

Topaz Systems, Inc.
LCD Developers Info

Topaz Systems, Inc.
650 Cochran Street, Unit 6
Simi Valley, CA, 93065
©1995-2005, all rights reserved, US patent 6,307,955, and pending

www.topazsystems.com
tech support: 805 520-8286
support@topazsystems.com

NOTE: This page contains highly technical information about Topaz LCD products and software interactions and is not designed for the lay end-user.

DEVELOPER'S NOTE: When discussing x, y positions and sizes below, it is important to keep in mind the LCD specs of each Topaz tablet:

SignatureGemLCD1X5 (TL462) and SigLiteLCD (TL460): 240px x 64px (x, y origin: 0, 0 when using LCD coordinates)

SignatureGemLCD4x3 (TL755): 240px x 128px (x, y origin: 0, 0 when using LCD coordinates)

SignatureGemLCD4X5 (TL766): 320px x 240px (x, y origin: 0, 0 when using LCD coordinates)

When using coordinates in your method calls, be sure to keep your positions and sizes within these boundaries.

Tablet Logical Coordinates vs LCD Coordinates and When to Use Them

Tablet Logical coordinates are the original coordinate system designed for Topaz tablets. They are based on a 0, 0 position which exists up and left from where the active area of the tablet begins. So, the point at which the active area of a particular tablet (where the tablet will start capture pen data) never begins at 0, 0 in tablet logical coordinates, and can vary to different degrees. For example, the active coordinate origin for a SignatureGemLCD1X5 is 400, 350. For the SignatureGemLCD4X3, it is 500, 400. Other tablets will vary.

LCD coordinates were added to SigPlus for ease of use in terms of calls to methods where an x and y position are required, for instance [KeyPadAddHotspot\(\)](#). The LCD coordinate system makes the upper-left corner of the lcd itself the 0, 0 position, and the bottom-right corner of the lcd itself the extent of the lcd, which is:

for SignatureGemLCD1X5 (TL462) and SigLiteLCD (TL460): 240, 64
for SignatureGemLCD4x3 (TL755): 240, 128
for SignatureGemLCD4X5 (TL766): 320, 240

In terms of tablet logical coordinates, the top-left corner of the LCD is a counterintuitive value, far from 0, 0 since 0, 0 in tablet coordinates is a position not only far beyond the upper-left corner of the LCD, it is beyond the active area. This is the reason Topaz highly recommends using LCD coordinates over tablet logical coordinates.

Topaz Systems, Inc.
LCD Developers Info

Topaz Systems, Inc.
650 Cochran Street, Unit 6
Simi Valley, CA, 93065
©1995-2005, all rights reserved, US patent 6,307,955, and pending

www.topazsystems.com
tech support: 805 520-8286
support@topazsystems.com

Writing Images and Text to the LCD

Images:

ActiveX: Using either the [LCDWriteBitmap\(\)](#) method or the [LCDWriteFile\(\)](#) method. Both methods take the following 6 arguments: Dest, Mode, XPosition, YPosition, XSize, YSize

Dest: 0=foreground, 1=background

Mode: 0=Clear, 1=Complement, 2=Write Opaque, 3=Write Transparent (mode 2 is the most-often used)

XPosition: The x coord location of the upper-left corner of the image

YPosition: The y coord location of the upper-left corner of the image

XSize: The width of the image to display in px

YSize: The height of the image to display in px

The final argument is the difference between the two methods. [LCDWriteBitmap\(\)](#) takes a Windows handle to the image (HBITMAP). For example, this can be obtained in Visual Basic by adding an Image object to the application, setting the Picture property to the image in question, and then using the Handle property of that object to assign to this argument.

[LCDWriteFile\(\)](#) instead takes the path to the file in question. For example, "C:\somefolder\myimage.bmp".

In either case, the image must be a black and white 1-bit bmp file.

Java: Use the following method:

```
public boolean lcdWriteImage( int dest, int mode, int xPos, int yPos, int xSize, int ySize, Image imageData )
```

The first 6 arguments match the listing for the ActiveX control (see above). The last argument is an Image object, making us of a JPG.

Text:

ActiveX: Using the [LCDWriteString\(\)](#) method. This method takes 8 arguments: Dest, Mode, XPos, YPos, XSize, YSize, Format, HexString

Dest: 0=foreground, 1=background

Topaz Systems, Inc.
LCD Developers Info

Topaz Systems, Inc.
650 Cochran Street, Unit 6
Simi Valley, CA, 93065

www.topazsystems.com
tech support: 805 520-8286
support@topazsystems.com

©1995-2005, all rights reserved, US patent 6,307,955, and pending

Mode: 0=Clear, 1=Complement, 2=Write Opaque, 3=Write Transparent (mode 2 is the most-often used)

XPosition: The x coord location of the upper-left corner of the image

YPosition: The y coord location of the upper-left corner of the image

XSize: The width of the image to display in px

YSize: The height of the image to display in px

Format: NOT CURRENTLY IMPLEMENTED - PASS A 0

HexString: String value to display

If 0's are passed for the size arguments, then the default size will be used. For example:

```
SigPlus1.LCDWriteString(0, 2, 0, 0, 0, 0, 0, "This is a test string")
```

You can also control the font more completely using the `LCDSSetFont()` method. The arguments for this method are all defined in the LOGFONT data structure (see `CreateFont` function of Windows API) in Windows for logical fonts. They are:

Height, Width, Weight, Italic, Underline, PitchAndFamily, FaceName

The FaceName argument is used to specify a particular font. Again, see the `CreateFont` function of Windows API for details on the other arguments. Leaving the arguments at 0 will result in the default for each value to be used. An example is below:

```
SigPlus1.LCDSSetFont(0, 0, 0, 0, 0, 0, "Courier New")
```

Java:

Use the `lcdWriteString()` method, which takes the following arguments:

Dest: 0=foreground, 1=background

Mode: 0=Clear, 1=Complement, 2=Write Opaque, 3=Write Transparent (mode 2 is the most-often used)

XPosition: The x coord location of the upper-left corner of the image

YPosition: The y coord location of the upper-left corner of the image

HexString: String value to display

Font: Font object based on java font class

For example:

```
Font myFont = new Font( "Dialog", Font.BOLD, 12 );  
sigObj lcdWriteString( 0, 2, 10, 10, "This is a sample line of text.", myFont );
```

Topaz Systems, Inc.
LCD Developers Info

Topaz Systems, Inc.
650 Cochran Street, Unit 6
Simi Valley, CA, 93065
©1995-2005, all rights reserved, US patent 6,307,955, and pending

www.topazsystems.com
tech support: 805 520-8286
support@topazsystems.com

Screen Size and Text Limitations

To help decipher the size of a Topaz LCD signature tablet and the size of text the user can use the two methods below, as well as the information provided.

GetLCDSize()

returns the X and Y size of the tablet. The Y size is in the upper 16 bits, X size is in the lower 16 bits

GetLCD textSize(String Str)

returns the size of the string in LCD coordinates using the current LCD font parameters. The Y size is in the upper 16 bits, X is in the lower 16 bits.

Assuming you are using a fixed-width font (with a 5 px spread), you can expect the following (you may need to extrapolate further data based on these stats--and remember these are rough, rule-of-thumb values):

SignatureGemLCD1X5 and SigLiteLCD-

At font size 8, the total number of possible lines is: 12
Number of characters per line: 39

At font size 10, the total number of possible lines is: 10
Number of characters per line: 34

At font size 12, the total number of possible lines is: 9
Number of characters per line: 26

SignatureGemLCD4X3-

At font size 8, the total number of possible lines is: 24
Number of characters per line: 39

At font size 10, the total number of possible lines is: 20
Number of characters per line: 34

At font size 12, the total number of possible lines is: 18
Number of characters per line: 26

SignatureGemLCD4X5-

Topaz Systems, Inc.
LCD Developers Info

Topaz Systems, Inc.
650 Cochran Street, Unit 6
Simi Valley, CA, 93065
©1995-2005, all rights reserved, US patent 6,307,955, and pending

www.topazsystems.com
tech support: 805 520-8286
support@topazsystems.com

At font size 8, the total number of possible lines is: 45
Number of characters per line: 52

At font size 10, the total number of possible lines is: 37
Number of characters per line: 45

At font size 12, the total number of possible lines is: 33
Number of characters per line: 34

All characters after the cutoff will be lost.

Setting the Font

Any font at all (provided it is installed on the machine running the application) can be used. You can specify which by using a method in SigPlus (must be called before the LCDWriteString method):

[LCDSetFont\(Height As Long, Width As Long, Weight As Long, Italic As Integer, Underline As Integer, PitchAndFamily As Integer, FaceName As String\)](#)

The FaceName argument is where you can pass the name of any font you wish. The previous arguments are all defined in the LOGFONT data structure (see CreateFont function of Windows API) in Windows for logical fonts.

Font height: This value is the height of the font in pixels. If this number is negative, the font mapper picks a font with characters with height approximately the absolute value of this number. If it's positive, the font mapper picks a font with a character cell height equal to the height value. The cell height includes some blank internal leading space (more about this later), so this makes the characters a little bit smaller.

Font_width: This value is the width of the font in pixels. If zero, the font mapper uses a default width that matches the height. If the font isn't fixed width, this is the approximate average character width.

Escapement: This is the font's rotation from the horizontal in degrees times 10 (as shown in Figure 2).

Orientation: This should be the orientation of the characters, but Windows assumes orientation is the same as escapement, so it ignores orientation!

Topaz Systems, Inc.
LCD Developers Info

Topaz Systems, Inc.
650 Cochran Street, Unit 6
Simi Valley, CA, 93065

www.topazsystems.com
tech support: 805 520-8286
support@topazsystems.com

©1995-2005, all rights reserved, US patent 6,307,955, and pending

Weight: This specifies the font's weight as a number between 0 and 900. The value 0 selects the default, 400 is normal, and 700 is bold.

Italic: If this value is non-zero, the text is italicized.

Underscore: If this value is non-zero, the text is underscored.

Strikeout: If this value is non-zero, the text is stricken out.

Character_set: This value specifies character sets such as Russian, Greek, and Arabic. Usually you should set this to ANSI_CHARSET (0).

Output_precision: This parameter determines how closely the font selected by the font mapper must match the other parameters you specify. The value OUT_TT_ONLY_PRECIS (7) forces the font mapper to select a TrueType font even if it doesn't match the other parameters exactly. For our purposes, this is reasonable since CreateFont can only rotate TrueType fonts.

Clipping_precision: This determines whether text that lies outside a window's clipping region is clipped by character, stroke, and so forth. When you draw rotated text, you should combine any other value you use here with the value CLIP_LH_ANGLES (16).
quality: This specifies draft or proof quality.

Pitch_and_family: This parameter specifies the pitch (fixed or variable width) and font family used if the font you request is unavailable. If you specify a font that's likely to be present on the computer, like Times New Roman, you don't need to worry about this.

Face_name: This is the font's name—for example, "Times New Roman" or "Courier New."

Managing HotSpots

ActiveX: The [KeyPadAddHotspot\(\)](#) method adds hotspots to the LCD tablet: hotspots are used to determine if the user has tapped within a specific area of the LCD, so the appropriate action can be taken in code. For example, if you place a "clear button" image on the lcd, you'll probably want to place a hotspot under this button so you'll know if the user has tapped it with the pen, and then take the appropriate action in code.

[KeyPadAddHotspot\(\)](#) takes the following 6 arguments:

[KeyCode](#), [Coords](#), [XPos](#), [YPos](#), [XSize](#), [YSize](#)

Topaz Systems, Inc.
LCD Developers Info

Topaz Systems, Inc.
650 Cochran Street, Unit 6
Simi Valley, CA, 93065

www.topazsystems.com
tech support: 805 520-8286
support@topazsystems.com

©1995-2005, all rights reserved, US patent 6,307,955, and pending

Keycode: This is essentially a hotspot numbering system, controlled by the developer

Coords: Coordinate system used for this hotspot: 0 = Logical tablet coordinates, 1 = LCD Coordinates (TOPAZ ALWAYS RECOMMENDS USING 1 = LCD Coordinates).

XPosition: The x coord location of the upper-left corner of the hotspot

YPosition: The y coord location of the upper-left corner of the hotspot

XSize: The width of the hotspot in px

YSize: The height of the hotspot in px

For example: `KeyPadAddHotspot(0, 1, 20, 30, 40, 40)`

(This is hotspot #0, using LCD coordinates. This hotspot's upper left corner is at 20, 30. It is 40 px wide and 40 px tall).

NOTE: hotspots have several characteristics worth mentioning that developers should note:

FIRST, the `KeyPadAddHotspot()` method will require slight variations in px coord location (arguments 3 and 4, varying for about 1 px to 5 px, the closer you get to the bottom of the lcd) from its counterpart `LCDWriteBitmap()`, `LCDWriteFile()`, or `LCDWriteString()` method calls. In other words, whereas `LCDWriteBitmap()`, `LCDWriteFile()`, or `LCDWriteString()` will line up precisely as expected, hotspots will start shifting slightly from 1 to 5 px as you move down the LCD.

For example, if you called:

`LCDWriteBitmap(0, 2, 20, 90, 50, 30)`

your call to `KeyPadAddHotSpot` might be something like:

`KeyPadAddHotspot(0, 2, 18, 87, 52, 33)`

For best results, Topaz recommends the following in terms of adding hotspots:

- a. Make the hotspot larger than the image/text representing it...this eliminates "hunting and tapping" on the part of the user
- b. Make all hotspots no smaller than 10 px in both height and width
- c. Leave at least 10 px of space between hotspots

SECOND, do NOT add all the hotspots you will be using in your application at once if your users will be moving from one "screen" to another. In other words, if you have, for example, a page that the user first makes choices on, then they click something like "OK" (or equivalent) to move to the next "screen" where they might sign, you will need to

Topaz Systems, Inc.
LCD Developers Info

Topaz Systems, Inc.
650 Cochran Street, Unit 6
Simi Valley, CA, 93065

www.topazsystems.com
tech support: 805 520-8286
support@topazsystems.com

©1995-2005, all rights reserved, US patent 6,307,955, and pending
create the hotspots for each page separately.

You do this by:

- a. Create the hotspots for screen 1 ONLY. When you move to the next screen, you
- b. Call `KeyPadClearHotSpotList()` to clear the hotspots out, then
- c. Create the hotspots for the next screen, and so on

For example, let's assume I plan to display 2 screens: the first uses 3 hotspots, and the second uses 2 hotspots.

I would create the first three hotspots something like the following:

```
KeyPadAddHotspot(0, 2, 0, 10, 40, 20)
KeyPadAddHotspot(1, 2, 0, 50, 40, 20)
KeyPadAddHotspot(2, 2, 0, 90, 40, 20)
```

When it's time to move to the next screen, I would first call:

```
KeyPadClearHotSpotList()
```

then I would create my screen 2 hotspots, something like:

```
KeyPadAddHotspot(3, 2, 20, 10, 20, 10)
KeyPadAddHotspot(4, 2, 80, 10, 20, 10)
```

The issue involved is possible overlapping hotspots, which will cause problems to arise.

Java:

In java, all the same rules apply as above. The method calls are also the same

Using HotSpots

ActiveX: Let's assume you have hotspots created as outlined above. You can choose to manage them either by polling at a given rate of time (perhaps 200-300 milliseconds or so) or you can use the SigPlus-provided pen events (Pen_Up and Pen_Down).

Within the polling or pen event routine, you call a method `KeyPadQueryHotSpot()`, which takes the following argument:

KeyCode: This is the number assignment of the hotspot, as assigned in the *KeyCode* argument of the `KeyPadAddHotspot()` method. Call this method within an if statement,

Topaz Systems, Inc.
LCD Developers Info

Topaz Systems, Inc.
650 Cochran Street, Unit 6
Simi Valley, CA, 93065

www.topazsystems.com
tech support: 805 520-8286
support@topazsystems.com

©1995-2005, all rights reserved, US patent 6,307,955, and pending
checking whether the value returned from `KeyPadQueryHotSpot() > 0`.

For example:

```
'assume within polling routine or pen event
```

```
If SigPlus1.KeyPadQueryHotSpot(0) > 0 Then  
'user has tapped hotspot #0  
SigPlus1.ClearTablet  
End If
```

or

```
//assume within polling routine or pen event  
if(sigObj.keyPadQueryHotSpot(0) != 0)  
{  
//user has tapped hotspot #0  
sigObj.clearTablet();  
sigObj.ClearSigWindow(1);  
}
```

It is very important that as soon as you fall within a `KeyPadQueryHotSpot()` routine you clear out the hotspot buffer (as shown in the code above), or your hotspot will trip every time your polling routine runs, or a pen event occurs. You can clear the hotspot buffer by calling `ClearTablet()` and/or `ClearSigWindow(1)`.

Be sure that in your polling routine or pen event routine, you call `ClearTablet()` and/or `ClearSigWindow(1)`, in case you do not trip any hotspots. If your code makes it through the routine without getting into one of the `KeyPadQueryHotspot()` routines, the hotspot buffer will still need to be cleared.

PenEvents and Polling

ActiveX: You will need an event handler for `Pen_Up`, `Pen_Down` or for both, for example:

```
SigPlus1_PenUp()
```

or

```
SigPlus1_PenDown()
```

Topaz Systems, Inc.
LCD Developers Info

Topaz Systems, Inc.
650 Cochran Street, Unit 6
Simi Valley, CA, 93065

www.topazsystems.com
tech support: 805 520-8286
support@topazsystems.com

©1995-2005, all rights reserved, US patent 6,307,955, and pending

Place the code you wish to run in the event of a pen event in the appropriate event handler. The methods used by the tablet which will need to be called in the event handler is:

[KeyPadQueryHotSpot\(\)](#)

Which is used to check if a hotspot has been tapped

It is very important to note that there is a method call necessary for the pen events to function: [SetEventEnableMask\(\)](#). The [SetEventEnableMask\(\)](#) method takes a single argument - [EventMask](#):

EventMask: an integer value used to set the mask for either 1=Pen Down, 2=Pen Up, 3=Both Pen Down and Pen Up

For each time the pen_up or pen_down event fires, the [SetEventEnableMask\(\)](#) method must be called. Therefore, at the end of each event, you'll need to call the [SetEventEnableMask\(\)](#) method to trigger the next pen event. This method is used to offer further control to the developer. A pen event will only fire once and will not fire again until the [SetEventEnableMask\(\)](#) method is called. Otherwise, it could fire repeatedly should the user keep the pen pressed to the tablet.

Instead of using the pen events, you may choose to poll. This methodology entails the use of a timer of some sort that checks at a certain rate for pen activity (calling [KeyPadQueryHotSpot\(\)](#), etc.) Typically, the rate is somewhere between 100-300 ms. This methodology does not require the use of the pen event handlers, nor a call to [SetEventEnableMask\(\)](#).

Java:

In java you can spin a new thread, and set the sleep rate as desired. Something like:

```
public void run()
{
    try
    {
        while (true)
        {
            Thread.sleep(200);
            //code here
        }
    }
}
```

Topaz Systems, Inc.
LCD Developers Info

Topaz Systems, Inc.
650 Cochran Street, Unit 6
Simi Valley, CA, 93065

www.topazsystems.com
tech support: 805 520-8286
support@topazsystems.com

©1995-2005, all rights reserved, US patent 6,307,955, and pending

```
catch (InterruptedException e)
{
}
}
```

No current implementation of the pen events currently exists in java...use the thread/sleep methodology shown above.

Signature Acceptance

ActiveX: There are two methods that work together for this functionality (although you can certainly use them separately should your application have specific requirements):

[SetSigWindow\(\)](#) and [LCDSetWindow\(\)](#)

[SetSigWindow\(\)](#) defines a specific area on the tablet that will accept pen data as "signature" data (collected in the signature object). Outside of this window, pen data will not be added to the signature. This is useful especially when hotspots are used; the area where the hotspots reside can be left out of the signature window, so that hotspot taps do not accumulate in the signature. It is important to indicate to the user, of course, the specific area set aside as the "signature" area (either with text or an image) when using the [SetSigWindow\(\)](#) method. This method takes 5 args - Coords, XPos, YPos, XSize, YSize:

Coords: Coordinate system used for this hotspot:

0 = Logical tablet coordinates, 1 = LCD Coordinates (TOPAZ ALWAYS RECOMMENDS USING 1 = LCD Coordinates).

XPosition: The x coord location of the upper-left corner of the signature window

YPosition: The y coord location of the upper-left corner of the signature window

XSize: The width of the signature window in px

YSize: The height of the signature window in px

For example:

```
SigPlus1.SetSigWindow(1, 3, 30, 122, 30)
```

[LCDSetWindow\(\)](#) defines a specific area on the LCD itself where "electronic ink" is displayed. It is important to note that this method has nothing to do with the captured signature...only the signature as it is displayed on the LCD. Typically, when you call the [SetSigWindow\(\)](#) method to create a window which limits the area in which signature data is captured, you want to also limit the LCD itself to display the signature representation in the same manner. This method takes 4 arguments - XStart, YStart, XSize, YSize:

Topaz Systems, Inc.
LCD Developers Info

Topaz Systems, Inc.
650 Cochran Street, Unit 6
Simi Valley, CA, 93065
©1995-2005, all rights reserved, US patent 6,307,955, and pending

www.topazsystems.com
tech support: 805 520-8286
support@topazsystems.com

XPosition: The x coord location of the upper-left corner of the signature window
YPosition: The y coord location of the upper-left corner of the signature window
XSize: The width of the signature window in px
YSize: The height of the signature window in px

For example:

```
SigPlus1.LcdSetWindow(0, 0, 60, 32)
```

In other words, if you call [SetSigWindow\(\)](#) without calling [LCDSetWindow\(\)](#), you will limit the signature in the SigPlus object to a certain area, but the LCD will display all of the ink drawn by the pen, even if it exceeds the window set by [SetSigWindow\(\)](#). Likewise, if you call [LCDSetWindow\(\)](#) without calling [SetSigWindow\(\)](#), all of what is drawn by the pen will be captured in the SigPlus object, but the LCD will be limited to displaying "electronic ink" in the specified area only.

To inhibit ink on LCD screen completely, use the following code:

```
SigPlus1LCDSetWindow(0,0,0,0)
```

To inhibit ink in the SigPlus object completely, use the following code:

```
SigPlus1.SetSigWindow(1,0,0,1,1)  
(be sure to leave at least one pixel in the Xsize and Ysize arguments)
```

If you are calling the [LCDSetWindow\(\)](#) or [SetSigWindow\(\)](#) and clearing and adding new hotspots in the same routine, be sure to first clear the hotspot list using [SigPlus1.KeyPadClearHotSpotList\(\)](#), then calling [LCDSetWindow\(\)](#) or [SetSigWindow\(\)](#), then adding your new hotspots using the [KeyPadAddHotSpot\(\)](#) method.

The idea is to set the windows while the hotspots are clear

Both SigLiteLCD and SignatureGemLCD tablets of type serial and HSB offer the interactive image displaying/hotspot functionality. So the list of included hardware is:

TL460-B (SigLite LCD serial)
TL460-HSB (SigLite LCD HID-USB)
TL462-B (SignatureGemLCD1X5 serial)
TL462-HSB (SignatureGemLCD1X5 HID-USB)
TL755-B (SignatureGemLCD4X3 serial)

Topaz Systems, Inc.
LCD Developers Info

Topaz Systems, Inc.
650 Cochran Street, Unit 6
Simi Valley, CA, 93065

www.topazsystems.com
tech support: 805 520-8286
support@topazsystems.com

©1995-2005, all rights reserved, US patent 6,307,955, and pending

TL755-HSB (SignatureGemLCD4X3 HID-USB)

TL766-B (SignatureGemLCD4X5 serial)

TL766-HSB (SignatureGemLCD4X5 HID-USB)

TL766-BHSB (SignatureGemLCD4X5 dual serial/HID-USB)

Software tools that support this interactive functionality are:

SigPlus ActiveX (<http://www.topazsystems.com/Software/sigplus.exe>)

SigPlusNET.NET assembly (<http://www.topazsystems.com/Software/sigplusnet.exe>)

SigPlus Java (<http://www.topazsystems.com/Software/sigplusjava>)

Demos for java are already included in the zip with the java bean; the same is true of the shared c lib.

All other demos for the ActiveX and the .NET assembly are available at <http://www.topazsystems.com/Software/download/developers.htm>

Any and all demos referencing LCD tablets in the description (see the Visual Basic 6.0, .NET (Using SigPlus ActiveX), .NET (Using SigPlus Pro .NET assembly), Visual C++, WebSign™ software ASP and HTML Internet Examples sections) are using the interactive graphics and hotspots, and are excellent examples with code.