and fills a portion of a destination bitMap or pixMap with 1s where paint can flow. It finds an enclosed area surrounding a specified point in the source (seed), and floods that area in the destination with 1s (RGB value equals that of seed). Use this function as one step in implementing a "paint bucket" tool.

srcMap is the address (srcBits) of a rectangle *inside* a bitMap or pixMap data area. **SeedCFill** examines this rectangle as it floods portions of the destination bitMap.

destMap is the address (destBits) of a rectangle *inside* a bitMap or pixMap data area. **SeedCFill** fills all or part of this rectangle with 1s.

srcRect and . . .
destRect are the rectangles within the BitMap or PixMap into which *srcMap* and *destMap*, respectively point; i.e., the function will add this value to its current address pointer to move "down one line" in the bitMap.

seedH and...
seedV identify the point to start flooding.

matchProc returns 0s for RGB values to be filled -- returns 1s when the values should not be filled.

matchData returns the value assigned.

Returns: none

Notes: The default setting for **SeedCFill** allows paint to flow from the seed position to all positions that touch it and whose RGB value equals that of the seed. Setting matchProc and matchData to zero calls the default mode.

Use **SeedCFill** to flood an area of a destination bitMap pixMap with paint (i.e., 1s) in the exact color of the source. The flooded area will match the inside and boundary of a section of the source which is enclosed by RGB pixels.

To flood with some other pattern requires intermediate steps of using

SeedCFill to create a mask, creating a bitmap filled with the desired pattern, and using **CopyMask** and **CopyBits** to move the data around. See the example, below.
When it makes the mask, **SeedCFill** uses **CopyBits** to convert the source location to a one-bit mask, installs a default searchProc into the gDevice that obtains a zero in all cases where the RGB value equals the RGB value of the seed and a one in every other case. You can customize your application by having it specify a matchProc that returns a 0 for RGB values you want filled, and a 1, otherwise. Then, when you call matchProc, the GDRefCon field of the current gDevice will have a point to a record that looks like:

Field	Value
red	short
green	short
blue	short
matchData	long

Red, green, and blue are the RGB values of the seed pixel and matchData is the value you gave **SeedCFill** as a parameter. This lets you, for example, pass a handle to a color table where you want all the entries to be filled and use matchProc to see of the specified RGB and any of the colors in the table are identical.

SeedCFill will not stretch or shrink an image, so your source and destination rectangles have to be the same size.

Calls to **SeedCFill** are not recorded in a Picture definition (See **OpenPicture**) and are not clipped to the current port.
The matchProc field is a procedure pointer which is delcared in **QuickDraw.h** as:

In the example below, there is a call to **SeedFill** and **SeedCFill**. Note that **SeedFill** takes pointers to fields of BitMaps and **SeedCFill** takes pointers to BitMaps.

Example

```
#include <Quickdraw.h>

#define WINDOW_ID 128 // this can be any window kind, etc.

// prototypes
void NewBitMap( BitMap *, Rect * );

main()
{
    Rect     r,winRect;
    BitMap realBits;
    BitMap srcBits, destBits, patBits;
    GrafPtr     oldPort;
    WindowPtr theWindow;

    InitGraf(&thePort);
```

```
    InitFonts();
    FlushEvents( everyEvent, 0 );
    InitWindows();
    InitMenus();
    TEInit();
    InitDialogs(0L);
    InitCursor();
    MaxApplZone();

    GetPort(&oldPort);
    theWindow = GetNewWindow(WINDOW_ID, 0L, (WindowPtr)-1L);
    SetPort(theWindow);

    winRect = theWindow->portRect;
    realBits = theWindow->portBits;     // the visible screen

    NewBitMap(&srcBits, &winRect);   // to hold copy of the source
    NewBitMap(&destBits, &winRect);  // receives the flooded mask
    NewBitMap(&patBits, &winRect);   // patter to filter via CopyMask

    SetPortBits(&patBits);           // create the pattern
    FillRect(&winRect, gray);

    SetPortBits(&realBits);
    EraseRect(&winRect);

    SetRect(&r, 20, 20, 100, 100);   // draw concetric rectangles
    FrameRect(&r);
    SetRect(&r, 50, 30, 80, 80);
    FrameRect(&r);

            // duplicate visible window into srcBits
    CopyBits(&realBits, &srcBits, &winRect, &winRect, srcCopy, 0);

    SeedCFill(&srcBits, &destBits, &winRect, &winRect, 25, 25, 0L, 0L);

    CopyMask( &patBits, &destBits, &srcBits,
                &winRect, &winRect, &srcBits.bounds);

    CopyBits( &srcBits, &realBits, &winRect, &winRect, srcCopy, 0 );

    while(!Button())
        ;

    DisposPtr(srcBits.baseAddr);
    DisposPtr(destBits.baseAddr);
    DisposPtr(patBits.baseAddr);

    SetPort(oldPort);
}

void NewBitMap(BitMap *theBits, Rect *rp)
{
    theBits->rowBytes = ((rp->right - rp->left + 15) / 16) * 2;
    theBits->baseAddr =
        NewPtr((long)theBits->rowBytes * (rp->bottom - rp->top));
```

```
    theBits->bounds = *rp;

    if(!theBits->baseAddr)        // out of memory!
        Debugger();
}
```