DELGEN

X-Graph Software Module

xgGUI Users Manual

Version 1.0 – August 16, 2007

If you have any remarks on this document, please report them to DELGEN.

Printed in Luxembourg.
Document Nr: XGGUI-001

Content

Content....3
Figures....5
Tables....7
1 Welcome....9
1.1 Introduction....9
1.2 How This Book Is Organized....9
1.3 More Questions....9
2 X-Graph GUI (xgGUI)....11
2.1 xgGUI Structure....11
2.2 xgGUI Low-Level Graphic functions....13
2.2.1 LCD type selection....13
2.2.2 LCD Style – Configuration....13
2.2.3 Graphic.lib Compatibility....13
2.2.4 Macro's....13
2.2.5 Initialization....14
2.2.6 Screen Locking....14
2.3 Graphic Primitives....15
2.3.1 Color....15
2.3.2 Clear & Fill Screen....16
2.3.3 Dot....16
2.3.4 Lines....17
2.3.5 Boxes....18
2.3.6 Circles....19
2.3.7 Polygons....19
2.3.8 Text....21
2.3.9 Language....24
2.3.10 Bitmaps....25
2.3.11 Scrolling....27
2.3.12 Clipping....28
2.3.13 Touchscreen....29
2.4 xgGUI Widgets....30
2.4.1 xgGUI Graphic Objects....30
2.4.1.1 Graphic Structures....30
2.4.1.2 Test Pattern....30
2.4.1.3 Cursor....30
2.4.1.4 Scroll List....31
2.4.2 DynamicC Widgets....32
2.4.2.1 Text Window Functions....32
2.4.2.2 Graphic Menu's....35
2.4.3 X-Graph Widgets....36
3 easyGUI....37
4 xgGUI Design Tools....39
Warranty....41
Notice to Users....43
Software License Agreement....45
Change List....47

Figures

Figure 1: xgGUI Structure.....12

Tables

1 Welcome

1.1 Introduction

The xgGUI libraries contain a set of low-level hardware (LCD type) independent graphic functions, an X-Graph interface to the easyGUI PC application, a collection of graphic widgets and an X-Graph GUI design tool.

The low level functions include the classic graphic primitives i.e. line, box, circle, text, bitmap drawing etc… These functions are independent of the X-Graph LCD type and offer an easy upgrade path. For example, software written for a QVGA B/W LCD can be recompiled to work without changes on a VGA color LCD.

The low level functions are compatible with the easyGUI PC application (www.easygui.com). easyGUI let's you create a complete graphical user interface on your PC, without the need for downloading and trial-and-error sessions. Once the GUI is designed it generates C source files. easyGUI supports graphical widgets, multiple fonts and multiple languages.

A complete collection of graphic widgets will be available in the near future.
Also an X-Graph GUI design tool is under development. This tool will allow GUI design directly on the X-Graph module by using the touchscreen and drag-and-drop.

These libraries support both the DynamicC and Softools WinIDE compilers. Note that libraries for both compilers might have the same names, but contain different source code. The files are installed in the proper directories of each compiler.

Note that DELGEN is currently testing this library, not all functions are available now. The WinIDE version is in beta stage.

1.2 How This Book Is Organized

You can find following chapters in it:

Chapter 1 contains a view on all the information in this book.

Chapter 2 includes information on the xgGUI structure, graphic primitives and widgets.

Chapter 3 is dedicated to the easyGUI X-Graph interface.

The X-Graph Graphic Design Center information can be found in **Chapter 4**.

1.3 More Questions

If you have questions while using this X-Graph software module, check first if the information is available in this book. If you cannot find the answer check the information and forum on the X-graph website (www.x-graph.be). Finally you can also contact your local distributor or the X-Graph technical support by e-mail (techsup@x-graph.be).

This manual includes all available documentation on this X-Graph software module. It is strongly advised to download and read documentation on the Rabbit processor available from the Rabbit Semiconductor (www.rabbitsemiconductor.com) website. This manual is complimentary to the documentation found on these websites.

2 X-Graph GUI (xgGUI)

2.1 xgGUI Structure

Have a look at the block diagram on the next page. This diagram describes the structure of the xgGUI graphic libraries.

The application user interface can make function calls to:

- the xgGUI library (XGGUI.LIB, section 2.2): draw lines, dots, text, …
- xgGUI widgets (XGGUI_WIDGET.LIB, section 2.4): draw text boxes, drop-down menus, clocks, …
- the bitmap library (XGGUI_BMP.LIB, section 2.2): display .bmp files
- the xgGUI graphic objects (see lower)
- the DynamicC graphics.lib compatibility layer

The xgGUI library contains user interface accessible functions for all graphic primitives and for xgGUI graphic object control (section 2.2 and 2.4). This library includes the default fonts library (XGGUI_FONT_DEFAULT.LIB) when the GUI_LOAD_DEFAULT_FONTS macro is enabled in the jumpstart.c file.

The XGGUI_PRIM.LIB library is the graphic hardware abstraction layer. It automatically adapts to the correct LCD type. It's of course important to define the correct LCD type in Step 1 of the jumpstart.c file.

Section 2.4 includes information on the xgGUI Widgets. All the widgets are stored in the XGGUI_WIDGET.LIB library. You will find functions to handle text boxes, drop-down boxes, file selectors, clocks, etc …. DELGEN is currently building a complete graphical widget library.

The XGGUI_BMP.LIB library has functions to read PC-style BMP files and copy the graphical contents to the video frame buffer.

Screendumps (for creating manuals) can be made with the functions in SCREENDUMP.LIB. This allows you to push BMP files to an FTP server. The macro XG_SCREENDUMP must be enabled in the jumpstart.c file (Step 3).

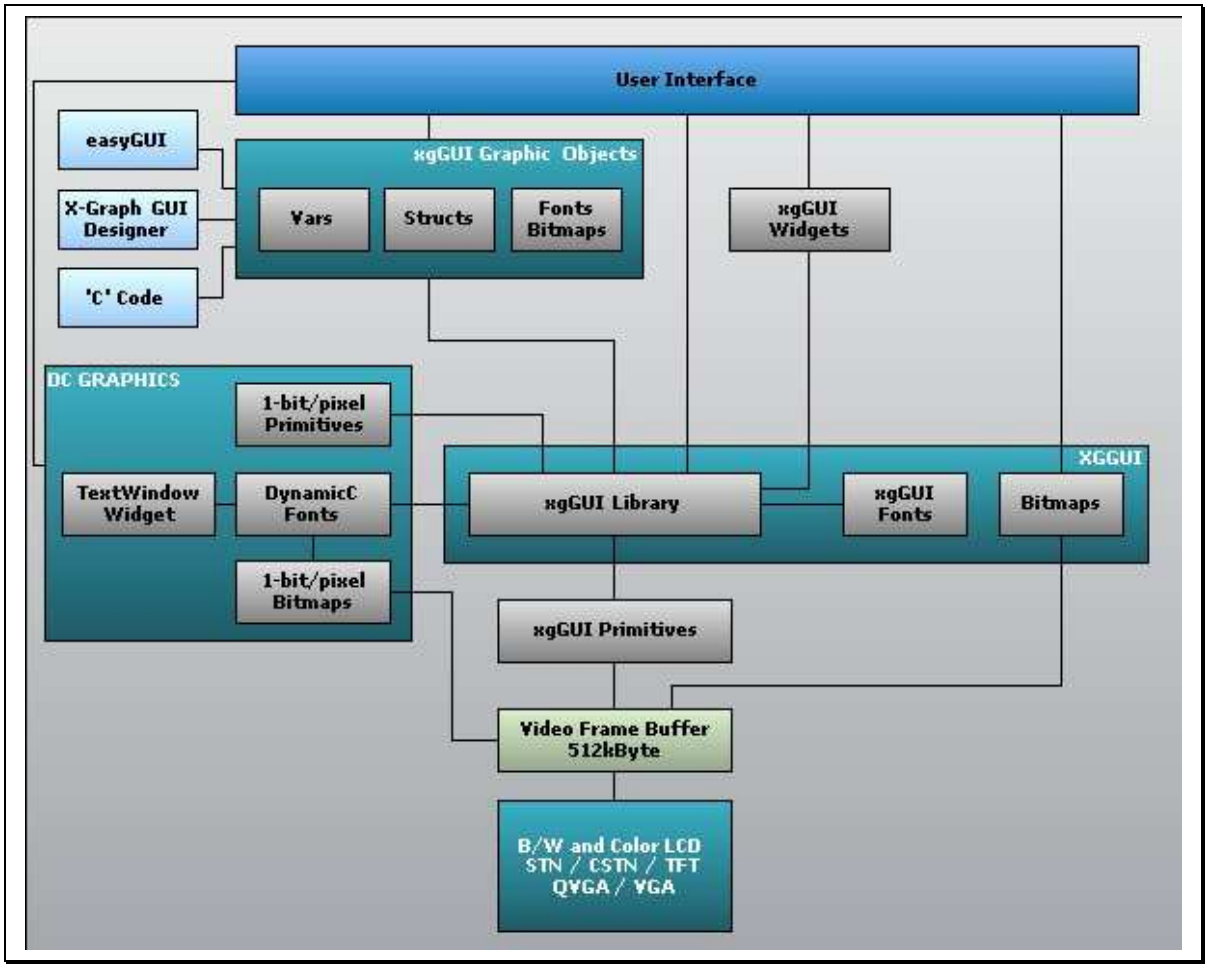

Figure 1: xgGUI Structure

The DynamicC graphic.lib compatibility layer includes four files. The XGGUI_DC_GRAPHIC.LIB file is a placeholder for the graphic.lib primitive functions. These functions are routed to standard xgGUI primitives. Read the header file for more info.
The …_FONTS.LIB, …_TEXTWINDOW.LIB and …_BITMAP.LIB contains slightly adjusted versions of the fonts, text window and bitmap sections of the graphic.lib library. The bitmap functions only support 1-bit/pixel LCD's. For other LCD's the xgGUI library should be used.
The other functions (primitives, fonts, textwindow) can be used with all available LCD's.

The xgGUI graphic objects can be created by the easyGUI PC application (section 3), the X-Graph GUI Designer (4) or just by including direct 'C' code. The complete in C code converted objects are typically stored in three separate files. These files are application dependant and actually part of the application software. Three empty framework files are included: XGGUI_VAR.LIB, XGGUI_STRUCT.LIB and XGGUI_FONTS.LIB. See section 3 to learn how to build these files.
If you do not use the xgGUI graphic objects, it's required to enable the GUI_LOAD_DEFAULT_OBJECTS macro in the jumpstart.c file.

Both the xgGUI bitmap functions and the 1-bit/pixel graphic.lib bitmap functions bypass the xgGUI HAL and directly write to the video frame buffer.
You do NOT need to include any of the mentioned library files. By enabling the XG_LCD macro in the jumpstart.c application, the XGGUI.LIB is automatically included. This library loads all the required other libraries.

2.2 xgGUI Low-Level Graphic functions

The xgGui.lib library contains a set of classic graphic primitives to draw lines, boxes, circles, text, bitmaps etc… All primitive function names start with xgGui…. (ex. xgGuiLine and xgGuiDot).

2.2.1 LCD type selection

The xgGUI library functions are hardware independent and work will all the different X-Graph LCD's. This independency is on the source file level. Before compiling or linking the library, the correct LCD type needs to be selected.
For applications with different LCD types, merely a recompilation with the correct LCD type definition is needed.

The LCD type is selected based on the macro definitions in the jumpstart application. Open this c file and check Step 2 of the configuration. This step lists macros of all available LCD's. Just select the one you're using.

Make sure you select only one LCD type.

2.2.2 LCD Style – Configuration

The WinIDE compiler allows execution time selection of the LCD type. More information will be included in the next manual release.

2.2.3 Graphic.lib Compatibility

DynamicC includes a graphic primitives library (graphic.lib) with support for 1-bit/pixel (= B/W) LCD's. It supports lines, circles, polygones, fonts, bitmaps, etc… The functions available in this low-level library are used in several higher level libraries to support menu's, touchscreens, etc…

To maintain source level compatibility with the DynamicC graphic.lib library, wrap-around functions are available for most graphic.lib functions. This compatibility is only needed to compile existing Z-World modules based software. All native X-Graph software should preferable use the xgGui.lib standard functions.
This compatible library allows users with an installed software base using the graphic.lib library, to port their applications very fast.

Note that the DynamicC graphic.lib only support B/W 1-bit/pixel LCD's. The xgGui.lib library extends this to all types of X-Graph LCD's also for the functions available in graphics.lib (except for bitmap functions).

Note, that some functions of graphic.lib are intended to be used only by other graphic.lib functions. These 'local functions' should not be used with Color LCD X-Graph modules.

Read below sections on text and bitmap primitives for compatibility issues with graphic.lib.

2.2.4 Macro's

To be defined.

2.2.5 Initialization

void xgGui_init(void);
void glInit(void);
Initializes the display devices, clears the screen. This function call must be made prior to any other graphic function calls. It should be called only once !!

PARAMETER
None

RETURN VALUE
None

2.2.6 Screen Locking

The XG5000 LCD hardware allows direct zero-waitstate writing to the LCD video buffer. In most situations this system is fast enough to get smooth LCD updating. It's a lot faster compared to dedicated LCD driver solutions which required multiple waitstates or a ready line structure to access the video buffer.

By using a well structured GUI strategy, screen updates can be made instantaneous without any visual flickering or update side effects.

If needed a dual buffer system can be implemented, given enough memory is available. A dual buffer system uses a paint buffer for all graphic drawing. An application can build a complex graphical image in the paint buffer and once that's done a command is given to copy the paint buffer to the LCD refresh buffer. This is nearly instantaneous and gives in all situations no visual flickering or update side effects.

To enable the dual buffer system the XG_LCD_DOUBLE_BUFFER macro must be enabled in the jumpstart application.

There are considerable disadvantages when using a dual buffer system. Such a system requires the double amount of memory. For a small QVGA B/W system this is no problem, but for a VGA LCD in 8-bit/pixel mode it poses a problem (not enough memory on the standard X-Graph module for double buffering).

In a double buffered system, the software has to take care for locking and unlocking the copying from the paint to refresh buffer. Is this is not done; each executed graphic function will force a refresh buffer update and considerable slow down the system. For example, drawing a line is done in the paint buffer with an optimized assembly function. But the copy process from paint to refresh buffer will always copy a full block, taking up maybe 100 times more cpu time compared to the actual line drawing.

Before the user interface application starts a block of graphic commands, the glBuffLock() function must be called. This will lock the copying of the paint to the refresh buffer. Once all graphic commands are done, the application needs to call the glBuffUnLock() function which will start the actual copy process. Only the 'changed' area will be copied.

void glBuffLock(void);
If used this function increments the LCD screen-locking counter. Graphics calls are recorded in a LCD paint buffer, and are not transferred to the LCD refresh buffer if the counter is non-zero (only in XG_LCD_DOUBLE_BUFFER systems).

NOTE: The functions glBuffLock() and glBuffUnlock() can be nested up to a level of 255, but be sure to balance the calls. It is not a requirement to use these procedures, but a set of glBuffLock() and glBuffUnlock() bracketing a set of related graphics calls significantly speeds things.

PARAMETER
None

RETURN VALUE
None

void glBuffUnLock(void);
See notes on the glBuffLock() function.

If used the function decrements the LCD screen-locking counter. The contents of the LCD buffer are transferred to the LCD if the counter goes to zero.

PARAMETER
None

RETURN VALUE
None

void glSwap(void);
See notes on the glBuffLock() function.

If used the function checks the LCD screen-locking counter. The contents of the paint buffer is transferred to the refresh buffer if the counter is zero. This function is automatically called by all graphic primitive functions and by glBuffUnLock(). Normally it's never required to call this function in the user application program.

PARAMETER
None

RETURN VALUE
None

2.3 Graphic Primitives

The graphic primitives functions are listed here arranged per subject. When available both the xgGui.lib function and the wrap-around graphic.lib functions are listed.

2.3.1 Color

These color functions are only needed for graphic.lib functions. The standard xgGui.lib library functions have build-in color support and never use these functions.

void glSetBrushType(int type);
Sets the drawing method (or color) of pixels drawn by subsequent graphics calls.
This function is only valid for B/W and CSTN Color LCD's.

PARAMETER for B/W LCD's:
PIXBLACK draws black pixels
PIXWHITE draws white pixels
PIXXOR draws old pixel XOR'ed with the new pixel

PARAMETER for CSTN Color LCD's:
CPIX_BLACK
CPIX_RED
CPIX_YELLOW
CPIX_GREEN
CPIX_MAGENTA

CPIX_BLUE
CPIX_CYAN
CPIX_WHITE

RETURN VALUE
None

int glGetBrushType(void);
Gets the current method (or color) of pixels drawn by subsequent graphics calls.
This function is only valid for B/W and CSTN Color LCD's.

PARAMETER
None

RETURN VALUE
The current brush type.

unsigned int glColor;
Only for TFT LCD's: sets the active color.
This variable is used in most of the graphic.lib graphic primitive functions to draw in the selected color.

2.3.2 Clear & Fill Screen

void xgGuiClear(void);
void glBlankScreen(void);
Blanks the LCD display screen, clears flags for cursors, auto redraw items and scrolling.

PARAMETER
None

RETURN VALUE
None

void xgGuiFillScreen(unsigned int iColor);
void glFillScreen(int pattern);
Fills the LCD display screen with a pattern on B/W or the specified color on Color LCD's. For the glFillScreen version the parameter is ignored on Color LCD's. In stead the active color is used to fill the screen.

The function can be used to paint the screen with a selected color before any other graphic objects are drawn, i.e. use a background color.

PARAMETER
B/W:
0xFF = all black
0x00 = all white
anything else = vertical stripes
Color: the selected or active color
RETURN VALUE
None

2.3.3 Dot

void xgGuiDot(int x, int y, unsigned int color);
void glPlotDot(int x, int y);
Draws a single pixel on the LCD.

If the coordinates are outside the LCD display area or the clipping area, the dot will not be plotted.

PARAMETERS
x is the *x* coordinate of the dot
y is the *y* coordinate of the dot
color is the color of the dot

RETURN VALUE
None

2.3.4 Lines

void xgGuiLine(int x1, int y1, int x2, int y2, unsigned int color);
void glPlotLine(int x1, int y1, int x2, int y2);
Draws a line on the LCD. The function will automatically call a faster function to draw vertical and horizontal lines. Any portion of the line that is beyond the LCD display area or clipping area will be clipped.

PARAMETERS
x1 is the *x* coordinate of one endpoint of the line
y1 is the *y* coordinate of one endpoint of the line
x2 is the *x* coordinate of the other endpoint of the line
y2 is the *y* coordinate of the other endpoint of the line
color is the color of the line

RETURN VALUE
None

void xgGuiVLine(int x, int y1, int y2, unsigned int color);
Draws a vertical line. Any portion of the line that is beyond the LCD display area or clipping area will be clipped.

PARAMETERS
x is the *x* coordinate of both endpoints of the line
y1 is the *y* coordinate of one endpoint of the line
y2 is the *y* coordinate of the other endpoint of the line
color is the color of the line

RETURN VALUE
None

void xgGuiHLine(int x1, int x2, int y, unsigned int color);
Draws a horizontal line. Any portion of the line that is beyond the LCD display area or clipping area will be clipped.

PARAMETERS
x1 is the *x* coordinate of one endpoint of the line
x2 is the *x* coordinate of the other endpoint of the line
y is the *y* coordinate of the both endpoints of the line
color is the color of the line

RETURN VALUE
None

2.3.5 Boxes

void xgGuiBox(int x1, int y1, int x2, int y2, unsigned int color);
Draws a single pixel wide rectangle. Any portion of the rectangle that is beyond the LCD display area or clipping area will be clipped.

PARAMETERS
x1 is the *x* coordinate of one corner of the box
y1 is the *y* coordinate of one corner of the box
x2 is the *x* coordinate of the other corner of the box
y2 is the *y* coordinate of the other corner of the box
color is the color of the line

RETURN VALUE
None

void xgGuiFillBox(int x1, int y1, int x2, int y2, unsigned int color);
void glBlock(int x, int y, int bmWidth, int bmHeight);
Draws a rectangular block on the LCD. Any portion of the box that is outside the LCD display area or clipping area will be clipped.

PARAMETER
x1 is the *x* coordinate of one corner of the box
y1 is the *y* coordinate of one corner of the box
x2 is the *x* coordinate of the other corner of the box
y2 is the *y* coordinate of the other corner of the box
color is the color of the box
x is the *x* coordinate of the upper left corner of the block
y is the *y* coordinate of the left top corner of the block
bmWidth is the width of the block
bmHeight is the height of the block

RETURN VALUE
None

void xgGuiInvertBox(int x1, int y1, int x2, int y2);
Inverts a rectangular box on the LCD. Any portion of the block that is outside the LCD display area or clipping area will be clipped.

PARAMETER
x1 is the *x* coordinate of one corner of the box
y1 is the *y* coordinate of one corner of the box
x2 is the *x* coordinate of the other corner of the box
y2 is the *y* coordinate of the other corner of the box

RETURN VALUE
None

void xgGuiBlinkBoxStart(int x1, int y1, int x2, int y2, int Rate);
Starts blinking a rectangular block on the LCD. Any portion of the block that is outside the LCD display area or clipping area will be clipped.

PARAMETER
x1 is the *x* coordinate of one corner of the box
y1 is the *y* coordinate of one corner of the box
x2 is the *x* coordinate of the other corner of the box
y2 is the *y* coordinate of the other corner of the box
Rate is the blinking rate in multiple times caling the xgGui_tick() function. The valid range is 0 to 255.

RETURN VALUE
None

void xgGuiBlinkBoxStop(void);
Stops blinking the currently active blinking box.

PARAMETER
None

RETURN VALUE
None

2.3.6 Circles

void xgGuiCircle(int x, int y, int r, unsigned int iColor);
void glPlotCircle(int x, int y, int r);
Draws a circle on the LCD. Any portion of the circle that is outside the LCD display area or clipping area will be clipped.

PARAMETERS
x is the *x* coordinate of the center of the circle
y is the *y* coordinate of the center of the circle
r is the radius of the circle (in pixels)
iColor is the color of the circle

RETURN VALUE
None

void xgGuiFillCircle(int x, int y, int r, int unsigned iColor);
void glFillCircle(int x, int y, int r);
Draws a filled circle on the LCD. Any portion of the circle that is outside the LCD display area or clipping area will be clipped.

PARAMETERS
x is the *x* coordinate of the center of the circle
y is the *y* coordinate of the center of the circle
r is the radius of the circle (in pixels)
color is the color of the circle

RETURN VALUE
None

2.3.7 Polygons

A polygon is an object with three or more sides. Typical examples of polygons are triangles, parallelograms, trapeziums and diamonds. But any other style of polygon can also be drawn.

Individual lines of a polygon can cross each other.

Polygons are very useful in GUI design to drawn triangles and pointers.

void xgGuiPolygon(int n, unsigned int iColor, int x1, int y1, int x2, int y2, …);
void glPlotPolygon(int n, int x1, int y1, int x2, int y2, …);
Plots the outline of a polygon on the LCD. Any portion of the polygon that is outside the LCD display area or clipping area will be clipped. The function will also return, doing nothing, if there are less than 3 vertici.

PARAMETERS
n is the number of vertici
iColor is the color of the polygon
x1 is the *x* coordinate of the first vertex
y1 is the *y* coordinate of the first vertex
x2 is the *x* coordinate of the second vertex
y2 is the *y* coordinate of the second vertex
... coordinates of additional vertici

RETURN VALUE
None

void xgGuiFillPolygon(int n, unsigned int iColor, int x1, int y1, int x2, int y2, …);
void glFillPolygon(int n, int x1, int y1, int x2, int y2, …);
Draws a filled polygon on the LCD. Any portion of the polygon that is outside the LCD display area or clipping area will be clipped. The function will also return, doing nothing, if there are less than 3 vertici.

PARAMETERS
n is the number of vertici
iColor is the color of the polygon
x1 is the *x* coordinate of the first vertex
y1 is the *y* coordinate of the first vertex
x2 is the *x* coordinate of the second vertex
y2 is the *y* coordinate of the second vertex
... coordinates of additional vertici

RETURN VALUE
None

*void xgGuiVPolygon(int n, unsigned int iColor, int *pFirstCoord);*
*void glPlotVPolygon(int n, int *pFirstCoord);*
Plots the outline of a polygon on the LCD. Any portion of the polygon that is outside the LCD display area or clipping area will be clipped. The function will also return, doing nothing, if there are less than 3 vertici.

PARAMETERS
n is the number of vertici
iColor is the color of the polygon
pFirstCoord is a pointer to an array of vertex coordinates x1,y1, x2,y2, x3,y3, ...

RETURN VALUE
None

*void xgGuiFillVPolygon(int n, unsigned int iColor, int *pFirstCoord);*
*void glFillVPolygon(int n, int *pFirstCoord);*
Draws a filled polygon on the LCD. Any portion of the polygon that is outside the LCD display area or clipping area will be clipped. The function will also return, doing nothing, if there are less than 3 vertici.

PARAMETERS
n is the number of vertici
iColor is the color of the polygon
pFirstCoord is a pointer to an array of vertex coordinates x1,y1, x2,y2, x3,y3, ...

RETURN VALUE
None

2.3.8 Text

The graphic.lib and xgGui.lib use two different approaches to draw text. Both can be used simultaneously. The graphic.lib approach uses 1-bit/pixel bitmap files which can be created with a utility included with DynamcC. These 1-bit/pixel bitmap fonts can be used as-is with all types of X-Graph LCD's. The fonts are converted to xx-bit/pixel as needed. Several standard fonts are included with DynamicC.

The xgGui.lib uses easyGUI compatible font description tables. Depending on the font type and scale this system takes less or more memory space compared to the graphic.lib approach. Several 'default' font files are included with the xgGui libraries. More fonts can be created using easyGUI.
These fonts can be used much more flexible compared to graphic.lib fonts. They all allow for proportional writing, transparency, underlining, alignments, background boxes with many options and background colors.

Note that the xgGui.lib functions for both ANSI and UniCode support are included, but the current library release does not yet support UniCode.

void xgGuiDrawChar(int x, int y, int font, char character, unsigned int color);
Draws a single character on the display. Any portion of the character that is outside the LCD display area or clipping area will be clipped.
For graphic.lib fonts use the glPutFont() function.

PARAMETERS
x is the *x* coordinate (column) of the upper left corner of the text
y is the *y* coordinate (row) of the left top corner of the text
font is the font index
character is the character code
color is the color of the character

RETURN VALUE
None

*void xgGuiDrawStr(int x, int y, int font, int CharSetSelector, char *String, char Alignment, char PsWriting, char Transparent, char Underlining, int BackBoxSize, char BackBorderPixels, unsigned int ForColor, unsigned int BackColor);*
Draws a formatted string on the display. Any portion of the string that is outside the LCD display area or clipping area will be clipped.
Also check the xgGuiPrintf() function which provides some compatibility with the glPrintf() function.

PARAMETERS
x is the *x* coordinate (column) of the upper left corner of the text
y is the *y* coordinate (row) of the left top corner of the text
font is the font index
CharSetSelector:
- -1 selects the default character set
- 0 selects the standard ANSI character set
- >0 selects special national character sets.

(only used in ANSI character mode, not used in UniCode mode)
String: A zero terminated text string.
Alignment:
- GUI_ALIGN_LEFT starts text writing from the X coordinate.
- GUI_ALIGN_CENTER centers text writing around the X coordinate.
- GUI_ALIGN_RIGHT positions the text so that it ends on the X coordinate.

Proportional writing:
- GUI_PS_OFF turns off proportional writing.

- GUI_PS_ON turns on proportional writing.
- GUI_PS_NUM uses numerical proportional writing.

Transparent can be:

- GUI_TRANSPARENT_OFF turns transparent writing off, i.e. the background is painted.
- GUI_TRANSPARENT_ON turns transparent writing on, i.e. only the text is painted.

Underlining can be:

- GUI_UNDERLINE_OFF
- GUI_UNDERLINE_ON

Background box size. Determines the size of a background box. Zero means no background box.
Border pixels for background box. One extra pixel can be added to the background box on each of its edges:

- GUI_BBP_NONE. No extra pixels.
- GUI_BBP_LEFT. One extra pixel on the left edge.
- GUI_BBP_RIGHT. One extra pixel on the right edge.
- GUI_BBP_TOP. One extra pixel on the top edge.
- GUI_BBP_BOTTOM. One extra pixel on the bottom edge.

The last four settings can be combined, like e.g.:
GuiLib_BBP_TOP + GuiLib_BBP_BOTTOM.
Foreground color: determines the text color.
Background color: determines the background color (if used, either for normal background (=non transparent) or background box).

RETURN VALUE
None

*char *xgGuiGetTextPtr(unsigned int Structure, unsigned int TextNr);*
Returns pointer to text in structure.

PARAMETERS
Structure ID.
Text Nr: 0 is first text in structure, items other than texts are ignored.

RETURN VALUE
Pointer to text based on structure, textnr and current language.
Returns NULL if no text was found.

*unsigned int xgGuiGetTextWidth(char *String, xgGuiFontRecConstPtr Font, char PsWriting);*
Returns width of text in pixels.

PARAMETERS
Strint: pointer to text string
Font: pointer to xgGui font descriptor
PsWriting:

- GUI_PS_OFF turns off proportional writing.
- GUI_PS_ON turns on proportional writing.
- GUI_PS_NUM uses numerical proportional writing.

RETURN VALUE
0: error
other value: width of text in pixels

*void glXFontInit(fontInfo *pInfo, char pixWidth,char pixHeight, unsigned startChar, unsigned endChar, unsigned long xmemBuffer);*
Initialise the graphic.lib style font descriptor structure. This function must be called before using any functions with the font. Fonts are always stored in extended memory

and don’t require root memory. Each font character's bitmap is column-major and byte-aligned.

PARAMETERS
pInfo is a pointer to the font descriptor to be initialized
pixWidth is the width of each font item (in pixels)
pixHeight is the height of each font item (in pixels)
startChar is the value of the first printable character in the font character set
endChar is the value of the last printable character in the font character set
xmemBuffer is an xmem pointer to a linear array of font bitmaps

RETURN VALUE
None

*void glPutFont(int x, int y, fontInfo *pInfo, char code);*
This function can only be used with graphic.lib fonts, NOT with xgGUI fonts. Use the xgGuiDrawChar() functions with xgGUI fonts

Puts an entry from the font table to the video page buffer. Any portion of the bitmap character that is outside the LCD display area or clipping area will be clipped.

PARAMETERS
x is the *x* coordinate (column) of the upper left corner of the text
y is the *y* coordinate (row) of the left top corner of the text
pInfo is a pointer to the window frame descriptor
code is the ASCII character to display

RETURN VALUE
None

*void xgGuiPrintf(int x, int y, int font, unsigned int iColor, char *fmt, …);*
*void glPrintf(int x, int y, fontInfo *pInfo, char *fmt, …);*
Prints a formatted string (much like printf) on the LCD screen. Only the character codes that exist in the font set are printed, all others are skipped over. For example, '\b', '\t', '\n', and '\r' (ASCII backspace, tab, newline, and carriage return, respectively) will be printed if they exist in the font set, but will not have any effect as control characters. Any portion of the bitmap character that is outside the LCD display area or clipping area will be clipped.

PARAMETERS
x is the *x* coordinate (column) of the upper left corner of the text
y is the *y* coordinate (row) of the left top corner of the text
font is the font index (xgGUI fonts)
color is the color of the character
pInfo is is a pointer to the window frame descriptor (graphic.lib fonts)
fmt is a formatted string
... is a formatted string of conversion parameter(s)
EXAMPLE
glprintf(0,0, &fi12x16, "Test %d\n", count);

RETURN VALUE
None

void xgGuiSetPfStep(int stepX, int stepY);
void glSetPfStep(int stepX, int stepY);
This function can only be used with graphic.lib fonts, NOT with xgGUI fonts.

Sets the glPrintf() printing step direction. The *x* and *y* step directions are independent signed values. The actual step increments depend on the height and width of the font being displayed, which are multiplied by the step values.
Use glGetPfStep() to examine the current *x* and *y* printing step direction.
With this function it's possible to change the printing direction of strings. I.e. it's possible to print a vertical aligned text or, less useful, an text going up-down or right-left.

PARAMETERS
stepX is the glPrintf *x* step value
stepY is the glPrintf *y* step value

RETURN VALUE
None

int xgGuiGetPfStep(void);
int glGetPfStep(void);
This function can only be used with graphic.lib fonts, NOT with xgGUI fonts.

Gets the current glPrintf() printing step direction. Each step direction is independent of the other, and is treated as an 8-bit signed value. The actual step increments depends on the height and width of the font being displayed, which are multiplied by the step values.
Use glSetPfStep() to control the *x* and *y* printing step direction.

PARAMETER
None

RETURN VALUE
The *x* step is returned in the MSB, and the *y* step is returned in the LSB of the integer result.

RETURN VALUE
None

2.3.9 Language

Different languages can be selected when using xgGUI graphic objects. Refer to the easyGUI section in this chapter for a description on these.

void xgGuiSetLanguage(int NewLanguage);
Selects the current language. Index zero is the reference language.

PARAMETERS
Language index

RETURN VALUE
None

*char *xgGuiGetTextLanguagePtr (unsigned int StructureNdx, unsigned int Texto, int LanguageIndex);*
Returns pointer to text in a structure. The language of the selected text can be freely selected, no matter what language is active.

PARAMETERS
StructureNdx: graphic structure ID
TextNo: 0 is the first text in the structure, items other then texts are ignored
LanguageIndex: selected language

RETURN VALUE
NULL if no text is found
Pointer to text based on a graphical structure, textnr and selected language.

2.3.10 Bitmaps

DynamicC includes a tool to convert PC based bitmaps in Dynamic C-style libraries. This tool 'fbmcnvtr.exe' can be found in the Utilities directory. It can only be used to convert monochrome bitmaps. The glX… functions only support these monochrome bitmaps.

The easyGUI graphical structures can include bitmaps. These bitmaps are build and maintained with the easyGUI PC application. They can be used on any LCD.

The xgGui_BMP.lib library adds functions to copy the graphic contents of .BMP files to the video frame buffer. BMP files are containers for different types of bitmaps. The included bmp functions only do a limited checking. The #ximport'd files should be prepared on a PC and care should be taken their format matches the used LCD format.

A BMP file format:
- A header: contains the file size
- Extra information: width, height and bits/pixel of the image
- The used color palette
- The bitmap

BMP files normally only support 1, 4, 8 and 24-bit/pixel formats. For X-Graph modules with 1-bit (B/W), 8-bit (most color LCD's) and 24-bit color LCD's (option) this is no problem. The CSTN Color LCD uses a 3-bit/pixel format. There are two possible solutions to this problem. Either 'illegal' BMP files must be created with a non-standard 3-bit/pixel. Or the conversion to 3-bit must be done in the libraries driver. Currently the library does not support this conversion as it would eat up a lot of cpu power.

The color pallet information in the bmp file is not used. The bitmap in the bmp file MUST use the X-Graph pallet. Most PC based graphic applications allow pallet conversions.

For 1-bit/pixel bitmaps the pallet information is not important as only black or white can be chosen.
For 8-bit/pixel bitmaps the pallet must be configured in the X-Graph RRR-GGG-BB format. I.e. the 3 MSB's in each byte represent the red value of the pixel, the next 3 bits the green value and the 2 LSB's the blue value. Only the 256 direct available colors can be used, no pallet conversion is available on X-Graph modules.

3-bit/pixel bitmaps represent a special challenge. Each pixel uses 3 bits (a single bit for each major color R-G-B) which can be combined to 8 colors. Each group of 3 bytes can store 8 pixels in the following sequence:
- Byte 0: RGB RGB RG
- Byte 1: B RGB RGB R
- Byte 2: GB RGB RGB

Data should be stored in this sequence in the bitmap file.

The bitmap is stored upside-down in a bmp file (note that this is not the same as mirrored). The lib takes care of this.

The bmp file definition also requires a bitmap file to be 4-byte aligned. This is not an xgGUI requirement.

Clipping is handled correctly. The images are always copied as-is to the video frame buffer. If the LCD is rotated (90deg or 180deg), the images should be stored rotated in the bmp file.

void xgGuiPutBmp(int left, int top, unsigned long bitmap);
Copies the bitmap part from a .BMP file to the video frame buffer. Any portion of a bitmap image outside the LCD display area or clipping area will be clilpped.

For 1-bit/pixel and 3-bit/pixel bitmaps, the X coordinate must be byte aligned and the bitmaps width must be a multiple of 8 pixels.

PARAMETERS:
x is the *x* coordinate (column) of the upper left corner of the bitmap
y is the *y* coordinate (row) of the left top corner of the bitmap
bitmap is a long pointer to a xmem file
EXAMPLE
#ximport "sample.bmp" sample_bmp
xgGuiPutBmp(10, 100, sample_bmp);

RETURN VALUE
None

void xgGuiShowbitmap(char BitmapIndex, int x, int y);
This function can only be used with easyGUI generated bitmap files.

Draws a bitmap as stored in a xgGUI graphical structure.

PARAMETERS
BitmapIndex: index to the bitmap
x is the *x* coordinate (column) of the upper left corner of the bitmap
y is the *y* coordinate (row) of the left top corner of the bitmap

RETURN VALUE
None

void glXPutBitmap(int left, int top, int width, int height, unsigned long bitmap);
This function can only be used with B/W LCD's and DynamicC bitmap files.

Draws bitmap in the specified space. The data for the bitmap are stored in xmem. This function automatically calls glXPutFastmap if the bitmap is byte-aligned (left edge and width are each evenly divisible by 8).
Any portion of a bitmap image or character that is outside the LCD display area will be clipped (note that the clipping area limitation will not apply).

PARAMETERS
left is the upper left corner of bitmap, must be evenly divisible by 8
top is the left top corner of the bitmap
width is the width of the bitmap
height is the height of the bitmap
bitmap is the address of the bitmap in xmem

RETURN VALUE
None

void glXPutFastmap(int left, int top, int width, int height, unsigned long bitmap);
This function can only be used with B/W LCD's and DynamicC bitmap files.

Draws bitmap in the specified space. The data for the bitmap are stored in xmem. This is like glXPutBitmap, except that it's faster. The restriction is that the bitmap must be byte-aligned.
Any portion of a bitmap image or character that is outside the LCD display area will be clipped (note that the clipping area limitation will not apply)..

PARAMETERS
left is the upper left corner of bitmap, must be evenly divisible by 8
top is the left top corner of the bitmap
width is the width of the bitmap
height is the height of the bitmap
bitmap is the address of the bitmap in xmem

RETURN VALUE
None

void glXGetBitmap(int x, int y, int bmWidth, int bmHeight, unsigned long xBm);
This function can only be used with B/W LCD's and DynamicC bitmap files.

Gets a bitmap from the LCD page buffer and stores it in xmem RAM. This function automatically calls glXGetFastmap if the bitmap is byte-aligned (left edge and width are each evenly divisible by 8).

PARAMETERS
x is the *x* coordinate of the left edge of the bitmap (in pixels)
y is the *y* coordinate of the top edge of the bitmap (in pixels)
bmWidth is the width of the bitmap (in pixels)
bmHeight is the height of the bitmap (in pixels)
xBm is the address of the bitmap in xmem RAM

RETURN VALUE
None

void glXGetFastmap(int left, int top, int width, int height, unsigned long xmemptr);
This function can only be used with B/W LCD's and DynamicC bitmap files.

Gets a bitmap from the LCD page buffer and stores it in xmem RAM This is like glXGetBitmap, except that it's faster. The restriction is that the bitmap must be byte-aligned.

PARAMETERS
left is the upper left corner of bitmap, must be evenly divisible by 8
top is the left top corner of the bitmap
width is the width of the bitmap
height is the height of the bitmap
xmemptr is the address of the bitmap in xmem

RETURN VALUE
None

2.3.11 Scrolling

Note: the glLeft1, glRight1, glUp1 and glDown1 graphic.lib functions are not supported. In the original graphic.lib library these functions are only used internally and they are never used in the DynamicC samples.
The functionality is replaced by high-speed assembly language functions in the xgGUI HAL.

void xgGuiHScroll(int left, int top, int cols, int rows, int nPix, unsigned int color);
void glHScroll(int left, int top, int cols, int rows, int nPix);
Scrolls right or left within the defined window by nPix number of pixels. The opposite edge of the scrolled window will be filled in with 'color' pixels (active color for glHScroll function). The window must be byte-aligned for B/W and CSTN Color LCD's.
The scroll area is adjusted to fit the display area or clipping area.

Parameters will be verified for the following for B/W and CSTN Color LCD's:
1. The left and column paramters will be verified that they are evenly divisible by 8. If not, they will be changed to be a value that is a multiple of 8.
2. Parameters will be checked to verify that the scrolling area is valid. The minimum scrolling area is a width of 8 pixels and a height of one row.

PARAMETERS
left is the upper left corner of bitmap, must be evenly divisible by 8 (B/W and CSTN Color LCD's)
top is the left top corner of the bitmap
cols is the number of columns in the window, must be evenly divisible by 8 (B/W and CSTN Color LCD's)
rows is the number of rows in the window
nPix is the number of pixels to scroll within the defined window (negative value to scroll left)
color is fill color

RETURN VALUE
None

void xgGuiVScroll(int left, int top, int cols, int rows, int nPix, unsigned int color);
void glVScroll(int left, int top, int cols, int rows, int nPix);
Scrolls up or down within the defined window by nPix number of pixels. The opposite edge of the scrolled window will be filled in with 'color' pixels (active color for glVScroll function). The window must be byte-aligned for B/W and CSTN Color LCD's.
The scroll area is adjusted to fit the display area or clipping area.

Parameters will be verified for the following for B/W and CSTN Color LCD's:
1. The left and column paramters will be verified that they are evenly divisible by 8. If not, they will be changed to be a value that is a multiple of 8.
2. Parameters will be checked to verify that the scrolling area is valid. The minimum scrolling area is a width of 8 pixels and a height of one row.

PARAMETERS
left is the upper left corner of bitmap, must be evenly divisible by 8
top is the left top corner of the bitmap
cols is the number of columns in the window, must be evenly divisible by 8
rows is the number of rows in the window
nPix is the number of pixels to scroll within the defined window (negative value to scroll up)
color is fill color

RETURN VALUE
None

2.3.12 Clipping

void xgGuiSetClilpping(int x1, int y1, int x2, int y2);
Sets clipping. Drawing can be limited to a rectangular portion of the screen, this routine sets the clipping limits expressed as two corner coordinates. All drawing falling outside the clipping rectangle is ignored. Default for the clipping rectangle is the entire screen.

PARAMETERS
x1 is the *x* coordinate of one corner of the box
y1 is the *y* coordinate of one corner of the box
x2 is the *x* coordinate of the other corner of the box
y2 is the *y* coordinate of the other corner of the box

RETURN VALUE
None

void xgGuiResetClilpping(void);
Resets clipping. Drawing can be limited to a rectangular portion of the screen, this routine resets the clipping limits to the entire screen.

PARAMETERS
None

RETURN VALUE
None

2.3.13 Touchscreen

The following touchscreen functions are not yet supported in the current xgGui.lib release.

void xgGuiTouchAdjustReset(void);
Resets touch coordinate conversion.

PARAMETERS
None

RETURN VALUE
None

void xgGuiTouchAdjustReset(unsigned int XTrue, unsigned int YTrue, unsigned int XMeasured, unsigned int YMeasured);
Sets one coordinate pair for touch coordinate conversion. Must be called two times, once for each of two diagonally opposed corners, or four times, once for each of the corners. The corner positions should be as close as possible to the physical display corners, as precision is lowered when going towards the display center.

PARAMETERS
XTrue is the *x* coordinate of the presumed display point
YTrue is the *y* coordinate of the presumed display point
XMeasured is the *x* coordinate of the touched display point
YMeasured is the *y* coordinate of the touched display point

RETURN VALUE
None

int xgGuiTouchAdjustReset(unsigned int x, unsigned int y);
Returns touch area number corresponding to the supplied coordinates. If no touch area is found at the coordinates -1 is returned. Touch coordinates are converted to display coordinates, if conversion parameters have been set with the xgGuiTouchAdjustSet function.

PARAMETERS
x is the x coordinate of the touched display point
y is the y coordinate of the touched display point

RETURN VALUE
-1: no touch area found
>=: touch area number

2.4 xgGUI Widgets

2.4.1 xgGUI Graphic Objects

The easyGUI PC application generates C source files which contain some C structs. These structs contain descriptor tables for graphical objects. These objects can be shown individual by calling the xgGuiShowScreen() function.

Some other functions are available to handle the behavior of different objects.

2.4.1.1 Graphic Structures

void xgGuiShowScreen(unsigned int StructureNdx, int CursorFieldToShow, char ResetAutoRedraw);
Draws a graphical structure

PARAMETERS
Structure ID
CursorFieldToShow: active cursor field number of GUI_NO_CURSOR if there is no cursor to show
ResetAutoRedraw:

- GUI_NO_RESET_AUTO_REDRAW
- GUI_RESET_AUTO_REDRAW

RETURN VALUE
None

2.4.1.2 Test Pattern

void xgGuiTestPattern(void);
Draws the test pattern used for initial development of the display controller driver.

PARAMETERS
None

RETURN VALUE
None

2.4.1.3 Cursor

void xgGuiCursor_Select(int NewCursorFieldNo);
Makes requested cursor field active, redrawing both current and new cursor field.

PARAMETERS
New cursor filed number

RETURN VALUE
None

char xgGuiCursor_Down(void);
Makes next cursor field active, redrawing both current and new cursor field.

PARAMETERS
None

RETURN VALUE
0 = cursor at end of range
1 = cursor moved

char xgGuiCursor_Up(void);
Makes previous cursor field active, redrawing both current and new cursor field.

PARAMETERS
None

RETURN VALUE
0 = cursor at end of range
1 = cursor moved

2.4.1.4 Scroll List

void xgGuiRedrawScrollList(void);
Redraws scroll list items

PARAMETERS
None

RETURN VALUE
None

char xgGuiScroll_Down(void);
Makes next scroll line active, and scrolls list if needed.

PARAMETERS
None

RETURN VALUE
0 = no change, list already at bottom
1 = active scroll line changed

char xgGuiScroll_Up(void);
Makes previous scroll line active, and scrolls list if needed.

PARAMETERS
None

RETURN VALUE
0 = no change, list already at top
1 = active scroll line changed

int xgGuiScrollLineOffsetY(void);
Returns Y coordinate offset for active scroll line, zero if active line is at top of visible scroll area.

PARAMETERS
None

RETURN VALUE
Y coordinate offset in pixels

char xgGuiScroll_End(void);
Makes last scroll line active, and scrolls list if needed.

PARAMETERS
None

RETURN VALUE
0 = no change, list already at bottom
1 = active scroll line changed

char xgGuiScroll_Home(void);
Makes first scroll line active, and scrolls list if needed.

PARAMETERS
None

RETURN VALUE
0 = no change, list already at top
1 = active scroll line changed

char xgGuiScroll_Home(int NewLine);
Makes specified scroll line active, and scrolls list if needed.

PARAMETERS
Scroll line, 0 is first line

RETURN VALUE
0 = no change, list already at specified line
1 = active scroll line changed

*void xgGuiSetScrollPars(void (*DataFuncPtr)(unsigned int LineIndex), unsigned int NoOfLines, unsigned int ActiveLine);*
Sets parameters for scroll box functions. Should be called immediately *before* xgGuiShowScreen() function call for the structure containing the scroll box.

PARAMETERS
DataFuncPtr: address of function
NoOfLines: Total number of lines in scroll box
ActiveLine: Active scroll line (will be shown inverted), -1 means no bar, just a scrolling window

RETURN VALUE
None

2.4.2 DynamicC Widgets

These DynamicC widgets all use graphic.lib fonts. They are not compatible with any xgGUI fonts.

2.4.2.1 Text Window Functions

*int TextWindowFrame(windowFrame *window, fontInfo *pFont, int x, int y, int winWidth, int winHeight);*
Defines a text-only display window. This function provides a way to display characters within the text window only using character row and column coordinates.
The text window feature provides end-of-line wrapping and clipping after the character in the last column and row is displayed.
NOTE: Be sure to execute the TextWindowFrame function before using any of the text-only functions (TextGotoXY, TextPutChar, TextPrintf, TextCursorLocation).

PARAMETERS
window is a pointer to the window frame
pFont is a pointer to the window frame descriptor
x is the *x* coordinate of where the text window frame is to start
y is the *y* coordinateof where the text window frame is to start
winWidth is the width of the text window frame
winHeight is the height of the text window frame

RETURN VALUE
0 = window frame was successfully created
-1 = *x* coordinate + width has exceeded the display boundary
-2 = *y* coordinate + height has exceeded the display boundary

*void TextCursorLocation(windowFrame *window, int *col, int *row);*
Sets the cursor location on the display of where to display the next character. The display location is based on the height and width of the character to be displayed.
NOTE: Be sure to execute the TextWindowFrame function before using any of the text-only functions (TextGotoXY, TextPutChar, TextPrintf, TextCursorLocation).

PARAMETERS
window is a pointer to the window frame
col is the character column location
row is the character row location

RETURN VALUE
None

*int TextGotoXY(windowFrame *window, int col, int row);*
Gets the current cursor location that was set by one of the graphic text functions.
NOTE: Be sure to execute the TextWindowFrame function before using any of the text-only functions (TextGotoXY, TextPutChar, TextPrintf, TextCursorLocation).

PARAMETERS
window is a pointer to the window frame
col is a pointer to the cursor column variable
row is a pointer to the cursor row variable

RETURN VALUE
lower word = cursor row location
upper word = cursor column location

*void TextPutChar(struct windowFrame *window, char ch);*
Displays a character on the display where the cursor is currently pointing. If any portion of the bitmap character is outside the LCD display area, the character will not to be displayed.
NOTE: Be sure to execute the TextWindowFrame function before using any of the text-only functions (TextGotoXY, TextPutChar, TextPrintf, TextCursorLocation).

PARAMETERS
window is a pointer to the window frame
ch is the character to be displayed on the LCD

RETURN VALUE
None

*void TextPrintf(struct windowFrame *window, char *fmt, ...);*
This function prints a formatted string (much like printf) on the LCD screen. Only printable characters in the font set are printed; escape sequences '\r' and '\n' are also

recognized. All other escape sequences will be skipped over. For example, nothing will be displayed for '\b' and 't'.

The text window feature provides end-of-line wrapping and clipping after the character in the last column and row is displayed.

NOTE: Be sure to execute the TextWindowFrame function before using any of the text-only functions (TextGotoXY, TextPutChar, TextPrintf, TextCursorLocation).

PARAMETERS
window is a pointer to the window frame
fmt is a formatted string
... formatted-string conversion parameter(s)
EXAMPLE
TextPrintf(&TextWindow, "Test %d\n", count);

RETURN VALUE
None

*void TextBorderInit(windowFrame *wPtr, int border, char *title);*
Initializes the window frame structure with the border and title information. The TextWindowFrame function must be executed before running this function.

PARAMETERS
WindowFrame is a pointer to the window frame descriptor
border is the border style:
SINGLE_LINE—single-line border around the text window
DOUBLE_LINE—double-line border around the text window
title is a pointer to the title:
1. If a NULL string is detected, then no title is written to the text menu
2. If a string is detected, then it will be written to the top of the text menu box as the centered title

RETURN VALUE
None

*void TextBorder(windowFrame *wPtr);*
Displays the border for a given window frame. The TextBorderInit function must be executed before running this function.
This function will automatically adjust the text window parameters to accommodate the space taken by the text border. This adjustment will only occur once after the TextBorderInit function executes.

PARAMETER
wPtr is a pointer to the window frame descriptor

RETURN VALUE
None

*void TextWinClear(windowFrame *wPtr);*
Clears the entire area within the specified text window.

PARAMETER
wPtr is a pointer to the window frame descriptor

RETURN VALUE
None

*int TextMaxChars(windowFrame *wPtr);*
Returns the maximum number of characters that can be displayed within the text window. The TextWindowFrame function must be executed before running this function.

PARAMETER
wPtr is a pointer to the window frame descriptor

RETURN VALUE
The maximum number of characters that can be displayed within the text window.

2.4.2.2 Graphic Menu's

*int glMenuInit(windowMenu *menu, fontInfo *pFont, int border, int shadow, char **menu_options, char* title, maxOptDisplayed);*
Initializes a menu structure with the required parameters to automatically build and display a text menu when the glMenu function is executed.

PARAMETERS
menu is a windowMenu descriptor pointer
pFont is a fontInfo descriptor pointer
border describes the menu border options:
0 = NO_BORDER, no border drawn
1 = SINGLE_LINE, a single-line border around the text menu
2 = DOUBLE_LINE, a double-line border around the text menu
shadow describes the menu shadow options:
0 = NO_SHADOW, no shadowing provided
1 = SHADOWING, shadowing is provided on the menu
menu_options is a pointer to the list of menu options
title is the menu title
ASCII string = title
null string = no title
maxOptDisplayed indicates the maximum number of options to be displayed by the menu:
-1 = forces all options to be displayed
>0 = menu box will only display the number of options indicated, which will require the user to use the scroll keys to bring an option into the menu box view area for the selection

RETURN VALUE
None.

*int glMenu(windowMenu *mPtr, int *state, int x, int y);*
Displays a menu on the LCD display and get the menu options from the user.

NOTE: This function will display an error message on the LCD if the menu width or height exceeds the LCD display boundaries.

PARAMETERS
mPtr is a pointer to structure that contains the information for the menu
state is a pointer to the menu control parameter. The state parameters are as follows:
0 = MENU_INIT, initialize and display menu
1 = MENU_NO_CHANGE, return to selected option, no changes to menu or highlightbar.
2 = MENU_REFRESH, display the last image of the menu, including the location of the highlight bar.
x is the *x* coordinate of where the text menu is to start
y is the *y* coordinate of where the text menu is to start

RETURN VALUE
0 = no option is selected
>0 = option the user has selected
-1 = menu has exceeded LCD screen width
-2 = menu has exceeded LCD screen height

*void glRefreshMenu(windowMenu *mPtr);*
Refreshes the menu indicated by the WindowMenu pointer.

PARAMETER
mPtr is a windowMenu descriptor pointer

RETURN VALUE
None.

*void glMenuClear(windowMenu *mPtr);*
Clears the menu indicated by the WindowMenu descriptor pointer.

PARAMETER
mPtr is a windowMenu descriptor pointer

RETURN VALUE
None.

2.4.3 X-Graph Widgets

More info will be included in the next manual release.

3 easyGUI

easyGUI let's you create a complete graphical user interface on your PC, without the need for downloading and trial-and-error sessions. Once the GUI is designed it generates C source files. easyGUI supports graphical widgets, multiple fonts and multiple languages.

The C files generated by easyGUI are ANSI C compatible, but not 100% compatible with DynamicC. Some conversions are required, listed below.

More info will follow in the next manual release.

4 xgGUI Design Tools

The X-Graph graphical design tool is currently under development.

Warranty

DELGEN warrants that the product delivered hereunder shall conform to the applicable DELGEN datasheet or mutually agreed upon specifications and shall be free from defects in material and workmanship under normal use and service for a period of 1 year from the applicable date of invoice. Products which are "samples", "design verification units", and/or "prototypes" are sold "AS IS," "WITH ALL FAULTS," and without a warranty.
If, during such warranty period, (1) DELGEN is notified promptly in writing upon discovery of any defect in the goods, including a detailed description of such defect; (2) such goods are returned to DELGEN, DDP DELGEN's facility accompanied by DELGEN's Returned Material Authorization form; and (3) DELGEN's examination of such goods discloses to DELGEN's satisfaction that such goods are defective and such defects are not caused by accident, abuse, misuse, neglect, alteration, improper installation, repair, improper testing, or use contrary to any instructions issued by DELGEN, DELGEN shall (at its sole option) either repair, replace, or credit Buyer the purchase price of such goods.
No goods may be returned to DELGEN without DELGEN's Returned Material Authorization form. Prior to any return of goods by Buyer pursuant to this Section, Buyer shall afford DELGEN the opportunity to inspect such goods at Buyer's location, and any such goods so inspected shall not be returned to DELGEN without its prior written consent.
DELGEN shall return any goods repaired or replaced under this warranty to Buyer transportation prepaid. The performance of this warranty does not extend the warranty period for any goods beyond that period applicable to the goods originally delivered.
The foregoing warranty constitutes DELGEN's exclusive liability, and the exclusive remedy of buyer, for any breach of any warranty or other nonconformity of the goods covered by this agreement. This warranty is exclusive, and in lieu of all other warranties, express, implied, or statutory, including without limitation any warranties of merchantability or fitness for a particular purpose. The sole and exclusive remedy for any breach of this warranty shall be as expressly provided herein.

Limitation on Liability

Notwithstanding anything to the contrary contained herein, DELGEN shall not, under any circumstances, be liable to Buyer or any third parties for consequential, incidental, indirect, exemplary, special, or other damages. DELGEN's total liability shall not exceed the total amount paid by Buyer to DELGEN hereunder. DELGEN shall not under any circumstances be liable for excess costs of reprocurement

Notice to Users

DELGEN PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE-SUPPORT DEVICES OR SYSTEMS UNLESS A SPECIFIC WRITTEN AGREEMENT REGARDING SUCH INTENDED USE IS ENTERED INTO BETWEEN THE CUSTOMER AND DELGEN PRIOR TO USE.

Life-support devices or systems are devices or systems intended for surgical implantation into the body or to sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling and user's manual, can be reasonably expected to result in significant injury.

No complex software or hardware system is perfect. Bugs are always present in a system of any size. In order to prevent danger to life or property, it is the responsibility of the system designer to incorporate redundant protective mechanisms appropriate to the risk involved.

All DELGEN products are 100 percent functionally tested. Additional testing may include visual quality control inspections or mechanical defects analyzer inspections. Specifications are based on characterization of tested sample units rather than testing over temperature and voltage of each unit. DELGEN products may qualify components to operate within a range of parameters that is different from the manufacturer's recommended range. This strategy is believed to be more economical and effective. Additional testing or burn-in of an individual unit is available by special arrangement.

Software License Agreement

Notice to Users
This is a legal agreement between you (an individual or single entity, referred to hereinafter as "you") and DELGEN for the computer software product(s) including any accompanying explanatory written materials (the "Software"). BEFORE INSTALLING, COPYING OR OTHERWISE USING THE SOFTWARE, YOU MUST AGREE TO THE TERMS AND CONDITIONS OF THIS AGREEMENT.
If you agree, you are allowed to use the software. If you do not agree with the terms and conditions of this Agreement, you are not allowed to use the software and must destroy all copies of the software.

DELGEN licenses this software to its customers upon acceptance of all the terms and conditions of this license agreement. Please read the terms carefully before downloading or installing the software.
If you do not accept all the terms, you may not install or use this software, and should contact your sales representative to receive a full refund.
If you have any questions, call +32-475-60.64.33, or write to the DELGEN office at 241, route de Longwy, LU-1941 Luxembourg, GD-Luxembourg.

1. Definitions. "Software" means the accompanying computer programs, data compilation(s), and documentation. "You" means the licensee, and are referred to as "You."

2. Term. The term of the license granted herein shall continue until terminated either (a) by You, for your convenience, by written notice to DELGEN or (b) automatically if a material breach by You is not cured within thirty (30) days of such breach. Immediately upon any termination of this license for any reason, You must return to DELGEN all copies of the Software.

3. License Grant. You are granted non-exclusive rights to install and use the Software on a single computer only; however, if the Software is permanently installed on the hard disk or other storage device of a computer (other than a network server), and one person uses that computer more than 80% of the time, then that person may also use the Software on a portable or home computer. You may not install the Software on a network or transmit the Software electronically from one computer to another or over a network. You may copy the Software for archival purposes, provided that any copy must contain the original Software's proprietary notices in unaltered form.

4. Restrictions. You may not: (i) rent, lease, sublicense, loan, timeshare, or permit others to use the Software, except as expressly provided above; (ii) modify or translate the Software; (iii) reverse engineer, decompile, or disassemble the Software, except to the extent this restriction is expressly prohibited by applicable law; (iv) except as permitted by Section 5 below, create a derivative work based on the Software or merge the Software with another product; (vi) copy the Software, except that a reasonable number of copies may be made for archival purposes; or (vii) remove or obscure any proprietary rights notices or labels.

5. Transfers. You may not transfer or assign, in any manner, including by operation of law, the Software or any rights under this Agreement without the prior written consent of DELGEN, which consent shall not be unreasonably withheld. A condition to any transfer or assignment shall be that the recipient agrees to the terms of this Agreement. Any attempted transfer or assignment in violation of this provision shall be null and void.

6. Ownership. DELGEN and its suppliers own the Software and all intellectual property rights embodied therein, including patents, copyrights and valuable trade secrets embodied in the Software's design and coding methodology. The Software is protected by

EC and United States patents, copyright and trade secret laws and international treaty provisions.

Change List

1.0

Initial release