



Contents

Product Overview, Range	2
Functions	3
Ratings and Performance, Installation	4
Using the Display	6
Configuration Utility	7
Application Programming Interface (API)	14
Code Examples	36
Change History	42

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 display IP65, -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		Fitting Kit with panel clips, fixings, underpanel gasket, replacement 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
 - 6 by 8, this will give 8 lines by 20 characters , and
 - 6 by 16, this will give 4 lines by 20 characters.

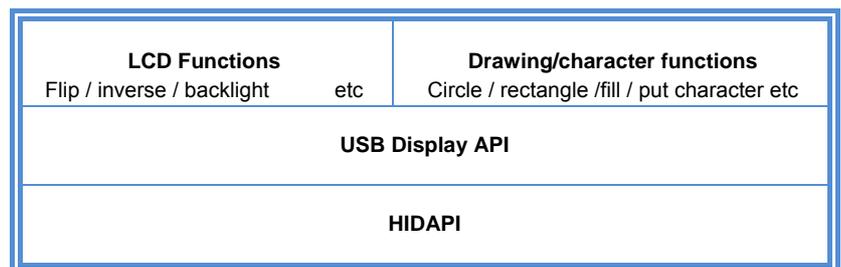
!"#\$%&'()*+,-.0123456789:;<=>?@
 ABCDEFGHIJKLMNOPQRSTUVWXYZ[]^_`abcdefghijklmnopqrstuvwxyz{|}~

- 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 is 193 grams)
Connection	mini-USB socket (locking type)

<i>Environmental</i>	<i>Industrial Version</i>	<i>Extreme Version</i>
Operational temperature	0°C to +60°C	-20°C to +70°C
Vibration/ shock IEC721	5M3	6M3
Impact Rating	5J	10J
Sealing	IP54	IP65

Storage temperature	-20°C to +70°C
Humidity	10% to 90% non-condensing
Insulation resistance	50Mohms (min)
Breakdown voltage	500V a.c. (60 secs)
Operating voltage	5V +/- 5% (USB) – must only be used with SELV circuit.
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	

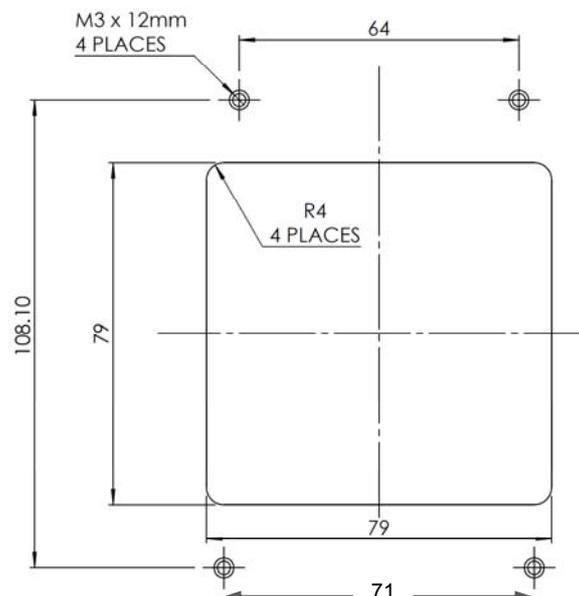
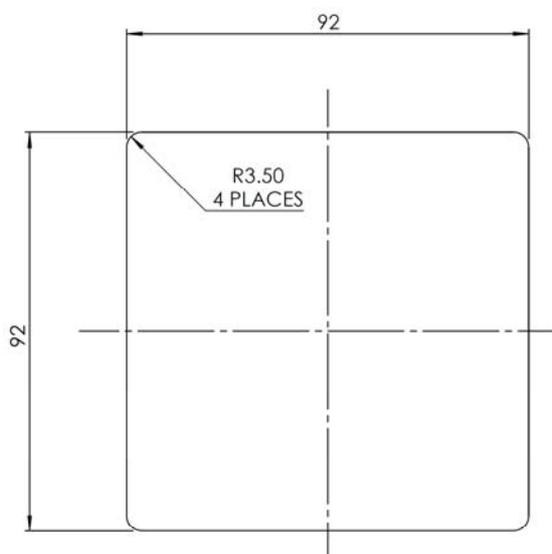
Panel Cutout Drawings

¼ DIN

Underpanel

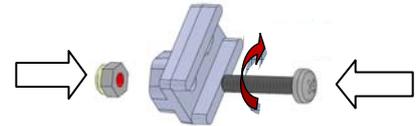
Recommended panel thickness 1.6mm – 4mm s/s

Use M3 x 12mm or equivalent weld studs

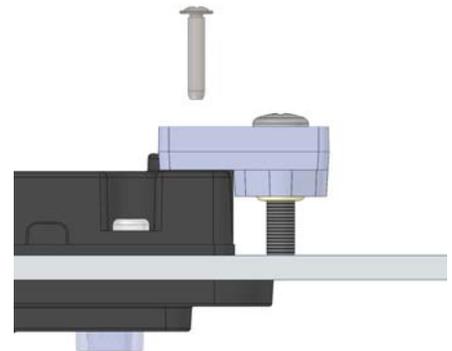


Installation into a ¼ DIN cutout

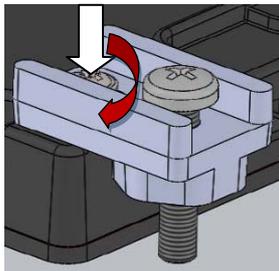
1. Fit the M4 nuts and screws to the brackets. Allow the screw to protrude to touch the panel



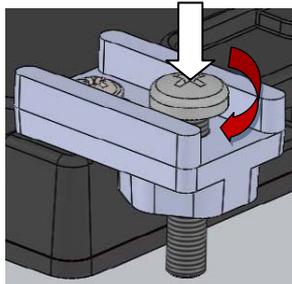
2. Fit the unit into the panel using 4 brackets



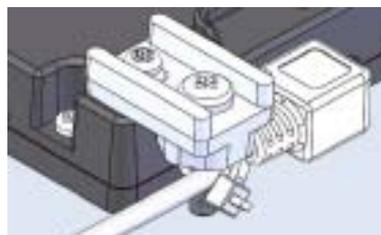
3. Tighten the M3 screws (#1 PZ) to attach each bracket to the rear of the unit.



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
If additional cable securing is required then use a nylon tie-wrap as shown

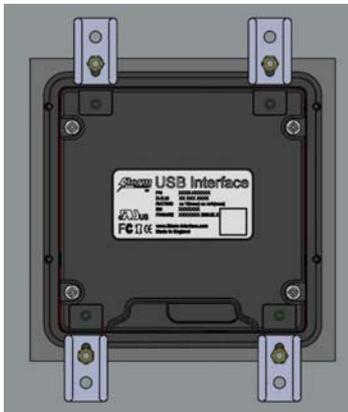


Installation Underpanel

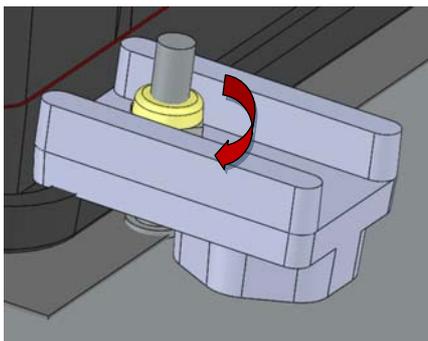
1. Prepare panel with studs M3 x 12mm (or equivalent 6-32 UNC)
2. Place the foam gasket around the display front



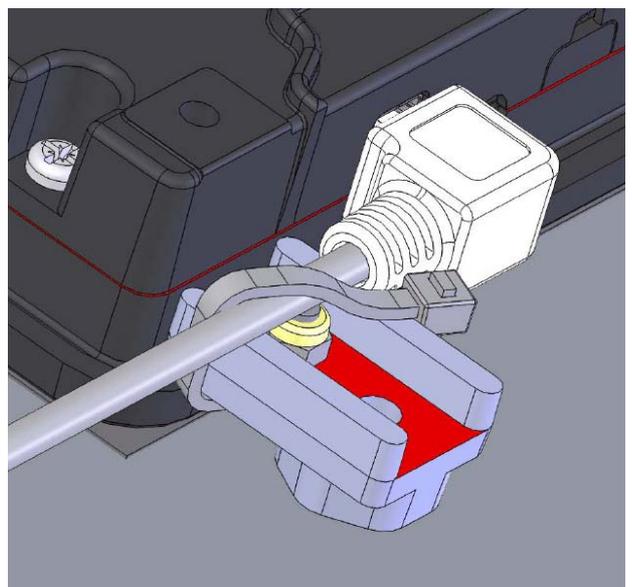
3. Fit the unit into the cutout – one bracket goes over each weld stud.



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



5. Remove the protective film from the screen and connect your USB cable
If additional cable securing is required then use a nylon tie-wrap as shown

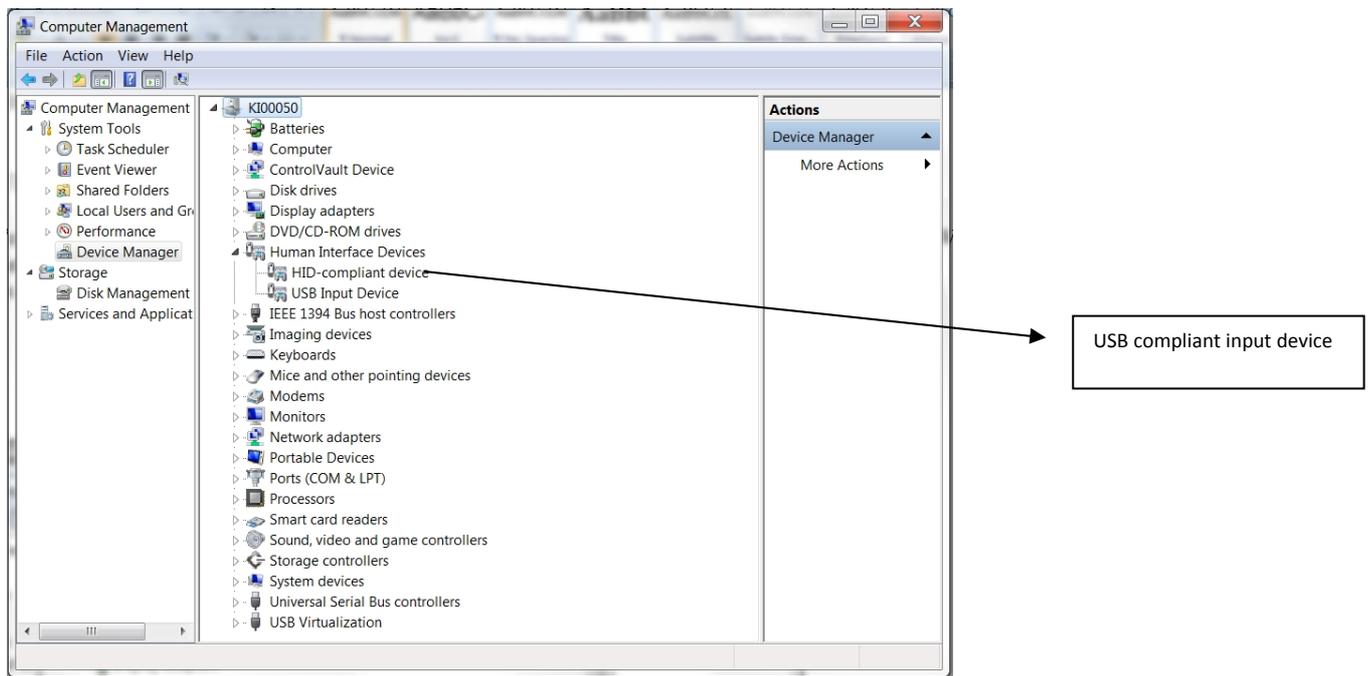


Using the USB Display

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.

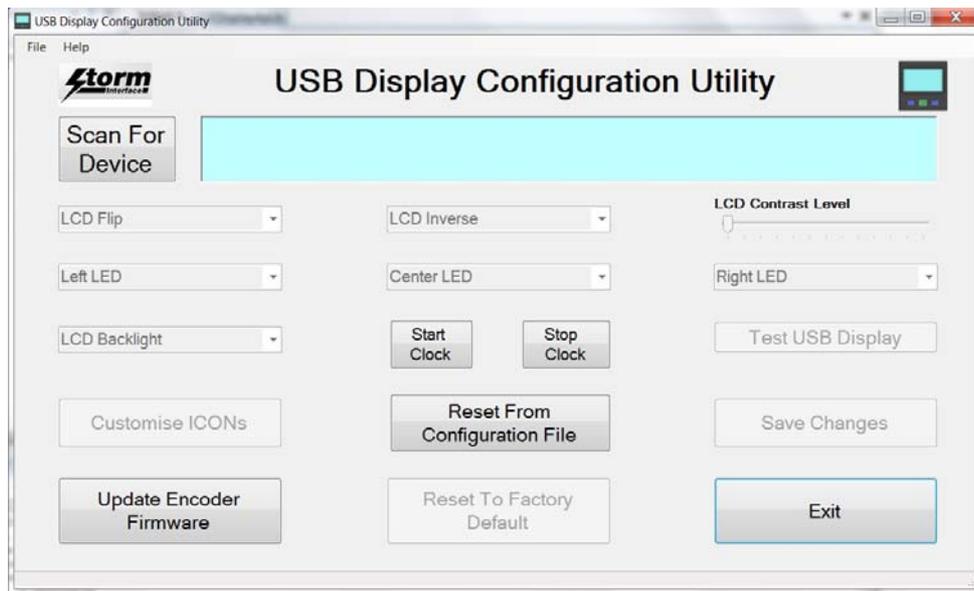
Download the Configuration Utility for free from www.storm-interface.com/downloads

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 (180° rotation)

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 – 0 (Factory Default)

LCD Backlight – Off

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

Test USB Display

This will execute a self test mode on the encoder.

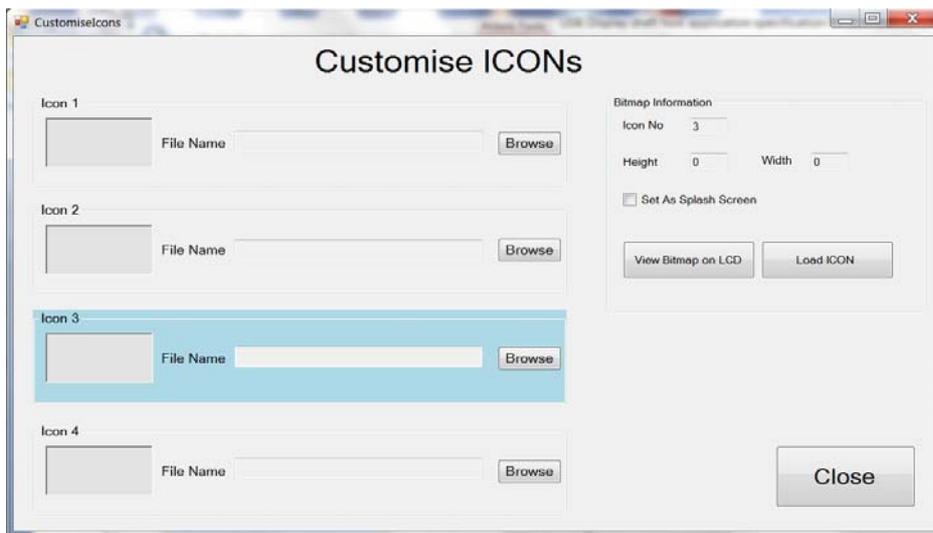
- Show a test pattern on LCD display
- Display circles, rectangle etc.,
- Test keys on unit.

Customise Bitmaps

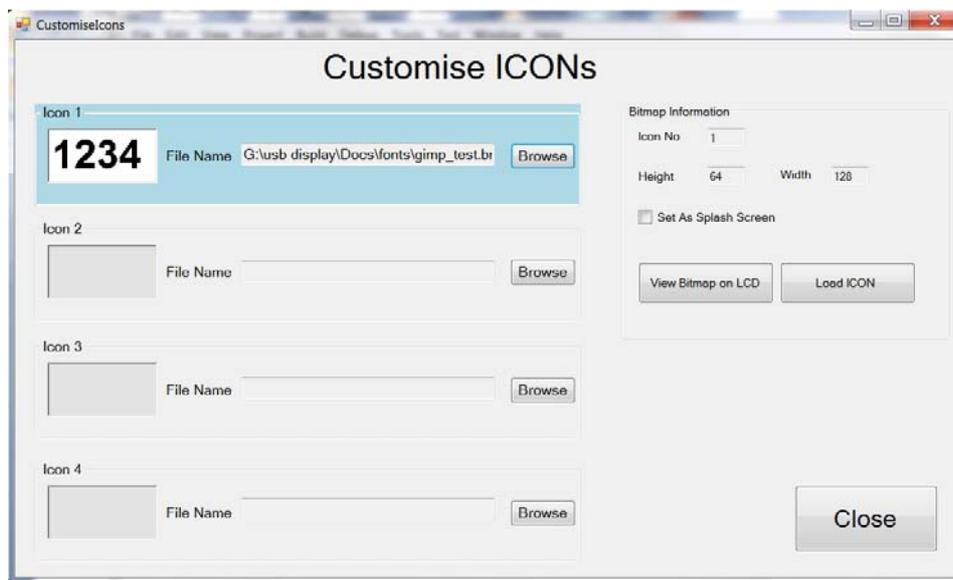
The USB display supports up to four downloadable bitmaps (128 by 64).

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”.

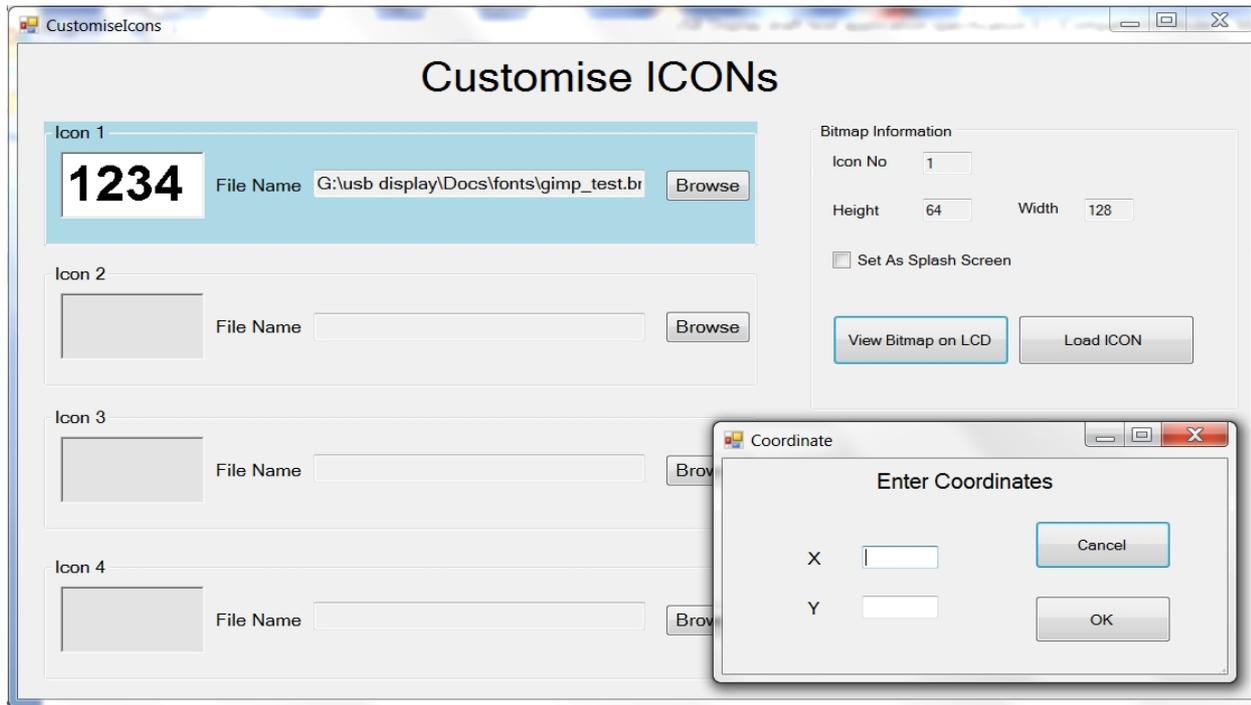


The ICON will be displayed in the icon picture box



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.

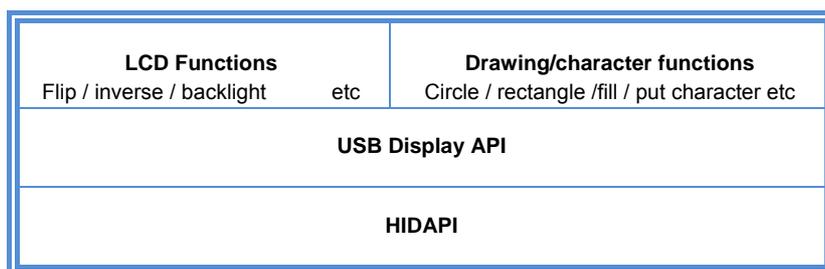
The Library is a middleware program between operating system and host application. The library encapsulates all the communication protocol and exposes a very simple API for host application.

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

All Message Types	14
Bitmaps	31
Character Fonts	21
DisplayChar	30
DisplayString	29
Draw Functions	21
DrawBargraph	25
DrawBitMapFromHost	32
DrawChar	26
DrawCircle	24
DrawIconFromFlash	34
DrawLine	22
DrawRectangle	23
DrawString	27
Example Code	36
GetDeviceStatus	17
InitialiseStormUSBDevice	15
LCDFunctions (1)	18
LCDFunctions (2)	19
LoadBitMap	33
RetrieveByteFromBuffer	35
SetDisplayConfig	20
SetLEDBACKLIGHTState	16
SetPixel	28

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,
DISPLAY_CHAR,
DISPLAY_STRING,
    GET_TEMPERATURE,

}
```

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.
```

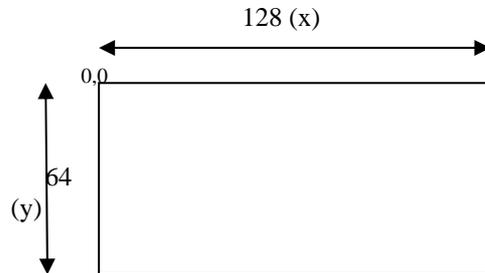
```
///  
//On failure, call GetErrorCode() to retrieve the error
```

```
///  
//
```

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


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 transferred 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;:<=>?@ABCDEFGHIJKLMNPOQRSTUVWXYZ[\\]^_`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 ±

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.



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 formatted 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.

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 - instantiate 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 issued 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_VERTICAL_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	<u>Date</u>	<u>Version</u>	<u>Details</u>
	10 Dec 13	1.0	First Release
	04 Mar 14	1.1	Typographical errors corrected
	14 Oct 14	1.2	Added Display Char and Display String Functions

Configuration Utility	<u>Date</u>	<u>Version</u>	<u>Details</u>
	10 Dec 13	1.0	First Release
	11 March 14	2.0	Production Issue (9v03 Firmware on device)

Object Library (Windows)	<u>Date</u>	<u>Version</u>	<u>Details</u>
	10 Dec 13	1.0	First Release

Object Library (Linux)	<u>Date</u>	<u>Version</u>	<u>Details</u>
	10 Dec 13	1.0	First Release

API Source Code	<u>Date</u>	<u>Version</u>	<u>Details</u>
	10 Dec 13	1.0	First Release
Visual Studio Project – TestApi	14 Oct 14	2.0	Speeded up refresh rate (in conjunction with device firmware 9v04)
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			

This document is provided for use and guidance of engineering personnel engaged in the installation or application of Storm Interface data entry products manufactured by Keymat Technology Ltd. Please be advised that all information, data and illustrations contained within this document remain the exclusive property of Keymat Technology Ltd. and are provided for the express and exclusive use as described above.

This document is not supported by Keymat Technology's engineering change note, revision or reissue system. Data contained within this document is subject to periodic revision, reissue or withdrawal. Whilst every effort is made to ensure the information, data and illustrations are correct at the time of publication, Keymat Technology Ltd. are not responsible for any errors or omissions contained within this document.

This device complies with part 15 of the FCC Rules. Operation is subject to the following two conditions: (1) This device may not cause harmful interference, and (2) this device must accept any interference received, including interference that may cause undesired operation.



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.

No part of this document may be reproduced in any form or by any means or used to make any derivative work (such as translation or adaptation) without written permission from Keymat Technology Ltd.

For more information about Storm Interface and its products, please visit our website at www.storm-interface.com © Copyright Storm Interface. 2013 All rights reserved