Contents

Product Overview

These sealed and rugged displays have 3 illuminated keys and a 60mm x 33mm screen.

- 128 x 64 dot graphic display or character display with black characters on white background
- Illuminated keys under software control (on / off / flashing)
- Screen only version available if keys not required
- Extreme version available with higher environmental spec.

Install into a panel in a ¼ DIN cutout or from the rear of the panel using the fixing kit (order separately)

Connect to host via mini USB. The display uses an HID-compliant device interface to communicate with the host

A host application must be written to send content to the display, using the display control functions.
These functions are all listed in the API and the use of these is illustrated with code examples.
Download the following from www.storm-interface.com/downloads :-

- PC based Configuration Utility
- Object Libraries for Windows (XP onwards) & Linux (Ubuntu)
- API Source Code (Contact sales@storm-interface.com for source code requests)

Product Range

		Character Display	Graphic Display
Screen with 3 Keys	**Industrial**	USB 3 key 4x20 char display IP54, 0°C to 60°C Impact 5J. Vibration& Shock IEC721-5M3 **Part Number 5103-000**	USB 3 key graphic display IP54, 0°C to 60°C Impact 5J. Vibration& Shock IEC721-5M3 **Part Number 5103-100**
	Extreme	USB 3 key 4x20 char display IP65,-20°C to 70°C Impact 10J. Vibration& Shock IEC721-6M3 **Part Number 5103-010**	USB 3 key graphic displayIP65, -20°C to 70°C Impact 10J. Vibration& Shock IEC721-6M3 **Part Number 5103-110**
Screen only	**Industrial**	USB 4x20 char display IP54, 0°C to 60°C Impact 5J. Vibration& Shock IEC721-5M3 **Part Number 5100-000**	USB graphic display IP54, 0°C to 60°C Impact 5J. Vibration& Shock IEC721-5M3 **Part Number 5100-100**
	Extreme	USB 4x20 char display IP65, -20°C to 70°C Impact 10J. Vibration& Shock IEC721-6M3 **Part Number 5100-010**	USB graphic display IP65, -20°C to 70°C Impact 10J. Vibration& Shock IEC721-6M3 **Part Number 5100-110**
Please note that if ordering from broadline distribution there will be an additional suffix at the end of the part number. This is for distributor labelling purposes only.			
Accessories		Fixing Kit with panel clips, fixings, underpanel gasket, silicone seal **Part Number 5100-FK0** USB Cable 1m, USB A to 90 degree USB mini-B **Part Number 4500-01**	
Downloads		Configuration Utility / Object Libraries for Windows and Linux / Source Code 3D CAD Models Panel Cutout Details **Download from www.storm-interface.com/downloads.** **Contact sales@storm-interface.com for source code requests**	

Functions

- The USB Display uses a USB HID-Compliant device interface to communicate with the host.
- The graphic LCD is 128 pixels by 64 pixels, with backlight, contrast level, and white on black capability.
- 3 Illuminated keys are under software control – on, off and flashing.
- The character LCD has three fixed fonts included
  - o 6 by 8, this will give 8 lines by 20 characters , and
  - o 6 by 16, this will give 4 lines by 20 characters.

    !"#$%&'()*+,-.\0123456789:;<=>?@
    ABCDEFGHIJKLMNOPQRSTUVWXYZ[/]^_'abcdefghijklmnopqstuvwxyz{|}~

  - o 26 by 64, this allows for 4 characters to be displayed 0123456789 , . : ° ±

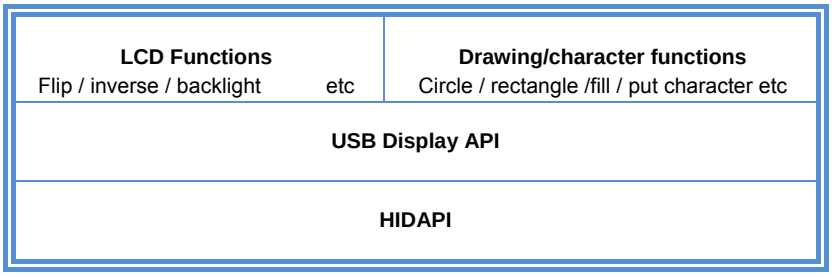
- Four user definable Icons (up to 128 By 64) and any one of them can be setup as a splash screen.
- A host utility will be supplied to configure the unit, including downloading of the Icons.
- Field upgradeable via the utility.
- The host API allows access to following functions:

Set Pixel	Write Character	Write Character String
Draw Circle	Fill Circle	Draw rectangle
Fill Rectangle	Draw bitmap directly to LCD	Load Icons
Draw bargraph.	Draw line.	

- Each button when pressed will output a fixed key code.
- The Icons can be designed using Microsoft Paint™.
- The utility will allow the user to preview the Icon before loading to the USB display.

The USB display uses USB for communicating with the host. It also includes an HID-datapipe back-channel. One of the advantages of using this implementation using only HID interfaces is that no drivers are required on host system.

Basic architecture of the USB display :



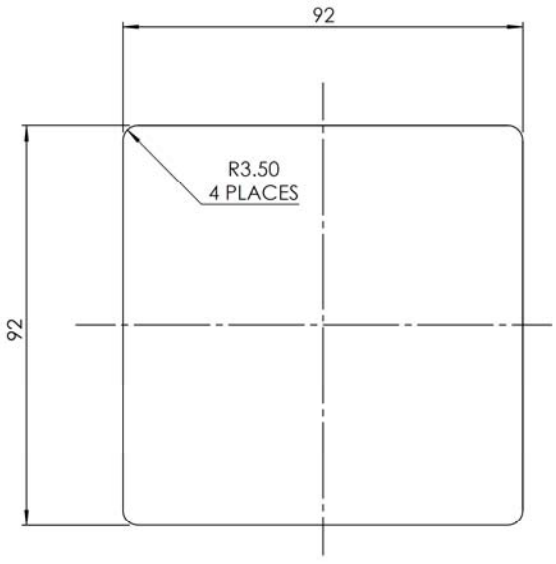
Ratings & Performance

Overall Dimensions	102mm x 102mm x 32mm	
Packed Dims	125mm x 110 mm x 40mm, 203grams (Screen only version is193 grams)	
Connection	mini-USB socket (locking type)	
Environmental	*Industrial Version*	*Extreme Version*
Operational temperature	0ºC to +60ºC	-20ºC to +70ºC
Vibration/ shock IEC721	5M3	6M3
Impact Rating	5J	10J
Sealing	IP54	IP65
Storage temperature	-20ºC to +70ºC	
Humidity	10% to 90% non-condensing	
Insulation resistance	50Mohms (min)	
Breakdown voltage	500V a.c. (60 secs)	
Operating voltage	5V +/- 5% (USB)	
Operating current	20mA (excluding key illumination current)	
Safety	EU Low Voltage Directive	EN60950
EMC:	Emissions and Immunity:	FCC part 15B Class B EN55022, EN55024
	ESD: Up to +/- 15kV air discharge, +/- 7.5kV contact discharge	
EU RoHS	Compliant	
WEEE Directive	Compliant	

Panel Cutout Drawings

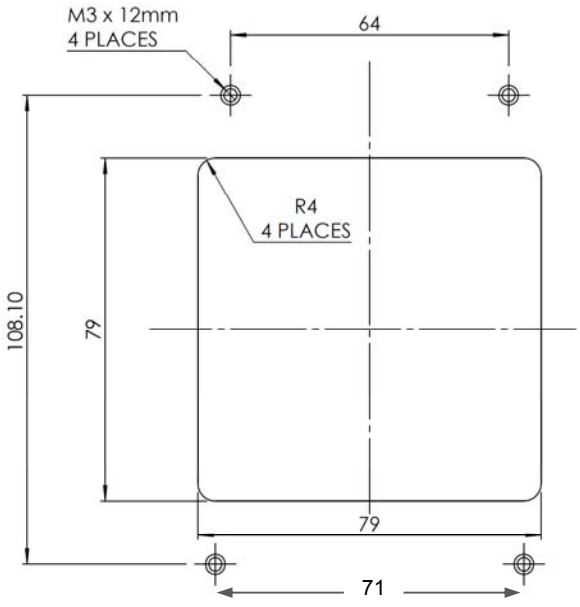
¼ DIN

Recommended panel thickness 1.6mm – 4mm s/s



Underpanel

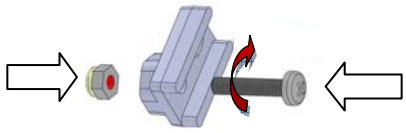
Use M3 x 12mm or equivalent weld studs



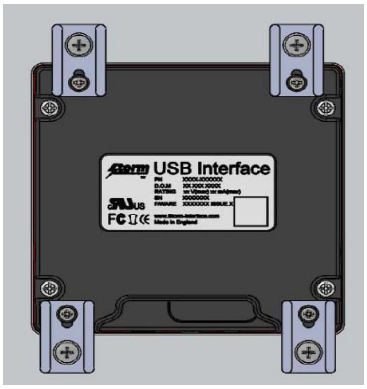
Installation into a ¼ DIN cutout

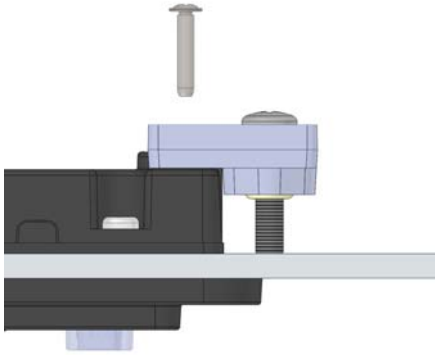
1. Push the white seal into the groove,

and fit the M4 nuts and screws to the brackets.
Allow the screw to protrude to touch the panel.

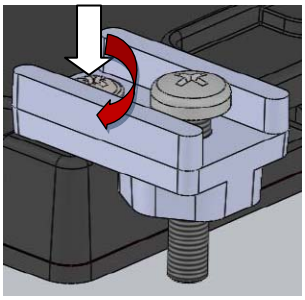


2. Fit the unit into the panel using 4 brackets

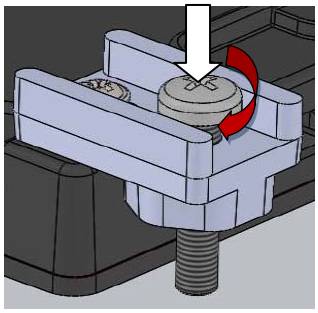




3. Tighten the M3 screws (#1 PZ) to attach each bracket to the rear of the unit.



4. Tighten the M4 screws (#2 PZ) to pull the unit down to the panel surface

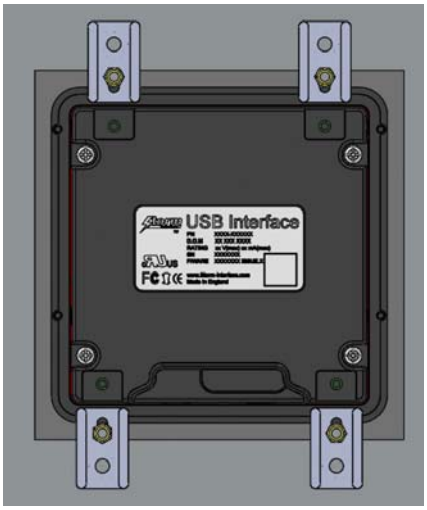


5. Remove the protective film from the screen and connect your USB cable

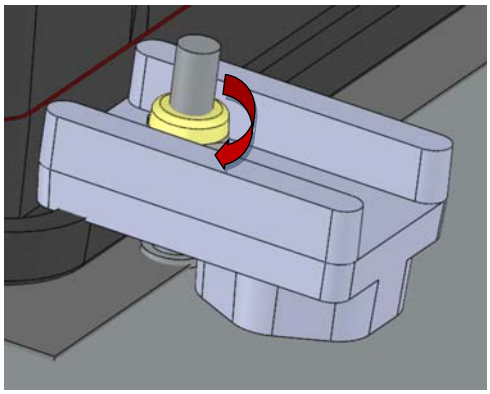
Installation Underpanel

1. Prepare panel with studs M3 x 12mm (or equivalent 6-32 UNC)
2. Place the foam gasket around the display front

3. Fit the unit into the cutout – one bracket goes over each weld stud.



4. Fit a nut over each weld stud and tighten down



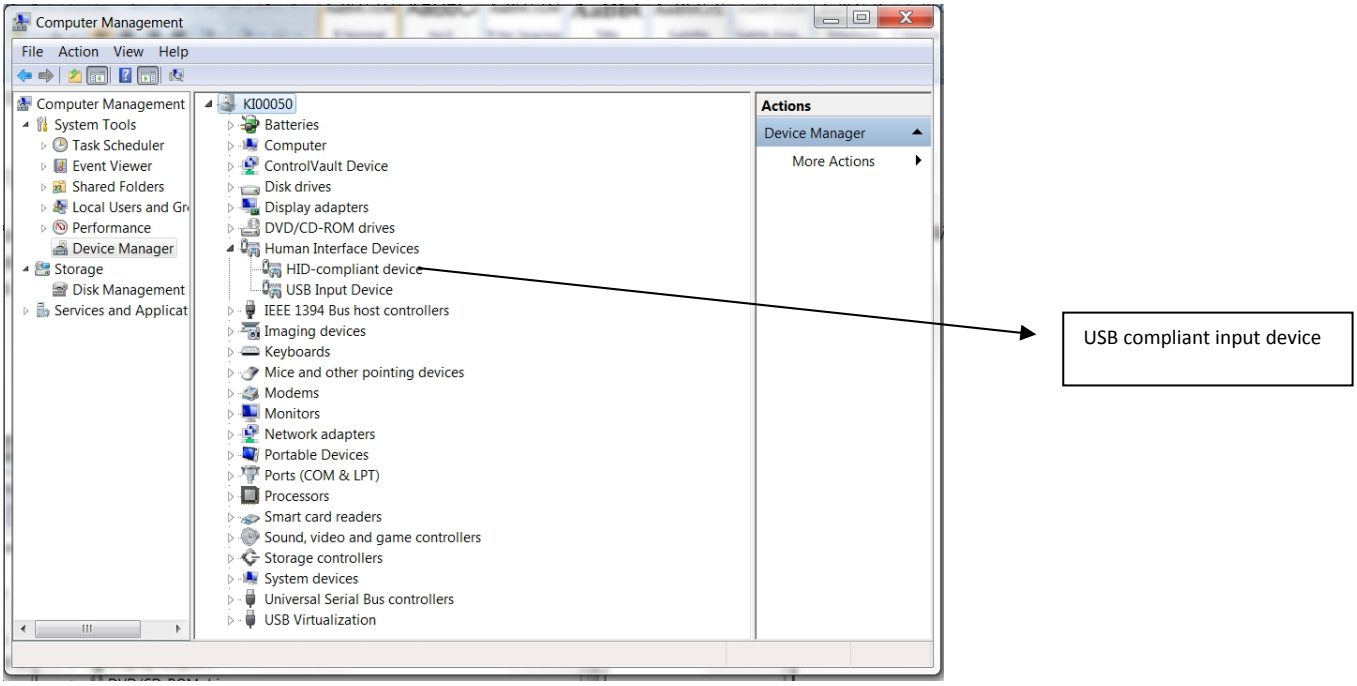
5. Remove the protective film from the screen and connect your USB cable

Using the USB Display

On power up the USB display will perform basic self test and then proceed to display an initial splash screen. The default is the “Storm” logo, customers can customise this splash screen using the software utility, see below for more detailed description.

Once the unit is connected to PC, Windows will detect the USB display as follows :-

When connected to a PC, the USB Display should be detected by the operating system and enumerated without drivers. Windows shows one device in the Device Manager : USB Human Interface Device: Compliant device



The USB Configuration Utility is supplied in order that the user can perform firmware updates, and upload icons to the USB display.

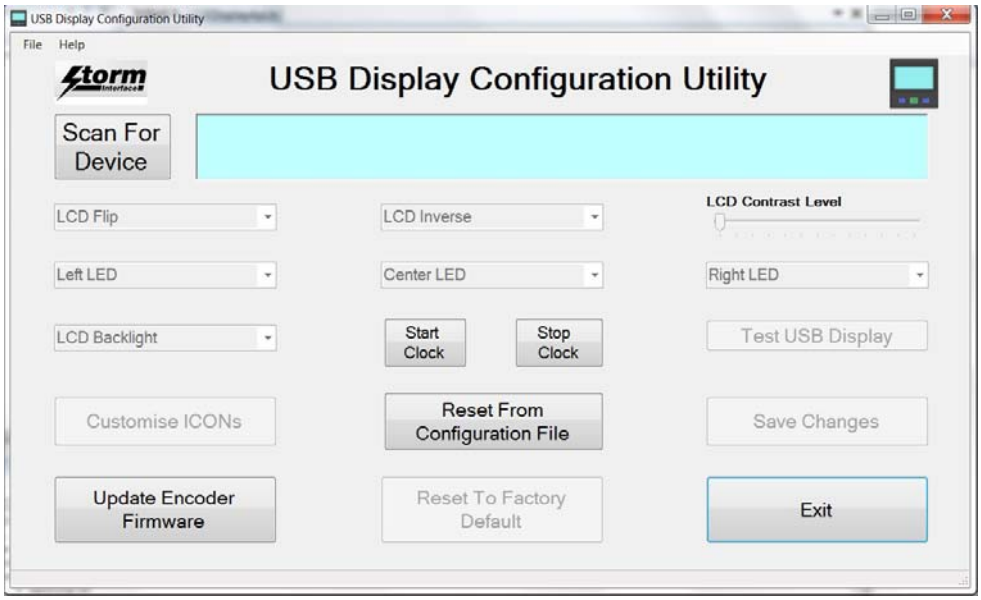
Download the Configuration Utility for free from www.storm-interface.com/downloads

All other functions in the Configuration Utility are also available in the API.

Controlling the USB Display with the Configuration Utility

Launch the application and it will display the following screen:

Before loading the form it initially detects the encoder using the VID/PID and if found it sends a device status message. If all successful then all the buttons are enabled. If not then they will all be disabled except for “Re-Scan” and “Exit”.



Buttons will be disabled/enabled depending on options installed.

Options Installed	Buttons disabled
3 keys + 4/8line character only	Customise ICONs
3 keys + 4/8line character + bitmap	None
No keys + 4/8line character only	All LEDs + Customise ICONs
No keys + 4/8line character + bitmap	All LEDs

- Note: Manufacturer and Product strings are recovered from the USB stack. The USB ID in our product is Vendor ID: 0x2047 Product ID: 0x0922.

  Firmware version is recovered from the encoder.

Once a configuration is selected and accepted by the USB Display then that information is stored in volatile memory of the unit. So if the user has not written to flash (using “Save Changes”) then powering down/up the encoder, that configuration will be lost.

Configuration Utility Functions

LCD Flip
This will set the default value of how the lcd data will be displayed.

LCD Flip – No (Factory Default)
LCD Flip – Yes

LCD Inverse
This will invert the colour of the pixels.

LCD Inverse – No (Factory Default)
LCD Inverse – Yes

LCD Contrast Level
This will set the contrast level of LCD display.

LCD Contrast Level – 0
LCD Contrast Level – 1
.
LCD Contrast Level – 10 (Factory Default)
..
LCD Contrast Level – 20

LCD Backlight
This will set the default value of the backlight.

LCD Backlight – Off (Factory Default)
LCD Backlight – On
LCD Backlight – Flashing

LEDs
If unit has the three keys installed then the LEDs can be controlled via software individually as follows:

Left LED
Left LED – Off
Left LED – On (Factory Default)
Left LED - Flashing

Right LED
Right LED – Off
Right LED – On (Factory Default)
Right LED - Flashing

Centre LED
Centre LED – Off
Centre LED – On (Factory Default)
Centre LED – Flashing

Self Test
This will execute a self test mode on the encoder.

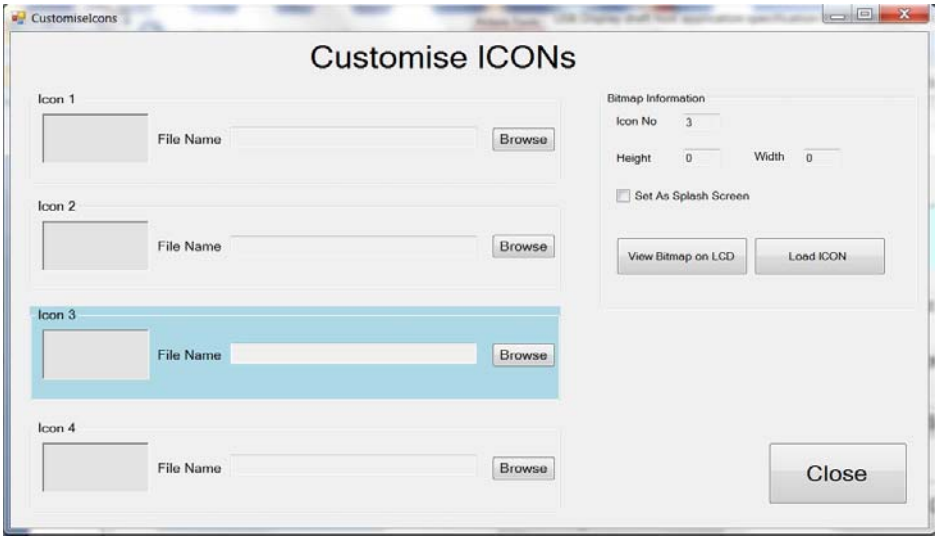
- Show a test pattern on LCD display
- Display circles, rectangle etc.,
- Test keys on unit.

Customise Bitmaps

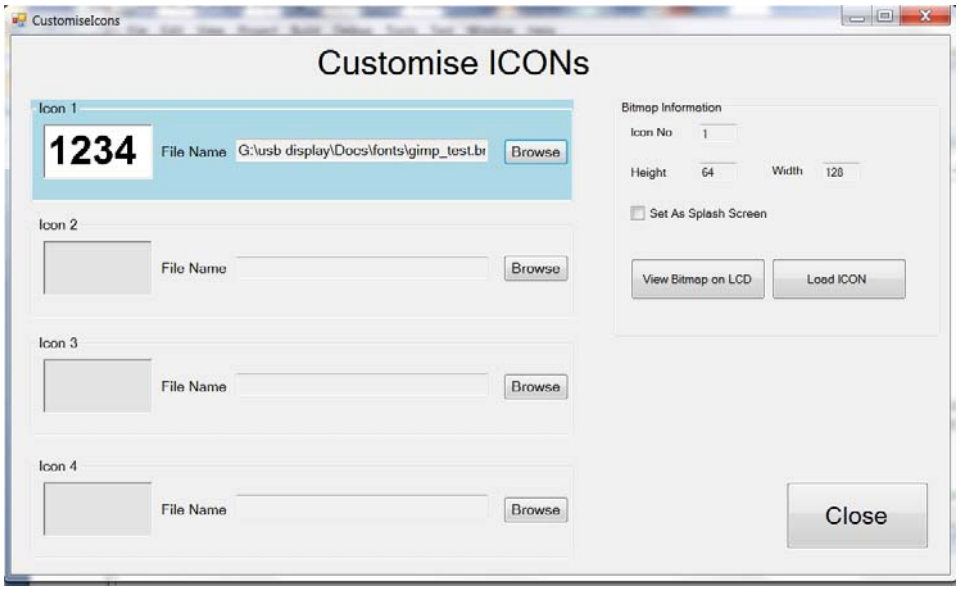
The USB display supports up to four downloadable bitmaps (128 by 64).

The ICONs must be first designed using Paint or any other package that supports the monochrome paint format (i.e. 1bpp format).

Select an Icon position eg Icon 3 and click on “Browse” button. This will open explorer : navigate to your bitmap file and click on “Open”.

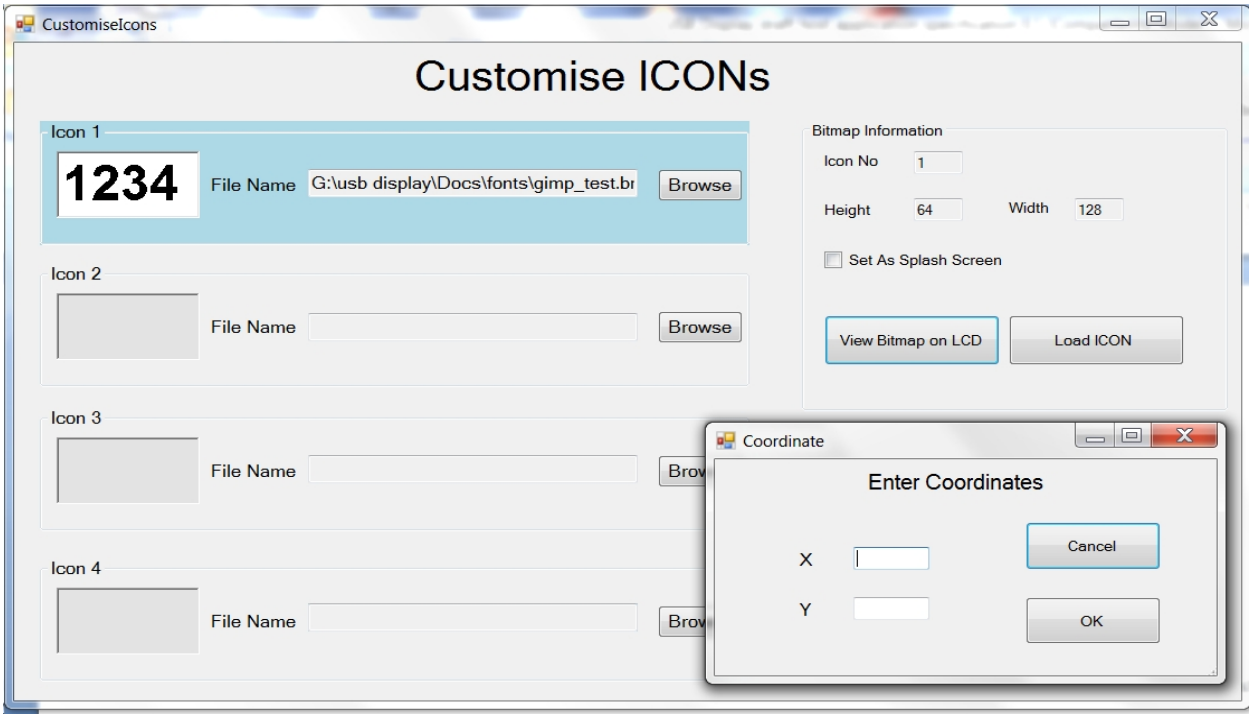


The ICON will be displayed in the icon picture box. The picture box is 128 by 64 bits, so this is exactly what will be displayed on the LCD screen.



On right hand side there is information about the ICON, height, width, icon number and if user wants to use this as the splash icon, when the unit starts up. Only one icon can be set as splash screen.

Now to view the icon on the LCD unit click on “View bitmap on LCD”. It will prompt you to enter X, Y coordinates. The ICON can be placed anywhere on the LCD screen.



Clicking on “OK”, the utility will send the ICON to the USB Display.

Once you are happy with the ICON then you can load the ICON into non volatile memory by clicking on “Load ICON”.

The ICON will be placed in appropriate ICON value on USB display. You can also select one of the icons to be used as a splash screen.

Save Changes

All configurations are written to volatile memory. So if after modifying and the user switches off the encoder then next time the encoder is powered on, it will revert back to previous configuration data. To save the modified data in non volatile memory, click on “Save Changes” button. All the information is also stored in configuration file.

Reset To Factory Default

Clicking on “Factory Default” will set the USB display with values that are preset.

Reset From Configuration File

Clicking on “Reset From Configuration File” will configure the unit from the configuration file from “Save Changes”.

Update Firmware

This option allows the user to update the firmware on the USB display unit.

API Overview

The USB Display API Library is a library program which currently is tested on Windows (from XP and above) and Linux (Ubuntu) platform.

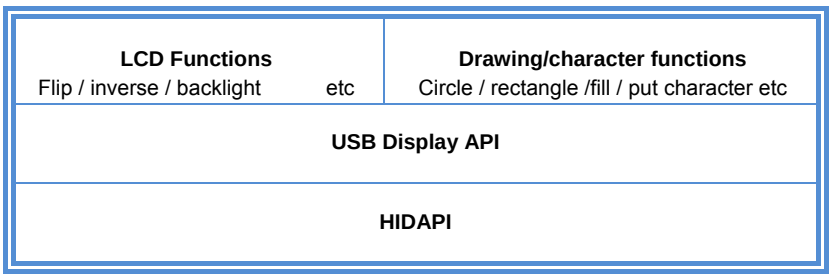
The Library is a middleware program between operating system and host application. The library encapsulates all the communication protocol and exposes a very simple API for host application.

This document is prepared for application developers who will implement a host application for the USB Display.

The USB Display API Library is a middleware application between USB Display Host application and USB Display system.

The USB display uses USB for communicating with the host. It includes an HID-compliant device . One of the advantages of using this implementation, which using only HID interfaces, is that no drivers are required on host system.

The protocol for communicating with host is described fully in the following pages. The basic architecture of the USB display API is shown below.



- USB Display API – The USBDisplayApi library allows for the host application to invoke USB display functions as listed above. The API encapsulates all the communications to USB and provides a simple API for the host application developers.

- HIDAPI - This is a third party library, which allows an application to interface with USB HID-Compliant devices on Windows, Linux, and Mac OS X. While it can be used to communicate with standard HID devices like keyboards, mice, and Joysticks, it is most useful with custom (Vendor-Defined) HID devices. This allows for host software to scan for the device using its VID/PID.

Libraries are provided for both the HIDAPI and USB display interface, so that it can be linked into the users host application. This exposes a well defined API for the host application.

The developer does not need to worry about the communication at low level. You can request source code for the implementation for library so it can be ported to your specific platform. Currently the library has been tested on Windows and Linux (Ubuntu) platform.

The API makes the following functions available to developers see page

Message Types

This is referenced in below functions:

```
enum REQUEST_TYPE{      // message types

DEVICE_STATUS = 1,      ///Device status message
LED_LEFT,               //< set led brightness
LED_RIGHT,              //right led
LED_CENTER,             //Center led
LCD_CLEAR_SCREEN,       //clears LCD display buffer
LCD_DISPLAY_SCREEN,     //displays whats in screen buffer
LCD_INIT,               //inits LCD
LCD_SCREEN_FLIP,        //FLIPS LCD SCREEN
LCD_INVERSE,            //INVERSE LCD
DISPLAY_TEST_PATTERN,   //displays test pattern
LCD_SET_CONTRAST,
LCD_BACKLIGHT,          //controls backlight
RESERVED,
WRITE_DEFAULT,          // Write defaults values from ram to flash
RESET_TO_FACTORY_DEFAULT, // reset the setting to factory default
ENABLE_BSL,             //start downloader
DRAW_LINE,
DRAW_RECTANGLE,
DRAW_CIRCLE,
DRAW_BITMAP_HOST,
PUT_CHAR,
PUT_STRING,
SET_PIXEL,
GET_PIXEL,
SET_BITMAP,
DRAW_BITMAP_FLASH,
RESERVED,
DRAW_BARGRAPH,
KEYPRESS

  }
```

InitialiseStormUSBDevice

This function is used to initialise the USB Display. The usb display is identified by the Product PID and Manufacturer VID. This are assigned to Keymat:

- Vendor ID – 0x2047
- Product ID – 0x0922

On successful finding the USB display the manufacturer_local will be filled with “Storm Interface” and product_local will be filled with “USB Display”. If not successful both of the strings will be filled with “none”

Parameters :

storm_vid	-	Vendor ID
product_pid	-	Product ID
manufacturer_local	-	vendors name will be stored
product_local	-	product name will be stored

Return Value:

True for success
False for failure.

```
///\brief InitializeStormUSBDevice is called at the beginning of the
application to

///Setup the PRODUCT ID (PID) and product vid

///\return false on failure, true on success.

///On failure, call GetErrorCode() to retrieve the error

///

bool  InitializeStormUSBDevice( int storm_vid, int product_pid, std::string
&manufacturer_local, std::string &product_local );
```

SetLEDBACKLIGHTState

This function is used to control the illumination of front panel button LEDs and screen backlight.

Parameters :

led_backlight - Which led to control :

- LED_LEFT
- LED_RIGHT
- LED_CENTRE
- LCD_BACKLIGHT

_Flag - 0 – off, 1 – on or 2 – Flashing

timeToWait - maximum time to wait for command to complete

Return Value:

True for success
False for failure.

```
///\brief SetLEDBACKLIGHTState turns on, off or flash the LED or backlight in
USB DISPLAY

///\param led_backlight          LEFT, RIGHT, CENTRE led or BACKLIGHT

///\param _Flag   '0' to turn it off, '1' to turn it ON, or '2' to flash.

///\return  0 on success, negative error code on failure

//

int SetLEDBACKLIGHTState(int led_backlight, int _Flag,  int _timeToWait );
```

GetDeviceStatus

This function retrieves status information about the USB Display. For example, contrast level, LED status etc. All information is stored in DEVICE_INFO structure.

Parameters :

```
typedef struct
{
      unsigned char           flip_mode;
      unsigned char           inverse_mode;
      unsigned char           contrast_level;
      unsigned char           backlight;
      unsigned char           left_led, right_led, centre_led;
      unsigned char           icon_splash_no;
      std::string             FirmwareName;

} DEVICE_INFO;
```

_deviceInfo	-	DEVICE_INFO sturcture, that will be filled by the function
timeToWait	-	maximum time to wait for command to complete

Return Value:

True for success
False for failure.

```
///\brief GetDeviceStatus Retrieves the USB Display's status information including:
Contrast Level, LED status, Backlight status, Firmware Name.
///The data are returned in  a DEVICE_INFO structure
///\param _deviceInfo is a pointer to a DEVICE_INFO structure that receives information
      retrieved from the USB Display.
///\param _timeToWait is the time in milliseconds to wait for the data to be retrieved.
///\return 0 on success, negative error code on failure
///

Int GetDeviceStatus( DEVICE_INFO *_deviceInfo, int _timeToWait );
```

LCDFunctions (1)

This is an overloaded function. This function is used to control various functionality of the USB Display Screen, e.g. Contrast level, Inverse display etc.

Parameters :

lcdFunction - LCD function supported are:

- LCD_SCREEN_FLIP
- LCD_INVERSE_DISPLAY
- LCD_SET_CONTRAST

Param1 - Following parameters for each of the functions:

LCD_SCREEN_FLIP	0 – no flip, 1 - flip
LCD_INVERSE_DISPLAY	0 – normal, 1 - inverse
LCD_SET_CONTRAST	10 levels provided, 0 to 9

timeToWait - maximum time to wait for command to complete

Return Value:

True for success
False for failure.

```
///\brief LCDFunctions - this functions allows to control the LCD units and incoporates
following functions:
///         LCD Screen Flip   Host To USB Display •  0 • normal, 1 • flips
///         LCD Inverse Display     Host To USB Display •  0 • Normal, 1 • Inverse
///         LCD set Contrast  Host To USB Display •  Sets contrast: 0 • 10 levels
///\Param - lcdFunctions
///\Param - param1 – parameters as stated above in each function.
///\param _timeToWait is the time in milliseconds to wait for function to complete
///
///\return 0 on success, negative error code on failure
///     Possible error codes are:
///         NO_USB_DISPLAY_CONNECTED                        = No usb display is
connected
///
int   LCDFunctions(int lcdFunction, int param1,  int _timeToWait);
```

LCDFunctions (2)

This is an overloaded function. This function is used to control various functionality of the USB Display Screen that takes no parameters.

Parameters :

lcdFunction - LCD function supported are:

- LCD_CLEAR_SCREEN
- LCD_DISPLAY_SCREEN
- LCD_INIT
- DISPLAY_TEST_PATTERN

timeToWait - maximum time to wait for command to complete

Return Value:

True for success
False for failure.

```
///\brief LCDFunctions - overloaded  functions allows to control the LCD units and
incoporates following functions: That does not take parameters
///         LCD Clear Screen    Clears LCD screen buffer
///         LCD Display         Displays LCD Screen Buffer
///         LCD Init            Initializes LCD unit
///         Display Test Pattern Displays a test pattern
///\Param - lcdFunctions
///\param _timeToWait is the time in milliseconds to wait for function to complete
///
///\return 0 on success, negative error code on failure
///     Possible error codes are:
///         NO_USB_DISPLAY_CONNECTED                        = No usb display is
connected
///
int   LCDFunctions(int lcdFunction, int _timeToWait);
```

SetDisplayConfig

This functions allows USB display to save config from ram to flash and also to reset to factory defaults.

Parameters :

configCommand - Config Commands:
- WRITE_DEFAULT – Saves status values to flash
- RESET_TO_FACTORY_DEFAULT – Resets status values

timeToWait - maximum time to wait for command to complete

Return Value:

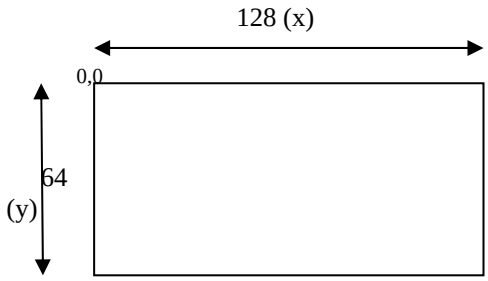
True for success
False for failure.

```
///\brief SetDisplayConfig - this functions allows USB display to save config from ram
to flash and also to reset to factory defaults
///\configCommand -   Write_to_Defaults  - saves parameters from ram to flash
///                   Restore_to_factory_default - restores parameters to preset
factory defaults
///\_timeToWait is the time in milliseconds to wait for function to complete
///
///\return 0 on success, negative error code on failure
///     Possible error codes are:
///         NO_USB_DISPLAY_CONNECTED = No usb display is connected
///


int   SetDisplayConfig(int configCommand, int _timeToWait);
```

Draw Functions

This set of draw functions allows the developer to draw various shapes with a simple API. The screen size is 128 X 64 pixels.



The USB Display has dedicated screen buffer (128 X 64) and it is this screen buffer holds the pixel image, before it is transfered to the LCD display. This allows the developer to first build up a image and then display it using the LCDFunction (SCREEN_DISPLAY) command.

The coordinates are referenced as shown above, with 0,0 (x,y) in top left hand corner.

Character Fonts

The USB Display also has two full set of character fonts (6X8 and 6X16) with following characters:

<SPC>!”#$%&()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_’abcdefghijklmnopqrstuvwxyz{|}~

The above fonts have a border of 4 pixels at beginning and 4 pixels at end of line

The characters fonts are display with x coordinate and line number, as specified below:

FONT 6X8 - line 0 to 7
FONT 6X16 - line 0 to 3

There is also special large font (26X64) but only a limited set of characters:

This font can only be specified with line as 0.

0123456789 - This are all defined as 26X64
:,.-+±° - This are all defined as 8X64

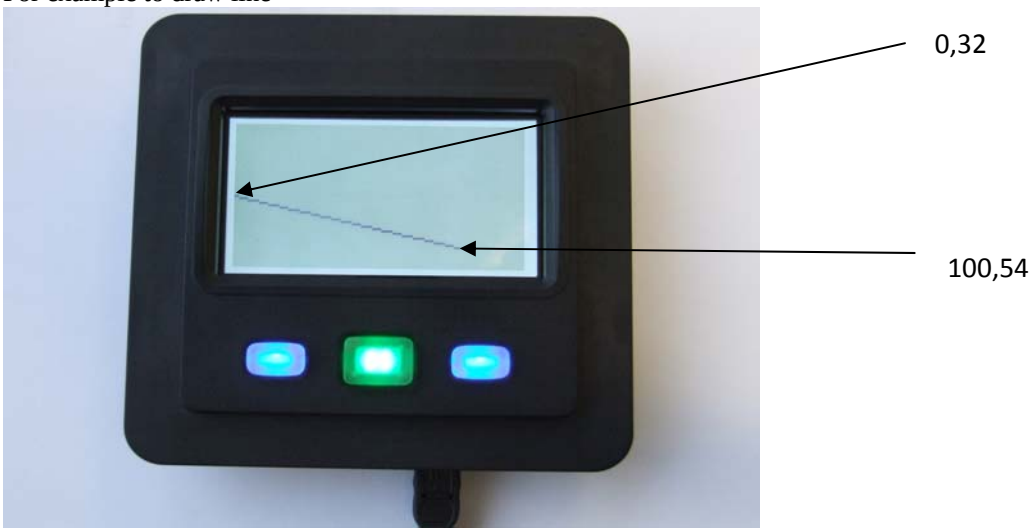
For the large fonts, following characters have been mapped:

~ will display °
! will display ±

DrawLine

This functions draw's a line with the supplied coordinates.

For example to draw line



In above example:

x1	-	0
y1	-	32
x2	-	100
y2	-	54

Paramaters :

x1, y1, x2, y2	-	coordinates as shown above
colour	-	1 – black, 0 - white
timeToWait	-	maximum time to wait for command to complete

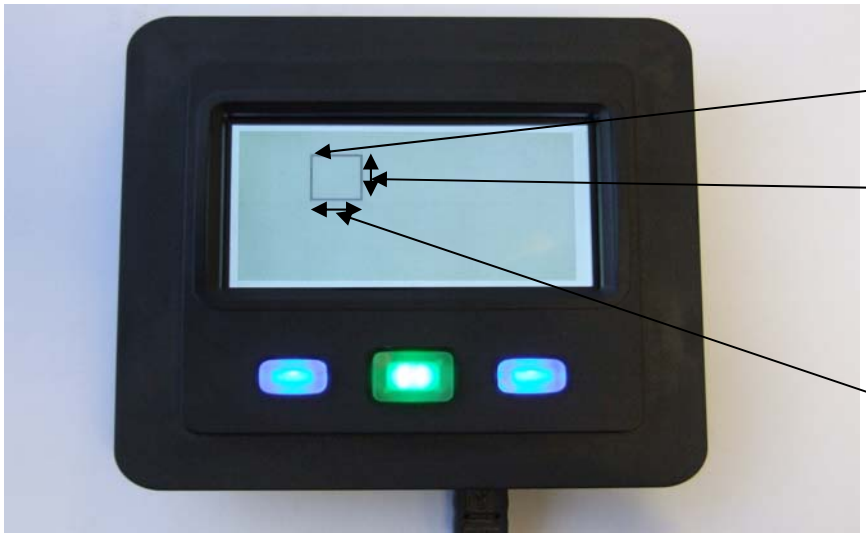
Return Value:

True for success
False for failure.

```
///\brief DrawLine - This functions draws a line between two coordinates
///\Param - x1, y1, x2, y2 coordinates
///\Param - colour - 0 white and 1 black
///\param _timeToWait is the time in milliseconds to wait for function to complete
///
///\return 0 on success, negative error code on failure
///     Possible error codes are:
///         NO_USB_DISPLAY_CONNECTED      = No usb display is connected
///
int DrawLine(unsigned char x1, unsigned char y1, unsigned char x2, unsigned char y2,
unsigned colour,  int _timeToWait);
```

DrawRectangle

This functions draw's a rectangle with the supplied coordinates.
For example to draw rectangle



In above example:

x	-	28
y	-	10
h	-	20
w	-	20
fill	-	0

If fill = 1 then the rectangle will be filled with colour.

Parameters :

x, y	-	coordinates as shown above
colour	-	1 – black, 0 - white
timeToWait	-	maximum time to wait for command to complete

Return Value:

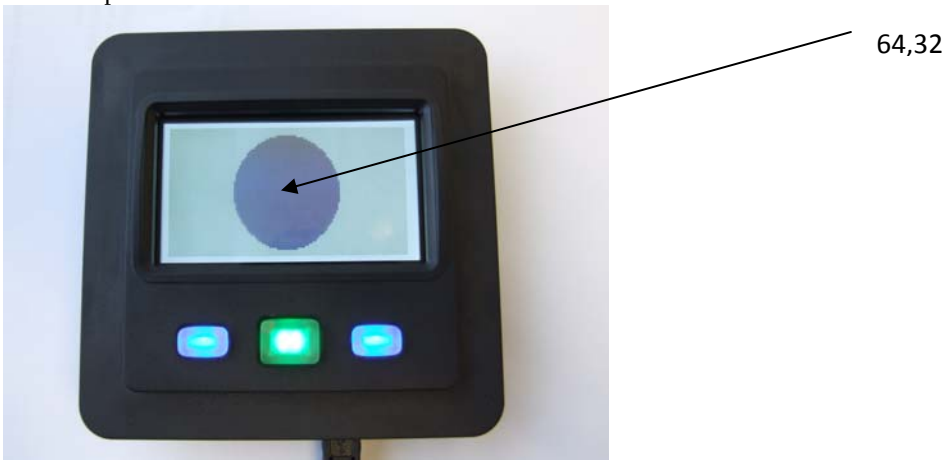
True for success
False for failure.

```
///\brief DrawRectangle - This functions draws a rectangle with supplied coordinates
///\Param - x, y coordinates
///\Param - fill 0 - no fill 1 - fill rectangle
///\Param - w      - width of rectangle
///\Param - h      - height of rectangle
///\Param - colour - 0 white 1 black
///\param _timeToWait is the time in milliseconds to wait for function to complete
///\return 0 on success, negative error code on failure
///     Possible error codes are:
///         NO_USB_DISPLAY_CONNECTED      = No usb display is connected
///
int DrawRectangle(unsigned char fill, unsigned char x, unsigned char y, unsigned char
w, unsigned char h, unsigned colour, int _timeToWait);
```

DrawCircle

This functions draw's a circle with the supplied parameters.

For example to draw circle and filled.



In above example:

The center of the circles coordinates are 64, 32

x	-	64
y	-	32
radius	-	28
fill	-	1
colour	-	1

If fill = 0 then the rectangle will not be filled with colour.

Parameters :

x, y	-	coordinates as shown above
colour	-	1 – black, 0 – white
radius	-	radius of circle in number of pixels.
Fill	-	1 – fill, 0 – no fill
timeToWait	-	maximum time to wait for command to complete

Return Value:

True for success
False for failure.

```
///\brief DrawCircle - This functions draws a circle with supplied coordinates
///\Param - x1, y1 coordinates
///\Param - fill 0 - no fill 1 - fill rectangle
///\Param - radius      - radius of circle
///\Param - colour - 0 white 1 black
///\param _timeToWait is the time in milliseconds to wait for function to complete
///
///\return 0 on success, negative error code on failure
///     Possible error codes are:
///         NO_USB_DISPLAY_CONNECTED                        = No usb display is
connected
///
int DisplayCircle(unsigned char fill, unsigned char x1, unsigned char y1, unsigned char
radius, unsigned char colour, int _timeToWait);
```

DrawBargraph

This functions draws a bargraph with the supplied parameters. The bargraph can be drawn in vertical or horizontal direction.

For example to draw bargraph, which shows two bargraph, one horizontal and one vertical.
The vertical shows with scale set to on (which gives 10 equal scales) and horizontal with no scaling.

In above example:

	Vertical Bargraph	Horizontal Bargraph
Direction	0	1
X	10	50
Y	4	4
W	50	30
H	30	50
Colour	1	1
Percentage Fill	20	66
Scale	1	0

Parameters :

Direction	-	0 – vertical, 1 – horizontal
x, y	-	coordinates as shown above
w	-	width
h	-	height
colour	-	1 – black, 0 – white
percentageFill	-	Total percentage of rectangle to fill with colour.
Scale	-	1 – insert scaling, 0 – no scaling
timeToWait	-	maximum time to wait for command to complete

Return Value:

True for success
False for failure.

DrawChar

This functions draws a single character at supplied coordinates

In above example, display single character ‘Z’ at coordinates x – 63, line – 3 and font used FONT6X8

Parameters :

X	-	0 to 127
Line	-	0 to 7 (FONT6X8) and 0 to 3 (FONT6X16) and 0 (FONT26X64)
fontSelected	-	FONT6X8, FONT6X16 or FONT26X64
colour	-	1 – black, 0 – white
character	-	character to be displayed
timeToWait	-	maximum time to wait for command to complete

Return Value:

True for success
False for failure.

```
///\brief DisplayChar - This functions draws supplied ascii character and specified
fonts at coordinates
///\Param - x (0 to 127)
///\Param - line (0 to 7)
///\Param - colour - 0 white 1 black
///\Param - character      - character string
///\Param - font_selected - Two full fonts (6 X 8, 6 X 16 and limited font 26 X 64)
///\param _timeToWait is the time in milliseconds to wait for function to complete
///
///\return 0 on success, negative error code on failure
///     Possible error codes are:
///         NO_USB_DISPLAY_CONNECTED                        = No usb display is
connected
int DisplayChar(unsigned char x, unsigned char line, unsigned char colour, char
character, int font_selected, int _timeToWait);
```

DrawString

This function draws a string of characters. The USB Display will autowrap the string to next line if more than 20 characters are on a single line or on a carriage return.

In above example, displays a mixture of fonts line 0 and 1 is using FONT6X8 and line 4 and 7 using FONT6X16.

Note: the line spacing of fonts for FONT6X16 will be left to the developer.

Parameters :

X	-	0 to 127
Line	-	0 to 7 (FONT6X8 and FONT6X16) and 0 (FONT26X64)
fontSelected	-	FONT6X8, FONT6X16 or FONT26X64
colour	-	1 – black, 0 – white
charString	-	Character string.
timeToWait	-	maximum time to wait for command to complete

Return Value:

True for success
False for failure.

```
///\brief DisplaysTRING - This functions DRAWS  supplied string of characters and
specified fonts at coordinates
///\Param - x (0 to 127)
///\Param - line (0 to 7)
///\Param - colour - 0 white 1 black
///\Param - charString      - character string
///\Param - font_selected - Two full fonts (6 X 8, 6 X 16 and limited font 26 X 64)
///\param _timeToWait is the time in milliseconds to wait for function to complete
///
///\return 0 on success, negative error code on failure
///     Possible error codes are:
///         NO_USB_DISPLAY_CONNECTED                        = No usb display is
connected
int DisplayString(unsigned char x, unsigned char line, unsigned char colour, char
*charString, int font_selected, int _timeToWait);
```

SetPixel

This functions sets a pixel on/off at supplied coordinates.

In above example, displays one pixel at x=63, y=31.

Parameters :

x, y	-	coordinates
colour	-	1 – black, 0 – white
timeToWait	-	maximum time to wait for command to complete

Return Value:

True for success
False for failure.

```
///\brief SetPixel - This functions set a pixel at specified coordinates
///\Param - x (1 to 128)
///\Param - y (1 to 64)
///\Param - colour - 0 white 1 black
///\param _timeToWait is the time in milliseconds to wait for function to complete
///
///\return 0 on success, negative error code on failure
///     Possible error codes are:
///         NO_USB_DISPLAY_CONNECTED                        = No usb display is
connected
int SetPixel(unsigned char x, unsigned char y, unsigned char colour, int _timeToWait);
```

Bitmaps

The API has 3 functions that deal with the bitmap.

- DrawBitMapFromHost
- LoadBitMap
- DrawIconFromFlash

Background

The bitmap image cannot exceed 128 X 64, below is the process for converting from bitmap (MicroSoft Paint – monochrome bitmap (1 bitmap per pixel) format) to the Storm USB display format. This data is then used in the above bitmap commands. Please note: The Configuration Utility allows the user to load/Display/set as splash icon an already created bitmap (1bpp MS Paint format). The utility converts from bitmap to Storm USB Display and loads the data to the display.

The bitmap data for the USB display is formated with following criteria, for a 128 X 64 bit display, the screen buffer is of size 1024bytes. The screen buffer is direct representation of the LCD display and represented as follows:

Screen Buffer

Byte 00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 (16 bytes represents 128 bit horizontal)
17 18 19 20 21 22 23.....
.
.
.
.
.
.....126 127

The pixels in each byte is represented as follows:

Bit 7
6
5
4
3
2
1
0

So to enable pixel at position (0, 0), bit 7 of byte 0 will be set to 1.

There are various free utilities available to help convert to this format. Please contact Storm for further information.

DrawBitMapFromHost

This function populates the screen buffer with the passed in converted bitmap data. Then use DISPLAY_SCREEN to display the bitmap.

In above example, a “Storm” bitmap is loaded from the host application.

This is a blocking function, that is the **DrawBitMapFromHost** function will not return control until all of the bitmap has been loaded to the USB Display.

Parameters :

x, y	-	coordinates of top left corner of bitmap
h,w	-	height and width of bitmap
colour	-	1 – black, 0 – white
nbytes	-	number of bytes in bitmap
iconPtr	-	pointer to the bitmap
timeToWait	-	maximum time to wait for command to complete

Return Value:

True for success
False for failure.

```
///\brief DrawBitMapFromHost - This functions loads the bitmap from iconPtr to the LCD
screen memory.
///\brief The bitmap must be created using MS paint and stored as monochrome 1 bpp.
///           Note: maximum number of bytes in bitmap must not exceed 1024 bytes.
///\Param - x (0 to 127)
///\Param - y (0 to 63)
///\Param - w - width of the icon in pixel
///\Param - h - height of icon in pixel
///\Param - colour - 0 white 1 black
///\Param - nbytes - number of bytes in bitmap
///\Param - iconPtr - pointer to start of bitmap
///
///\return 0 on success, negative error code on failure
///     Possible error codes are:
///         NO_USB_DISPLAY_CONNECTED                        = No usb display is
connected
int DrawBitMapFromHost(unsigned char  x, unsigned char  y, unsigned char  w, unsigned
char  h, unsigned char color, char *iconPtr, int nbytes, int _timeToWait);
```

LoadBitMap

This functions loads the passed in icon data from host to the USB Display at specified icon location in flash. It also allow it to be set as the splash screen on a reboot.

This is a blocking function, that is the **LoadBitMap** function will not return control until all of the bitmap has been loaded to the USB Display.

Parameters :

h,w	-	height and width of bitmap
İcon_location	-	4 icons can be stored (0 to 3)
İcon_set_as_splash_screen	-	0 – donot set as splash, 1 – set as splash
nbytes	-	number of bytes in bitmap
iconPtr	-	pointer to the bitmap
timeToWait	-	maximum time to wait for command to complete

Return Value:

True for success
False for failure.

```
///\brief LoadBitMap - This functions loads the bitmap from iconPtr to the ICON memory
in flash.
///\brief The bitmap must be in the USB Display format.
///\brief The icon can also set up as splash screen.
//            Note: maximum number of bytes in bitmap must not exceed 1024 bytes.
///\Param - w - width of the icon in pixel
///\Param - h - height of icon in pixel
///\Param - icon_location - (0 - 3) icon position to be written in flash
///\Param - icon_set_as_splash_screen - true - set icon as splash false - do not set as
splash screen
///\Param - nbytes - number of bytes in bitmap
///\Param - iconPtr - pointer to start of bitmap
///\Param - _timeToWait
      ///
      ///\return 0 on success, negative error code on failure
      ///     Possible error codes are:
      ///         NO_USB_DISPLAY_CONNECTED                        = No usb display is
connected

int LoadBitMap(unsigned char  w, unsigned char  h, int icon_location, bool
icon_set_as_splash_screen, char *iconPtr, int nbytes, int _timeToWait);
```

DrawIconFromFlash

This functions populates the screen buffer with the passed in data from the flash. Then use DISPLAY_SCREEN to display the bitmap.

In above example, a icon 0 from flash is loaded from the host application.

This is a blocking function, that is the **DrawIconFromFlash** function will not return control until all of the bitmap has been loaded to the USB Display.

Parameters :

x, y	-	coordinates
iconLocation	-	icon location in flash (0 to 3)
timeToWait	-	maximum time to wait for command to complete

Return Value:

True for success
False for failure.

```
///\brief DrawIconFromFlash - This functions draws stored icon in flash.
///           Note: maximum number of bytes in bitmap must not exceed 1024 bytes.
//\Param - x, y - coordinates
///\Param - icon_location - (0 - 3) icon position to be written in flash
///
///\return 0 on success, negative error code on failure
///     Possible error codes are:
///         NO_USB_DISPLAY_CONNECTED                        = No usb display is
connected

int DrawIconFromFlash(unsigned char  x, unsigned char  y, int icon_location, int
_timeToWait);
```

RetrieveByteFromBuffer

This function retrieves bytes from the buffer a key press value. The following fixed values are passed to host:

```
	enum KEYPRESSED_VALUE
	{
		RIGHT_KEY_CODE				=	0x4F,
		LEFT_KEY_CODE				=	0x50,
		CENTRE_KEY_CODE				=	0x58
	};


///\brief RetrieveByteFromBuffer gets the next available keystroke character from the
input buffer.
///This retrieves the keystrokes from the keypad that have been received.

///	Following values are received for key presses:
//	enum KEYPRESSED_VALUE
//	{
//		RIGHT_KEY_CODE				=	0x4F,
//		LEFT_KEY_CODE				=	0x50,
//		CENTRE_KEY_CODE				=	0x58
//	};
///\return Positive number if valid keystroke, negative number if error occurred.
	///	Possible error codes are:
	///		NO_DATA_AVAILABLE	= There are no keystrokes to retrieve
	///		COULDNT_LOCK_MUTEX		= Internal error - retry
	///		NO_KEYPAD_CONNECTED		= No keypad is connected so cannot
retrieve info
	///
int		RetrieveByteFromBuffer( void );
```

Example Code

Below is an example code on how to use the USB Display API.

On request this source code can be downloaded. The following files are included :

- Visual Studio Project – TestApi
- TestApi.c - Source Code to test the UBDisplayApi
- Header files - All header files for above
- Debug - Debug Folder with USBDisplayApi.lib and hidapi.lib
- Release - Release Folder with USBDisplayApi.lib and hidapi.lib

The workspace also contains project settings for Eclipse under Ubuntu (Linux).

The version of Eclipse used is the Indigo version and currently the Linux version uses SDL library.

testAPI - demonstration project that includes and shows how to use the 'USBDisplayApi.lib' to communicate with the USB Display.

USBDisplayApi is based on the HIDAPI library which is a multi-platform library which allows an application to interface with USB HID-Class devices on Windows, Linux, and Mac OS X. The HIDAPI is encapsulated within the USBDisplayApi.lib and the developer should not be concerned with the usage of this library.

testAPI directory contains the project. The 'debug' and 'release' subdirectories of the project contain pre-built executables that are immediately usable for testing.
Also, this directory contains Visual Studio 9 project and solution that will build these executables directly.

The includes pre-built executables should demonstrate useage of the USB Display API.
This program simply demonstrate most of the API like draw circle, draw rectangle, draw string etc.
It also displays the front panel key presses.

```
//============================================================================
// Name        : testAPI.cpp
// Author      : prakash
// Version     :
// Copyright   : Storm Interface Ltd, 2013 **all rights reserved**
// Description : USB Display Example Code – Initialiase API
//============================================================================

#include <iostream>
#include <stdio.h>
#include "USBDisplayApi.h"
using namespace std;
#define STORM_VID           0x2047
#define USB_DISPLAY_PID   0x0922

//this are external files that contains icons that are already converted to USB //display format.
extern unsigned char icon0[];
extern unsigned char icon1[];
extern unsigned char icon2[];
extern unsigned char icon3[];
enum LCD_STATE
{
LCD_FLIP_STATE,
LCD_INVERSE_STATE,
LCD_BM_TO_HOST_1,
LCD_BM_TO_HOST_2,
LCD_LOAD_BM_1,
LCD_LOAD_BM_2,
LCD_DISP_ICON_1,
LCD_DISP_ICON_2,
LCD_DISPLAY_TEST_PATTERN,
LCD_DRAW_CHAR,
LCD_SET_PIXEL,
LCD_DRAW_LINE,
LCD_DRAW_RECTANGLE,
LCD_DRAW_RECTANGLE_FILL,
LCD_DRAW_CIRCLE,
LCD_DRAW_HORIZONTAL_BG_1,
LCD_DRAW_HORIZONTAL_BG_2,
LCD_DRAW_HORIZONTAL_BG_3,
LCD_DRAW_HORIZONTAL_BG_4,
LCD_DRAW_HORIZONTAL_BG_5,
LCD_DRAW_HORIZONTAL_BG_6,
LCD_DRAW_HORIZONTAL_BG_7,
LCD_DRAW_VERTICLE_BG_1,
LCD_GET_DEVICE_STATUS,
LCD_IDLE
};

#ifndef WIN32

#include <termios.h>
#include <unistd.h>
#include <fcntl.h>
int _kbhit(void)
{  struct termios oldt, newt;
  int ch;
  int oldf;

  tcgetattr(STDIN_FILENO, &oldt);   newt = oldt;   newt.c_lflag &= ~(ICANON | ECHO);  tcsetattr(STDIN_FILENO, TCSANOW, &newt);
  oldf = fcntl(STDIN_FILENO, F_GETFL, 0);        fcntl(STDIN_FILENO, F_SETFL, oldf | O_NONBLOCK);

  ch = getchar();
  tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
  fcntl(STDIN_FILENO, F_SETFL, oldf);
  if(ch != EOF)
  {
    ungetc(ch, stdin);
    return 1;  }

  return 0;}
```

```
#else
#include <conio.h>
#endif

int         main()  {USBDisplayApi         *usbDisplayPtr;  std::string  manufacturer, product;
int         retval;
int         lastReturnValue=0;
long        counter = 0;
int         left_led = 0, center_led=0, right_led=0;
int         lcd_state;
int         screen_flip=0, screen_inverse=0;
int         x2;
int         radius;
int         fill;
int         clear_screen;
int         iconNo;

// First - instatiate our class that communicates with the USB Display
usbDisplayPtr = new USBDisplayApi( );

// Next, initialize it and get it ready to use. STORM_VID and USB_DISPLAY_PID are the ids isssued to storm
//
usbDisplayPtr->InitialiseStormUSBDevice(STORM_VID, USB_DISPLAY_PID, manufacturer, product);


DEVICE_INFO                     deviceInfo;

retval      =           usbDisplayPtr->GetDeviceStatus(&deviceInfo, 3000);

if (retval == 0)
{
printf(" flip mode %d\r\n", deviceInfo.flip_mode);
printf(" Inverse Mode %d\r\n", deviceInfo.inverse_mode);
printf(" backlight Mode %d\r\n", deviceInfo.backlight);
printf(" centre_led Mode %d\r\n", deviceInfo.centre_led);
printf(" contrast_level Mode %d\r\n", deviceInfo.contrast_level);
printf(" icon_splash_no Mode %d\r\n", deviceInfo.icon_splash_no);
printf(" left_led Mode %d\r\n", deviceInfo.left_led);
printf(" right_led Mode %d\r\n", deviceInfo.right_led);
printf(" FirmwareName Mode %s\r\n", deviceInfo.FirmwareName.c_str());
printf(" Counter %d\r\n\r\n", counter++);
}

lcd_state = LCD_FLIP_STATE;
x2 = 1;
radius = 4;
fill = 0;
clear_screen = 1;
iconNo = 0;
//clear s0ree
retval      =           usbDisplayPtr->LCDFunctions(MessageRequest::LCD_CLEAR_SCREEN, 4000);
//set all lights on
retval      =           usbDisplayPtr->SetLEDBACKLIGHTState(MessageRequest::LED_LEFT, 1, 3000);
retval      =           usbDisplayPtr->SetLEDBACKLIGHTState(MessageRequest::LED_RIGHT, 1, 3000);
retval      =           usbDisplayPtr->SetLEDBACKLIGHTState(MessageRequest::LED_CENTER, 1, 3000);
while(!_kbhit())
{
//-------------------------------------------------------------------------
// Check for decoded keypresses
//
retval = usbDisplayPtr->RetrieveByteFromBuffer() ;
// Positive value means a keypress was retrieved
if( USBDisplayApi::SUCCESS <= retval )
{switch(retval)
{case USBDisplayApi::LEFT_KEY_CODE:printf("Left key pressed\r\n");
retval      =           usbDisplayPtr->SetLEDBACKLIGHTState(MessageRequest::LED_LEFT, left_led, 3000);
if (retval == USBDisplayApi::SUCCESS)
{if (left_led) left_led=0; else left_led=1;    }
break;

case USBDisplayApi::RIGHT_KEY_CODE:                             //RIGHT_KEY_CODE:
printf("Right key pressed\r\n");
retval      =           usbDisplayPtr->SetLEDBACKLIGHTState(MessageRequest::LED_RIGHT, right_led, 3000);
```

```
if (retval == USBDisplayApi::SUCCESS)
{if (right_led) right_led=0;	else	right_led=1;}	break;
case USBDisplayApi::CENTRE_KEY_CODE:				//CENTRE_KEY_CODE:
printf("Centre key pressed %ld\r\n", counter++);
retval	=	usbDisplayPtr->SetLEDBACKLIGHTState(MessageRequest::LED_CENTER, center_led, 3000);
if (retval == USBDisplayApi::SUCCESS)
{
if (center_led)
center_led=0;
else
center_led=1;
}
break;
default:
printf("Invalid key pressed\r\n");
break;
}
}
#ifdef WIN32
Sleep(100);
#else
usleep(100*1000);
#endif

//clear screen
if (clear_screen)
{clear_screen = 0;
retval	=	usbDisplayPtr->LCDFunctions(MessageRequest::LCD_CLEAR_SCREEN, 3000);
}


switch(lcd_state)
{
case LCD_FLIP_STATE:
{
retval	=	usbDisplayPtr->LCDFunctions(MessageRequest::LCD_SCREEN_FLIP, screen_flip, 3000);
if (retval == USBDisplayApi::SUCCESS)
{
if (screen_flip)
screen_flip=0;
else
screen_flip=1;
}
lcd_state = LCD_INVERSE_STATE;
break;
}
case LCD_INVERSE_STATE:
{
retval	=	usbDisplayPtr->LCDFunctions(MessageRequest::LCD_INVERSE, screen_inverse, 3000);
if (retval == USBDisplayApi::SUCCESS)
{
if (screen_inverse)
screen_inverse=0;
else
screen_inverse=1;
}
lcd_state = LCD_DISPLAY_TEST_PATTERN;
break;
}
case LCD_DISPLAY_TEST_PATTERN:
{
retval	=	usbDisplayPtr->LCDFunctions(MessageRequest::DISPLAY_TEST_PATTERN, 3000);
if (retval == USBDisplayApi::SUCCESS)
{
lcd_state = LCD_BM_TO_HOST_1;
clear_screen = 1;
}
break;
}
case LCD_BM_TO_HOST_1:
{
retval	=	usbDisplayPtr->DisplayString(0, 1,  1, "Display Bitmap from Host", USBDisplayApi::FONT6X8, 3000);
retval	=	usbDisplayPtr->DisplayString(0, 3,  1, "Please Wait...", USBDisplayApi::FONT6X8, 3000);
if (retval == USBDisplayApi::SUCCESS)
{
```

```
usbDisplayPtr->LCDFunctions(MessageRequest::LCD_DISPLAY_SCREEN, 3000);
lcd_state = LCD_BM_TO_HOST_2;
}
break;
}
case LCD_BM_TO_HOST_2:
{
//		ifstream myfile;
		//first the bmp file needs to be converted into our lcd format

retval	=		usbDisplayPtr->DrawBitMapFromHost(0, 0, 128, 64, 1, (char *)&icon2[0], 1024, 3000);
retval = USBDisplayApi::SUCCESS ;
if (retval == USBDisplayApi::SUCCESS)
{
usbDisplayPtr->LCDFunctions(MessageRequest::LCD_DISPLAY_SCREEN, 3000);
lcd_state = LCD_LOAD_BM_1;
clear_screen = 1;
}
break;
}
case LCD_LOAD_BM_1:
{
retval	=		usbDisplayPtr->DisplayString(0, 1,  1, "Load Bitmap from Host", USBDisplayApi::FONT6X8, 3000);
retval	=		usbDisplayPtr->DisplayString(0, 3,  1, "Please Wait...", USBDisplayApi::FONT6X8, 3000);
if (retval == USBDisplayApi::SUCCESS)
{
usbDisplayPtr->LCDFunctions(MessageRequest::LCD_DISPLAY_SCREEN, 3000);
lcd_state = LCD_LOAD_BM_2;
}
break;
}
case LCD_LOAD_BM_2:
{
//		ifstream myfile;
		//first the bmp file needs to be converted into our lcd format
retval	=		usbDisplayPtr->LoadBitMap(128, 64, 0, 1, (char *)&icon1[0], 1024, 3000);

  /		retval = USBDisplayApi::SUCCESS ;

if (retval == USBDisplayApi::SUCCESS)
{
usbDisplayPtr->LCDFunctions(MessageRequest::LCD_DISPLAY_SCREEN, 3000);
lcd_state = LCD_DISP_ICON_1;
clear_screen = 1;
}
break;
}
case LCD_DISP_ICON_1:
{
//		ifstream myfile;
//first the bmp file needs to be converted into our lcd format
retval	=		usbDisplayPtr->DrawIconFromFlash(0, 0, iconNo, 3000);
if (retval == USBDisplayApi::SUCCESS)
{
iconNo++;
if (iconNo > 3)
iconNo = 0;
usbDisplayPtr->LCDFunctions(MessageRequest::LCD_DISPLAY_SCREEN, 3000);
lcd_state = LCD_DRAW_CHAR;
clear_screen = 1;
}
break;
}
case LCD_DRAW_CHAR:
{
retval	=		usbDisplayPtr->DisplayChar(1, 0, 1, 'A', USBDisplayApi::FONT6X8, 3000);
if (retval == USBDisplayApi::SUCCESS)
{
usbDisplayPtr->LCDFunctions(MessageRequest::LCD_DISPLAY_SCREEN, 3000);
lcd_state = LCD_SET_PIXEL;
clear_screen = 1;

}
break;
}
```

```
case LCD_SET_PIXEL:
{
retval      =           usbDisplayPtr->SetPixel(1, 1, 1, 3000);
if (retval == USBDisplayApi::SUCCESS)
{
usbDisplayPtr->LCDFunctions(MessageRequest::LCD_DISPLAY_SCREEN, 3000);
lcd_state = LCD_DRAW_LINE;
clear_screen = 1;
}
break;
}
case LCD_DRAW_LINE:
{
retval      =           usbDisplayPtr->DrawLine(63, 1, x2, 63, 1, 4000);
retval      =           usbDisplayPtr->DrawLine(63, 63, x2, 1, 1, 4000);
if (retval == USBDisplayApi::SUCCESS)
{
usbDisplayPtr->LCDFunctions(MessageRequest::LCD_DISPLAY_SCREEN, 3000);
x2 += 8;
if (x2 > 127)
{
clear_screen = 1;
lcd_state = LCD_DRAW_RECTANGLE;
x2 = 1;
}
}
break;
}
case LCD_DRAW_RECTANGLE:
{
retval      =           usbDisplayPtr->DrawRectangle(0, 1, 1, 32, 30, 1, 4000);
if (retval == USBDisplayApi::SUCCESS)
{
usbDisplayPtr->LCDFunctions(MessageRequest::LCD_DISPLAY_SCREEN, 3000);
clear_screen = 1;
lcd_state = LCD_DRAW_RECTANGLE_FILL;
}
break;
}
case LCD_DRAW_RECTANGLE_FILL:
{
retval      =           usbDisplayPtr->DrawRectangle(1, 1, 1, 32, 30, 1, 4000);
if (retval == USBDisplayApi::SUCCESS)
{
usbDisplayPtr->LCDFunctions(MessageRequest::LCD_DISPLAY_SCREEN, 3000);
clear_screen = 1;
lcd_state = LCD_DRAW_CIRCLE;
}
break;
}
case LCD_DRAW_CIRCLE:
{
retval      =           usbDisplayPtr->DisplayCircle(fill, 63, 32, radius, 1, 3000);
if (retval == USBDisplayApi::SUCCESS)
{
usbDisplayPtr->LCDFunctions(MessageRequest::LCD_DISPLAY_SCREEN, 3000);
radius += 4 ;
if (radius > 32)
{
radius = 4;
if (fill)
fill =0;
else
fill =1;
lcd_state = LCD_DRAW_HORIZONTAL_BG_1;
clear_screen = 1;

}
}
break;
}
case LCD_DRAW_HORIZONTAL_BG_1:
{

//vertical draw bargraph
```

```
usbDisplayPtr->DisplayString(10, 6,  1, "Temp", USBDisplayApi::FONT6X8, 3000);
usbDisplayPtr->DisplayString(80, 6,  1, "Vol", USBDisplayApi::FONT6X8, 3000);
usbDisplayPtr->DisplayBargraph(0, 10, 1, 40, 20, 1, 20 ,1, 3000);
retval = usbDisplayPtr->DisplayBargraph(0, 80, 1, 40, 20, 1, 80, 1, 3000);
if (retval == USBDisplayApi::SUCCESS)

usbDisplayPtr->LCDFunctions(MessageRequest::LCD_DISPLAY_SCREEN, 3000);
lcd_state = LCD_DRAW_HORIZONTAL_BG_2;
}
break;
}
case LCD_DRAW_HORIZONTAL_BG_2:
{
usbDisplayPtr->DisplayBargraph(0, 10, 1, 40, 20, 1, 10, 1, 3000);
retval = usbDisplayPtr->DisplayBargraph(0, 80, 1, 40, 20, 1, 80, 1, 3000);
if (retval == USBDisplayApi::SUCCESS)
{
usbDisplayPtr->LCDFunctions(MessageRequest::LCD_DISPLAY_SCREEN, 3000);
lcd_state = LCD_DRAW_HORIZONTAL_BG_3;
}
break;
}
case LCD_DRAW_HORIZONTAL_BG_3:
{
usbDisplayPtr->DisplayBargraph(0, 10, 1, 40, 20, 1, 100, 1, 3000);
retval = usbDisplayPtr->DisplayBargraph(0, 80, 1, 40, 20, 1, 15, 1, 3000);
if (retval == USBDisplayApi::SUCCESS)
{
usbDisplayPtr->LCDFunctions(MessageRequest::LCD_DISPLAY_SCREEN, 3000);
lcd_state = LCD_DRAW_HORIZONTAL_BG_4;
}
break;
}
case LCD_DRAW_HORIZONTAL_BG_4:
{
usbDisplayPtr->DisplayBargraph(0, 10, 1, 40, 20, 1, 50, 1, 3000);
retval = usbDisplayPtr->DisplayBargraph(0, 80, 1, 40, 20, 1, 40, 1, 3000);
if (retval == USBDisplayApi::SUCCESS)
{
usbDisplayPtr->LCDFunctions(MessageRequest::LCD_DISPLAY_SCREEN, 3000);
lcd_state = LCD_DRAW_HORIZONTAL_BG_5;
}
break;
}
case LCD_DRAW_HORIZONTAL_BG_5:
{
usbDisplayPtr->DisplayBargraph(0, 10, 1, 40, 20, 1, 70, 1, 3000);
retval = usbDisplayPtr->DisplayBargraph(0, 80, 1, 40, 20, 1, 44, 1, 3000);
if (retval == USBDisplayApi::SUCCESS)
{
usbDisplayPtr->LCDFunctions(MessageRequest::LCD_DISPLAY_SCREEN, 3000);
lcd_state = LCD_DRAW_HORIZONTAL_BG_6;
}
break;
}
case LCD_DRAW_HORIZONTAL_BG_6:
{
retval = usbDisplayPtr->DisplayBargraph(0, 10, 1, 40, 20, 1, 30, 1, 3000);
if (retval == USBDisplayApi::SUCCESS)
{
usbDisplayPtr->LCDFunctions(MessageRequest::LCD_DISPLAY_SCREEN, 3000);
lcd_state = LCD_GET_DEVICE_STATUS;
clear_screen = 1;
}
break;
}
case LCD_DRAW_VERTICLE_BG_1:
{
break;
}

case LCD_GET_DEVICE_STATUS:
{
retval      =           usbDisplayPtr->GetDeviceStatus(&deviceInfo, 3000);
if (retval == 0)
```

```
{
printf(" flip mode %d\r\n", deviceInfo.flip_mode);
printf(" Inverse Mode %d\r\n", deviceInfo.inverse_mode);
printf(" backlight Mode %d\r\n", deviceInfo.backlight);
printf(" centre_led Mode %d\r\n", deviceInfo.centre_led);
printf(" contrast_level Mode %d\r\n", deviceInfo.contrast_level);
printf(" icon_splash_no Mode %d\r\n", deviceInfo.icon_splash_no);
printf(" left_led Mode %d\r\n", deviceInfo.left_led);
printf(" right_led Mode %d\r\n", deviceInfo.right_led);
printf(" FirmwareName Mode %s\r\n", deviceInfo.FirmwareName.c_str());
printf(" Counter %d\r\n\r\n", counter++);
}
lcd_state = LCD_FLIP_STATE;
break;
}
default:
break;
}
}
printf(" Exiting USBDisplayApi_Demo.....\r\n\r\n");
usbDisplayPtr->~USBDisplayApi();
//              delete usbDisplayPtr;
return 0;
}
```

Change History

Engineering Manual	Date	Version	Details
	10 Dec 13	1.0	First Release

Configuration Utility	Date	Version	Details
	10 Dec 13	1.0	First Release

Object Library (Windows)	Date	Version	Details
	10 Dec 13	1.0	First Release

Object Library (Linux)	Date	Version	Details
	10 Dec 13	1.0	First Release

API Source Code	Date	Version	Details
	10 Dec 13	1.0	First Release
Visual Studio Project – TestApi			
TestApi.c - Source Code to test the UBDisplayApi			
Header files -All header files for above Debug - Debug Folder with USBDisplayApi.lib and hidapi.lib Release - Release Folder with USBDisplayApi.lib and hidapi.lib			

Note: This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:
—Reorient or relocate the receiving antenna.
—Increase the separation between the equipment and receiver.
—Connect the equipment into an outlet on a circuit different from that to which the receiver is connected.
—Consult the dealer or an experienced radio/TV technician for help.