gx Graphics Library Center lbjoseph

The gx Graphics Library, or gxGL, is a 2D memory drawing library based off of the Win32 GDI libraries.

Table of Contents

gx Graphics Library Center user:lbjoseph

Understanding gxGL

Getting gxGL

Setting Up gxGL

Closing gxGL

Rendering Graphics

Graphics Persistance

Colors

Graphics Concepts

The Pen

The Brush

A Basic Example

Graphics Commands

CLS

FILL color

FILLAT x y

PENCOLOR color

PENWIDTH width

PENSTYLE style

FILLCOLOR color

STYLECOLOR color

BRUSHSTYLE style

BLIT bitmap x y

BLITFIELD bitmap srcX srcY srcW srcH destX destY destW destH

BLITKEY color | none

BLITSTYLE style

BLITANGLE angle

BLITFLIP none | horizontal | vertical | both

BLITORIGIN x y | default

BOX x y width height

ROUNDBOX x y width height [roundRadius] [yRoundRadius]

ELLIPSE x y width height

LINE x1 y1 x2 y2

POLYGON p1x p1y p2x p2y p3x p3y ...

POLYGONMODE alternate | winding

TEXTAT x y

TEXTCOLOR color

FONT face_name sizept [bold] [italic] [strike] [underline]

GETBMP bitmapName x y width height

gxGL Services

Bitmap Dimensions

Font Height

Getting the Most Out of gxGL

Drawing Sprites

The functionality offered by gxGL allows you to draw in memory almost exactly as you would in a Liberty BASIC graphicbox. The commands are slightly different, but allow for more functionality.

Understanding gxGL

gxGL is a library of code. This means you don't have to understand how gxGL works, only how to use it. The subroutines and functions that make gxGL are very easy to call and interact with. gxGL does all of its drawing on a bitmap stored in memory. This makes it very easy to show the entire bitmap at once on a window or graphicbox.

gxGL makes it very easy to create smooth, flicker free animations. By clearing, redrawing, and displaying the memory bitmap many times a second, it is easy to construct frames of animation in memory and display them on your window or graphicbox control. This avoids the traditional flicker with Liberty BASIC graphics, and is technically the same method Liberty BASIC uses to render sprites.

Getting gxGL

Get the Current gxGL Code.

Setting Up gxGL

It's easy to get started using gxGL. All you need to do is tell gxGL how big of a canvas you want to draw on.

```
Call gx.InitMemoryDrawing CanvasWidth, CanvasHeight
```

This tells gxGL to allocate the properly sized surface available so you can begin drawing on it.

Closing gxGL

When your program ends, you need to tell gxGL to clean up after itself. This too is very simple:

```
Call gx.Finish
```

Once you do this, gxGL deletes all the memory drawings and closes the handles to the resources it needs.

Rendering Graphics

It's easy to display the graphics that you've drawn with gxGL on the screen.

```
Call gx.RenderTo window,    destX,destY,    srcAreaX,srcAreaY,
    srcAreaW,srcAreaH
```

The first argument **window** is the handle to a window or graphicbox. This can be obtained like so:

```
window = HWnd(#winHandle.graphicboxHandle)
```

For more information on window handles, see the Liberty BASIC help file.

The second and third arguments (**destX**,**destY**) specify the (x,y) location to render the memory drawing at on **window**.

The fourth and fifth arguments (**srcAreaX**,**srcAreaY**) specify the (x,y) location of the upper left hand corner of the area in the memory bitmap you would like to render.

The final two arguments (**srcAreaW**,**srcAreaH**) specify the width and height of the area in the memory bitmap you would like to render.

If rendering the image isn't enough, you can stretch the in-memory drawing to a particular window or graphics box with the following code:

```
Call gx.StretchTo window,    destX,destY,    destW,destH,
    srcAreaX,srcAreaY,    srcAreaW,srcAreaH
```

Where **destW** and **destH** are the size you would like the image to be stretched to. The rest of the parameters are identical to the gx.RenderTo command.

Graphics Persistance

Once a memory drawing is shown on a graphics window or graphicbox, it can be "flushed" so it will stay shown. This can easily be accomplished by using the native Liberty BASIC command sequence "GETBMP bmpname x y width height; drawbmp bmpname 0 0; flush" - just fill in your own graphics dimensions. For more information, please see Flushing Sprite Graphics (applies to gxGL rendering).

Colors

Colors in gxGL are exactly like the colors in Liberty BASIC commands. You can use one of the 16 colors recognized by LB, or any color in a "R G B" string format. You can even use the "buttonface" color.

Graphics Concepts

Commands are issued to gxGL with a simple subroutine call:

```
Call gx.Draw "cls; fill blue"
```

Commands are separated by a semi-colon, and you can put as many commands in a string as you like. Commands are case-insensitive, which means it doesn't matter if they're uppercase or lowercase.

The Pen

The "pen" in gxGL is used to draw the outlines of shapes. The pen has three properties: width, style, and color.

The Brush

The "brush" in gxGL is used to create filled shapes. The brush is very flexible and can fill shapes in various patterns, much like how the pen can make dashed outlines. The brush is affected by the **FILLCOLOR**, **BRUSHSTYLE**, and **STYLECOLOR**.

A Basic Example

Below is a short example that demonstrates a few basic pen techniques. It sets the **BRUSHSTYLE** to **none** so that shapes are not filled.

```
[Demo1]

    NoMainWin

    WindowWidth = 640
    WindowHeight = 480
    UpperLeftX = Int((DisplayWidth-WindowWidth)/2)
    UpperLeftY = Int((DisplayHeight-WindowHeight)/2)

    Open "gx Graphics Demo 1 - Pens" For Graphics_NF_NSB As #win

    #win "TrapClose [Quit]"
    #win "CLS; Down; Fill White; Flush;"


' Must be called first. You can set the dimensions to be as big as you
 like.

' I won't need to draw anything bigger than the size of the window, ho
wever.
    Call gx.InitMemoryDrawing 640, 480

    centerX = Int(640/2) : centerY = Int(480/2)
    st = 4

    Call gx.Draw "cls; fill 0 0 0"
    ' Let's draw some red and blue circle outlines:
    Call gx.Draw
"penstyle dot; penwidth 1; pencolor blue; stylecolor red; brushstyle n
one;"
    For i = 1 To 100 Step st
        a = i*st
        Call gx.Draw "ellipse ";centerX-a;" ";centerY-a;" ";a*2;" "
;a*2
    Next i

    ' Now, let's show the graphics on screen:

'                             (destination loc.)    (source area start)
    (source area width and height)
```

```
    Call gx.RenderTo hWnd(#win),        0,0,
                  0,0,                      640,480

Wait

[Quit]
    Call gx.Finish
    Close #win
End
```

Graphics Commands

The following section provides reference for all of the graphics commands present in gxGL.

CLS

The **CLS** command clears the drawing and fills it white.

FILL color

The **FILL** command fills the entire memory drawing with the specified **color**.

FILLAT x y

The **FILLAT** command fills the surrounding area of (x,y) with the **FILLCOLOR** until it reaches a color that wasn't the color of the pixel at (x,y).
Think of this is as the paint dipper in your favorite raster graphics application.

PENCOLOR color

The **PENCOLOR** command sets the color of the pen (for outlines) to be the specified color. As with all colors, it can be a single Liberty BASIC color name, or *"r g b"* format.
For example: *"pencolor blue; pencolor 42 100 232"*.

PENWIDTH width

The PENWIDTH command sets the width of the pen (in pixels) to be the following number. For example *"penwidth 4"* will give the pen a thickness of 4 px.

PENSTYLE style

The pen style specifies what kind of pen to use. The following table lists the styles available:

normal	The default setting. This is a solid pen.
solid	Same as normal.
none	The pen is invisible. This allows filled shapes to be drawn without an outline.
insideframe	This is a solid pen that stays within the boundaries of the shape drawn.
dash	This is a dashed pen.
dot	This is a dotted pen.
dashdot	This pen follows a dash-dot pattern.
dashdotdot	This pen follows a dash-dot-dot pattern.

The following pen styles only work when the **PENWIDTH** is 1 px: **dash**, **dot**, **dashdot**, and **dashdotdot**.
The pen style can be used to make decorative pens, solid pens, or make the pen completely invisible.

For example *"penstyle dashdot"*, *"penstyle insideframe"*, and *"penstyle none"* give the pen the respective properties.

FILLCOLOR color

The **FILLCOLOR** command sets the color of the brush to the specified color. The brush color is the color that shapes are filled.
This is also the color that is used with the **FILLAT** command.

For example, *"fillcolor 42 42 96"* or *"fillcolor blue"* will change the fill color to the given colors.

STYLECOLOR color

The **STYLECOLOR** command sets the style color to be the specified color. The style color is different from the **FILLCOLOR** and the **PENCOLOR.** It is the secondary color that is used on dotted pen styles

and pattern brushes. The style color can be optionally turned off to make dotted pens and pattern brushes appear to be transparent. This is done by simply sending a *"stylecolor none"* statement.

BRUSHSTYLE style

The **BRUSHSTYLE** command changes the style of the brush to be specified the style. The brush can even be made invisible by setting it to the "none" style. Brush styles are useful for creating pattern brushes, solid brushes, or making the brush invisible. Below is a list of the valid brush styles.

normal	This is the default style. The brush fills with the solid color specified with the **FILLCOLOR** command.
solid	Same as normal.
none	The brush is invisible. Use this style to draw shapes with only an outline.
vertical	The brush fills with a vertical line pattern.
horizontal	The brush fills with a horizontal line pattern.
cross	The brush fills with a cross pattern.
45	The brush fills with a 45° diagonal-cross line pattern.
45down	The brush fills with a 45° line downward pattern.
45up	The brush fills with a 45° line upward pattern.

For example, "brushstyle 45up", "brushstyle none", and "brushstyle solid" set the respective style for the brush.

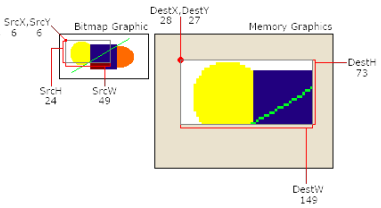
BLIT bitmap x y

This command blits the specified bitmap (that has been loaded via LOADBMP or other means) onto the memory drawing. This is very similar to the Liberty BASIC graphicbox command DRAWBMP.

The BLIT command is influenced by the **BLITSTYLE** and **BLITKEY** properties.

BLITFIELD bitmap srcX srcY srcW srcH destX destY destW destH

The BLITFIELD command can be used to draw a stretched bitmap onto the memory drawing, or draw

(and optionally stretch) part of a bitmap onto the memory drawing. 

The srcX and srcY parameters indicate where the top left corner of the area in the bitmap you want to draw begins.
The srcW and srcH are the width and height of the area in the bitmap you want to draw.

The **destX** and **destY** parameters are the point to render the specified area in the bitmap onto the memory drawing.
The **destW** and **destH** parameters are the width and height that the specified area in the bitmap appears on the memory drawing. If these are larger or smaller than **srcW** and **srcH**, **BLITFIELD** will stretch the bitmap to fit the size. This allows for bitmap scaling.

BLITFIELD is also influenced by the **BLITSTYLE** and **BLITKEY** properties.

Below is an example of drawing a box in memory, rendering it, getting the rendered bitmap, and drawing that bitmap stretched.

```
[Demo3]

    NoMainWin

    WindowWidth = 640
    WindowHeight = 480
    UpperLeftX = Int((DisplayWidth-WindowWidth)/2)
    UpperLeftY = Int((DisplayHeight-WindowHeight)/2)

    Open
"gx Graphics Demo 3 - Blitting Bitmaps" For Graphics_NF_NSB As #win

    #win "TrapClose [Quit]"
    #win "CLS; Down; Fill White; Flush"

    width = 640 : height = 480

    Call gx.InitMemoryDrawing width, height

    Call gx.Draw
"cls; fillcolor pink; brushstyle horizontal; stylecolor darkblue; pens
tyle none"
    Call gx.Draw "roundbox 10 10 100 100 20"
    ' Now get that rendered rounded box as a bitmap:
```

```
    Call gx.Draw "GetBMP box 8 8 104 104"
    ' Now stretch the box bitmap and render it:
    Call gx.Draw "blitstyle normal; blitkey none"
    Call gx.Draw "blitfield box 0 0 104 104 2 2 308 408"
    Call gx.RenderTo hWnd(#win), 0,0, 0,0, width,height

Wait

[Quit]
    Call gx.Finish
    Close #win
End
```

BLITKEY color | none

The **BLITKEY** command specifies a transparent color for drawing bitmaps. This color is not shown when the bitmap is drawn. This allows for using bitmaps as icons or game graphics. Transparency can be easily disregarded by sending a *"blitkey none"* statement.

For example, a *"blitkey blue"* or *"blitkey 255 255 255"* sets the transparent color to be the respective color given. When blitkey is set to a color, the **BLITSTYLE** is ignored when drawing images with a transparent color.

BLITSTYLE style

The **BLITSTYLE** command is an advanced feature that specifies a raster operation mode for drawing bitmaps onto the memory drawing. The style can be one of the following:

normal	Copies the source rectangle directly to the destination rectangle. This is the default mode.
copy	Same as normal.
and	Combines the colors of the source and destination rectangles by using the Boolean AND operator.
or	Combines the colors of the source and destination rectangles by using the Boolean OR operator.
xor	Combines the colors of the source and destination rectangles by using the Boolean XOR operator.
invert	Copies the inverted source rectangle to the destination.

orinvert	Combines the colors of the source and destination rectangles by using the Boolean OR operator and then inverts the resultant color.
invertormerge	Merges the colors of the inverted source rectangle with the colors of the destination rectangle by using the Boolean OR operator.
invertfinal	Inverts the destination rectangle.
invertfinaland	Combines the inverted colors of the destination rectangle with the colors of the source rectangle by using the Boolean AND operator.

For example, *"blitstyle and"*, *"blitstyle invert"*, and *"blitstyle normal"* change the blitstyle to the specified style.

BLITANGLE angle

The **BLITANGLE** command specifies the angle at which bitmaps rendered with **BLIT** and **BLITFIELD** will be rendered at. The origin point (the point the bitmap is rotated around) is specified with **BLITORIGIN**.

For example, *"blitangle 45"* will set the blitting angle to be 45°.

BLITFLIP none | horizontal | vertical | both

The **BLITFLIP** command mirrors (or flips) an image in the specified direction. Unfortunately, this does not work when using a **BLITKEY** for transparent blitting, due to limitations in Windows itself.

none	Draws an image normally.
horizontal	Mirrors the image along the x-axis.
vertical	Mirrors the image along the y-axis.
both	Mirrors the image along both axes.

BLITORIGIN x y | default

The **BLITORIGIN** command specifies the rotation origin for blitting bitmaps. By default, the origin is

assumed to be the middle of the bitmap. For example, *"blitorigin 50 50"* will tell gxGL to render the bitmap rotated by the amount specified by **BLITANGLE** around the point (50,50). Origin points are relative to the bitmap. If you blit a bitmap at (x,y) and specify an origin of (50,75), the bitmap will be drawn rotated around (x+50,y+75).

Specifying *"blitorigin default"* will cause gxGL to revert back to centered origins.

```
[Demo4]

    NoMainWin

    WindowWidth = 640
    WindowHeight = 480
    UpperLeftX = Int((DisplayWidth-WindowWidth)/2)
    UpperLeftY = Int((DisplayHeight-WindowHeight)/2)

    Open
"gx Graphics Demo 4 - Rotating Bitmaps" For Graphics_NF_NSB As #win

    #win "TrapClose [Quit]"
    #win "CLS; Down; Fill White; Flush"

    width = 640 : height = 480

    Call gx.InitMemoryDrawing width, height

    Call gx.Draw
"cls; fillcolor green; brushstyle 45up; stylecolor darkgreen; penstyle
 none"
    Call gx.Draw "roundbox 4 4 100 100 20"
    Call gx.Draw "GetBMP box 2 2 106 106"

[Rotate]
    Timer 0
    angle = angle + 1
    If angle >= 360 Then angle = 0
    Call gx.Draw
"
cl
s; b
litstyle normal; blitkey none; blitorigin default; blitangle ";angle
    Call gx.Draw "blitfield box 0 0 104 104 100 100 204 204"
    Call gx.RenderTo hWnd(#win), 0,0, 0,0, width,height
    Timer 40, [Rotate]
Wait
```

```
[Quit]
    Call gx.Finish
    Close #win
End
```

BOX x y width height

This draws a rectangle with the current pen and brush at (**x**,**y**) with the respective **width** and **height**. The rectangle is outlined with the current pen, and filled with the current brush. Both the pen and the brush can be set to be invisible to create only an outline, or just a filled shape.

For example, *"box 2 2 100 100"* creates a 100x100 rectangle (square) at (2,2).

ROUNDBOX x y width height [roundRadius] [yRoundRadius]

This draws a rounded rectangle with the current pen and brush at (**x**,**y**) with the respective **width** and **height**. The rounded rectangle is outlined with the current pen, and filled with the current brush.

The optional **roundRadius** and **yRoundRadius** parameters specify how much to round the edges of the box. If **yRoundRadius** is omitted, it is assumed to be **roundRadius**. If both are omitted, the rounding radius is assumed to be 10 px.

For example, *"roundbox 2 2 100 100 20"* creates a 100x100 rectangle with 20 px corner rounding at (2,2) on the memory drawing.

ELLIPSE x y width height

This draws an ellipse at (**x**,**y**) with the respective **width** and **height**. The ellipse is outlined with the current pen, and filled with the current brush.

For example, *"ellipse 10 10 150 200"* draws an ellipse at (10,10) with a width of 150 px, and a height of 200 px.

LINE x1 y1 x2 y2

This draws a line from (**x1**,**y1**) to (**x2**,**y2**). The line is drawn with the current pen color and styling.

For example, *"line 20 20 200 200"* draws a line from (20,20) to (200,200) on the memory drawing with the pen.

POLYGON p1x p1y p2x p2y p3x p3y ...

The POLYGON command draws a polygon with the specified point pairs. You can add practically any number of points by just including the point pairs. The fill mode for the polygon is determined by **POLYGONMODE**.

For example, *"polygon 14 10 200 10 300 300 10 400"* draws a quadrilateral with the four given point pairs.

POLYGONMODE alternate | winding

This specifies the fill mode for polygons. It is set to **winding** by default. The difference between polygon fill modes can be found here.

For example, *"polygonmode alternate"* changes the polygon filling mode to **alternate**.

TEXTAT x y

This draws the text that comes after the | symbol in the current query. The text is influenced by the **FONT** and **TEXTCOLOR**.

Here's a short example that demonstrates text drawing:

```
[Demo2]

    NoMainWin

    WindowWidth = 640
    WindowHeight = 480
    UpperLeftX = Int((DisplayWidth-WindowWidth)/2)
    UpperLeftY = Int((DisplayHeight-WindowHeight)/2)

    Open
"gx Graphics Demo 2 - Text Drawing" For Graphics_NF_NSB As #win

    #win "TrapClose [Quit]"
    #win "CLS; Down; Fill White; Flush"

    Call gx.InitMemoryDrawing 640, 480
```

```
    Call gx.Draw
"cls; font Verdana 12; textcolor black; textat 2 2 |Hello, world!"
    Call gx.Draw "textat 2 ";2+gxInfo.FontHeight.struct;
" |The height of this font is ";_
        gxInfo.FontHeight.struct;" px. :)"

    Call gx.RenderTo hWnd(#win), 0,0, 0,0, 640,480

Wait

[Quit]
    Call gx.Finish
    Close #win
End
```

TEXTCOLOR color

Sets the color for text drawing to be the specified color.

FONT face_name sizept [bold] [italic] [strike] [underline]

Sets the font for text drawing to be the specified font. The face name of the font must have _ (underscores) in place of spaces. The size follows in points, and optional attributes (bold, italic, strikeout, underline) can be listed afterwards in any order.

For example, "font comic_sans_ms 14 bold" sets the font to be Comic Sans MS at 14 points and bold.

GETBMP bitmapName x y width height

This command is equivalent to the native Liberty BASIC graphics command. However, this is useful in the sense that you do not have to draw the bitmap onscreen to get it in memory again. You can draw in memory, and get as many bitmaps as you like without the user ever seeing anything.

This command gets the memory drawing area starting at (**x**,**y**) with the specified **width** and **height** and loads it as **bitmapName**. For example, "getbmp mydrawing 10 10 200 200".

gxGL Services

gxGL provides several different functions and variables to make working with graphics easier.

Bitmap Dimensions

To get the dimensions of a bitmap loaded with LOADBMP, use the *gx.BitmapSize$()* function. It returns a string with two words - the first word is the width of the bitmap in pixels, and the second word is the height.

```
size$ = gx.BitmapSize$("bitmapName")
width = Val(Word$(size$,1))
height = Val(Word$(size$,2))
```

Font Height

To get the height of the current font in pixels that is being used by gxGL, just use the *FontHeight* property from the gxInfo structure.

```
fontHeight = gxInfo.FontHeight.struct
```

Getting the Most Out of gxGL

Here are a few other things to help you when using gxGL in your application.

Drawing Sprites

```
Sub DrawSprite bmp$, x, y
    ' Draw a sprite image at x y:
    size$ = gx.BitmapSize$(bmp$) : width = Val(Word$(size$,1))
    height = Val(Word$(size$,2)) : halfh = Int(height/2)
    ' Draw the mask and the sprite.
    Call gx.Draw "blitangle 0; blitorigin default; blitstyle and; blit
field ";_
    bmp$;" 0 0 ";width;" ";halfh;" ";x;" ";y;" ";width;" ";halfh;";";_
    "blitstyle or; blitfield ";bmp$;" 0 ";halfh;" ";width;" ";halfh;"
";x;" ";y;" ";width;" ";halfh;" ; ";_
    "blitstyle normal;"
End Sub

Sub DrawSpriteAdv bmp$, x, y, xScale, yScale, rotDeg, origin$
    ' Draws a sprite scaled, rotated, or both.
    ' x and y is where the sprite will be rendered to.
    ' xScale is a decimal percentage for the rendered width of the spr
```

```
ite image.
    '   To render a sprite with a width of 150%, just make xScale = 1.
5
    '   Basically, the original width of the bitmap is just multiplied
 by xScale.
    ' yScale is the same as xScale, except it is for the height of the
 sprite.
    ' rotDeg is the angle of rotation for the rendered sprite.
    '   If 0, no rotation will be applied.
    ' origin$ is the location of the point relative to the sprite's lo
cation (x,y).
    '   When in doubt, pass "default" to rotate the sprite around it's
 computed center.
    '   Passing a point (such as "5 5") will rotate the sprite around
x+5, y+5.
    size$ = gx.BitmapSize$(bmp$) : width = Val(Word$(size$,1))
    height = Val(Word$(size$,2)) : halfh = Int(height/2)
    newW = Int(width*xScale) : newH = Int(halfh*yScale)
    Call gx.Draw "blitangle ";rotDeg;"; blitorigin ";origin$;"; blitst
yle and; ";_
    "blitfield ";bmp$;" 0 0 ";width;" ";halfh;" ";x;" ";y;" ";newW;" "
;newH;" ;";_
    "blitstyle or; blitfield ";bmp$;" 0 ";halfh;" ";width;" ";halfh;"
";x;" ";y;" ";newW;" ";newH;";";_
    "blitstyle normal;"
End Sub
```

The above subroutines allow a sprite to be easily drawn using gxGL.