API Overview (Host Implementation)

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

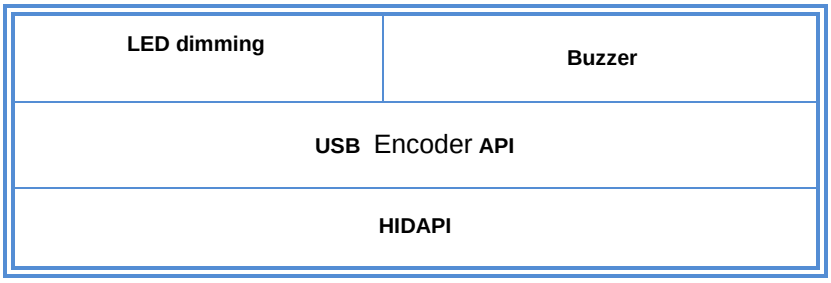
The Library is a middleware program between operating system and host application. The library encapsulates all the communication protocol and exposes a very simple API for host application.

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

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

The USB Encoder 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 Encoder API is shown below.



- USB Encoder API – The USBEncoderApi library allows for the host application to invoke USB Encoder 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 Encoder 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. .

Currently the library has been tested on Windows and Linux (Ubuntu) platform.

API

USB Encoder Device Communications

The encoder uses the ASCII/binary Message format described below.

Every message that is sent from a host should be acknowledged with the control byte ACK (0x06).
A retransmission should be initiated if an NAK (0x15) is received or if nothing is received.
When encoder transmits a key press code to host, it will only send ASCII code. No acknowledgement is required.

Message Format

	Message Field	Type	Length	Description
1	STX	C	1	Control character Start of Text = 0x02
2	Message ID	H	2	Defines the type of message and format of the data field
3	Data Length	H	2	Hexadecimal value represented in ASCII defines the number of bytes in the data field. '00' to 'FF'. Maximum data field size is 256 bytes.
4	Data Field	S	var	In binary format
5	ETX	C	1	Control character ETX = 0x03
6	LRC	C	1	Longitudinal Redundancy Check Digit, calculated on all previous data including STX

Message Format Example – enable the buzzer on key press

	STX	ID	DATA LENGTH	DATA FIELD	ETX	LRC
HEX	0x02	04	1	1	0x03	calculated

The full message is always as per the format above. For clarity only the ID and the Data Field are shown in the message definitions on pages 4 – 14.

Character Types Used

A	Alpha character, 'A'-'Z' and 'a' - 'z'
C	Control character one byte in length.
H	Hexadecimal characters, ‘0’-‘9’, ‘A’-‘F’
N	Numeric character, ‘0’-‘9’
S	Special characters, entire character set 0x00 - 0xFF

Message Definitions

The messages are listed below; each one is fully detailed on the following pages.

ID.	Data	Message	Description
01		Device Status Request	Host To USB Encoder – Output the firmware version and all currently selected parameters
02	lb	LED Brightness	Host To USB Encoder – adjust led brightness. (default: 0)
03		Reserved	
04	bof	Key Press Buzzer On/Off	Host To USB Encoder - Enable/Disable buzzer. (Default: Enable)
05	lt	Load New code table	Host To USB Encoder – Load new code table
06	bp	Change Buzzer Duration	Host To USB Encoder – change the buzzer period duration
07	kt	Keypad Type	Host To USB Encoder – Select layout table 0 – Function key – 4way (default) 1 – Arrow Key - 4 way 2 – Customised 4way, 3 – Telephone – 12way, 4 – Calculator – 12way , 5 – Customised – 12way, 6 – Telephone – 16way, 7 – Calculator – 16way , 8 – Customised – 16way
08	st	Self Test	Host To USB – The encoder start/end a self test
09		Save Configuration	Host To USB Encoder – Encoder writes configuration data from ram to flash.
10		Reset to factory default	Host To USB Encoder – Reset device back to factory default
11		Reserved	
12		Enable BSL	Host To USB Encoder – Sets the encoder to detect the device loader for firmware loading
13	sb	Status Buzzer	Host to USB encoder – Sounds the buzzer for x period. X is passed in value (0 -9)

Error Code

Every response message contains one of the following error codes:

00	No error
01	Command not recognized
02	Command not support at this stage
03	Parameter not supported
04	Hardware fault

Device ID

Following table shows the possible values for the device ID field:

00	Keymat Technology USB Encoder

Device Status (01)

Host sends this message to USB encoder to request the status of the encoder.

USB Encoder Status Response

USB Encoder sends this message to Host in response to the Device Status message.

	Data Field	Type	Length	Description
ec	Error Code	H	2	
ldb	LED Brightness	N	1	Value (0 – 9)
	reserved	N	1	
bof	Buzzer	N	1	0 – OFF, 1 – ON
op	Option	N	1	0x01 – LEDs, 0x02 – Buzzer, Rest is reserved for future use.
bp	Buzzer on period	N	1	Value (0 – 9)
kt	Keypad Type	N	1	0 – Function key – (default) 4 way 1 – Arrow Key 4 way 2 – Customised 4 way 3 – Telephone 12 way 4 – Calculator 12 way 5 – Customised 12 way 6 – Telephone 16 way 7 – Calculator 16 way 8 – Customised 16 way
ktab	Keycode table	H	Up to 32	Layout selected table – data could be for 4 way or 12/16 way
fwv	Firmware Version	A,N	20	Left justified, if Firmware Version is less than 20 then just add enough spaces after the Firmware Version until this field is completed, for instance, “123456” becomes: “123456 “

Host Device		USB Encoder
[01]	⇨	
	⇦	[01][ec][ldb][bof][op][bp][kt][ktab][fwv]

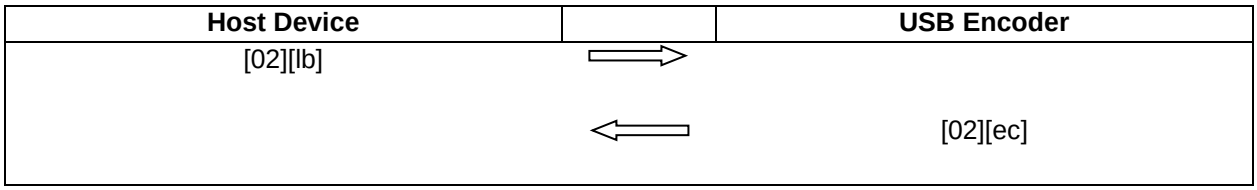
LED Brightness Command (02)

Host sends this message to control brightness of LEDs

	Data Field	Type	Length	Description
lb	LED brightness	N	1	0 - 9

LED Brightness Command Response

	Data Field	Type	Length	Description
ec	Error Code	H	2	



Note: LED brightness of 0 value indicates LEDs are off

LED brightness of 9 value indicates full brightness

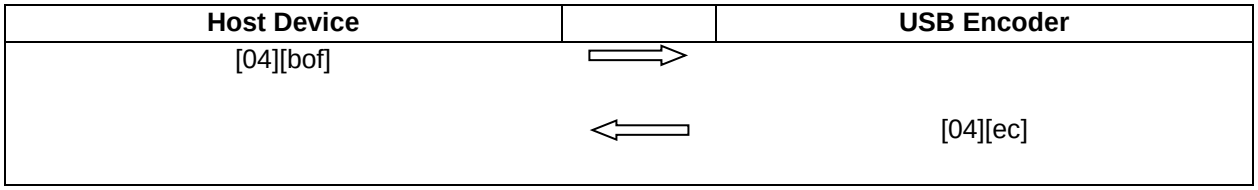
Buzzer On / Off Command (04)

Host sends this message to enable/disable buzzer on key presses

	Data Field	Type	Length	Description
bof	Buzzer	N	1	0-Disable, 1-Enable

Buzzer Command Response

	Data Field	Type	Length	Description
ec	Error Code	H	2	



Load New Key Code Table Command (05)

Host sends this message to Load New Code Table

	Data Field	Type	Length	Description
lt	Load New Code Table	H	Always 32	Key Code Table: 8 for 4W, 24 for 12W, or 32 for 16W

Load New Table Command Response

	Data Field	Type	Length	Description
ec	Error Code	H	2	

Host Device		USB Encoder
[05][lt][32 scan codes]	⇒	
	⇐	[05][ec]

Note: Length is always 32, for example for 4 way, it requires 8 codes and this will be the first 8 bytes, subsequent 24 bytes will be ignored by the encoder.

Format of table is as follows:

<modifier for key 1><code for Key 1><modifier for key 2><Code for Key 2>.....etc

The code table is specified in the user manual together with the modifier code. For example to program the following for 4 way :

Key 1 – A

Key 2 – a

Key 3 – 9

Key 4 - !

<0xE1><0x04><0x00><0x04><0x00><0x26><0xE5><0x1E>< 0x00><0x00>< 0x00><0x00>< 0x00><0x00>< 0x00><0x00>< 0x00><0x00>< 0x00><0x00>< 0x00><0x00>< 0x00><0x00>< 0x00><0x00>< 0x00><0x00>< 0x00><0x00>< 0x00><0x00>

Note: 32 bytes must be sent, for unused key code pad the values with 0x00.

Note: For shift modifiers there is a left and right modifiers value defined. So we can use 0xE1 – Left Shift and 0xE5 – Right shift. Similarly there is left and right Alt

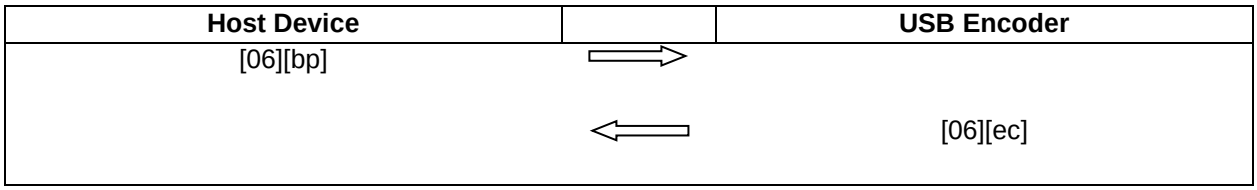
Buzzer Duration Command (06)

Host sends this message to change the duration of the buzzer period (when a key is pressed)

	Data Field	Type	Length	Description
bp	Duration	N	1	Value 0 - 9

Buzzer Duration Command Response

	Data Field	Type	Length	Description
ec	Error Code	H	2	



Buzzer value increments in 0.25s. For example 1 = 0.25s, 2 – 0.5s, 3 – 0.75s, 4 – 1.0s etc.,

Keypad Type Command (07)

Host sends this message to set keypad type

	Data Field	Type	Length	Description
kt	Keypad Type	N	1	0 – Function key – (default) 4 way 1 – Arrow Key 4 way 2 – Customised 4 way 3 – Telephone 12 way 4 – Calculator 12 way 5 – Customised 12 way 6 – Telephone 16 way 7 – Calculator 16 way 8 – Customised 16 way

Keypad Command Response

	Data Field	Type	Length	Description
ec	Error Code	H	2	

Host Device		USB Encoder
[07][kt]	⇨	
	⇦	[07][ec]

Self Test Command (08)

Host sends this command to request the USB Encoder to start/end a self test of the encoder.

	Data Field	Type	Length	Description
st	Self test	N	1	1 – start self test 2– end self test

Self Test Start/End Command Response

	Data Field	Type	Length	Description
ec	Error Code	H	2	

Host Device		USB Encoder
[08][st]	⇒	
	⇐	[08][ec] and if error_code is 00 Then it will be followed by keypresses
	⇐	[08][keypresses] After self test is started and If keypad key is not pressed within 5 sec It will timeout and self test will end and get following [08][0xff]

The self test command is used to test the key presses on keypad.

Host USB Encoder

-------Self Test (1) ----------→

This command disables the USB encoder USB devices, so if any keys are pressed the key codes are sent to Host over the HID datapipe channel as shown above. So if user presses key button 1.

←-------[08][0] ---------- User presses Key 1

←------[08][01]---------- User presses Key 2

If no key is presses for 5 sec then the command ends

←-------[08][0xff]----------

The host can stop the self test command by issuing

----------Self test(2) ----------→

←----------[08][00]----------

Write Config Data To Flash command (09)

Host sends this command to request the USB Encoder to write the configuration data from RAM to FLASH.

This command has no data associated with it.

RAM to FLASH Command Response

	Data Field	Type	Length	Description
ec	Error Code	H	2	

Host Device		USB Encoder
[09]	⇒	
	⇐	[09][ec]

Reset To Factory Default command (10)

Host sends this command to request the USB Encoder to reset parameters back to factory default.

This command has no data associated with it.

Reset To Factory Default Response

	Data Field	Type	Length	Description
ec	Error Code	H	2	

Host Device		USB Encoder
[10]	⇒	
	⇐	[10][ec]

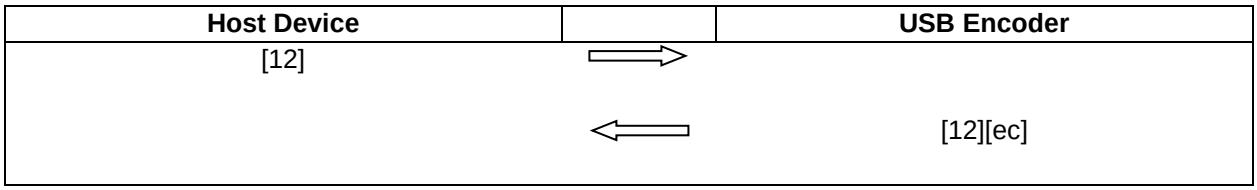
Enable BSL Command (12)

Host sends this command to request the USB Encoder to start downloader.

This command has no data associated with it.

Enable BSL Command Response

	Data Field	Type	Length	Description
ec	Error Code	H	2	



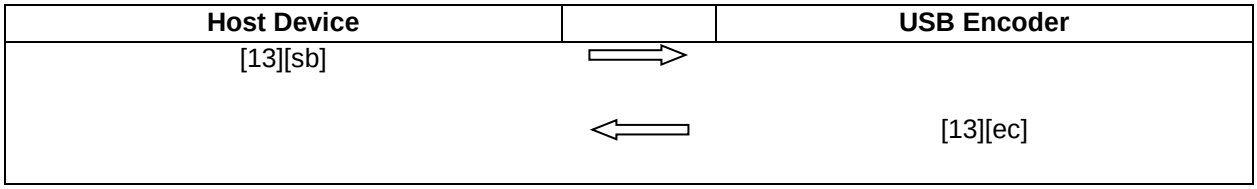
Status Buzzer Command (13)

Host sends this message to sound the buzzer for specified duration

	Data Field	Type	Length	Description
sb	Duration	N	1	Value 0 - 9

Status Buzzer Command Response

	Data Field	Type	Length	Description
ec	Error Code	H	2	



Key Press Code

Each time a key is pressed on keypad the USB encoder sends the keyboard report back to HOST.

When the key is released the USB Encoder sends 00000000 back to the HOST.

Keypress Code Type Report

	Data Field	Type	Length	Description
1	Key Press Code	A	1	Sends appropriate key code to host when keypad key is pressed.

Host Device		USB Encoder
	⇦	[KEY CODE]

Keyboard Report

HID Keyboard Report Format

	BIT7	***BIT6***	***BIT5***	***BIT4***	***BIT3***	***BIT2***	***BIT1***	***BIT0***
Byte0	*Right GUI*	*Right Alt*	*Right Sft*	Right Ctrl	Left GUI	Left Alt	Left Shift	Left Ctrl
Byte1	*Reserved*							
Byte2	Key_array[0]							
Byte3	Key_array[1]							
Byte4	Key_array[2]							
Byte5	Key_array[3]							
Byte6	Key_array[4]							
Byte7	Key_array[5]							

For example if user has a 4 way keypad connected to encoder and configured for Arrow key. If the user now presses the top key, which is “up arrow” and USB code of 52. Then keyboard report sent to host would be:

Byte 0 – 0
Byte 1 – 0
Byte 2 – 52
Byte 3 – 0
Byte 4 – 0
Byte 5 – 0
Byte 6 – 0
Byte 7 – 0

Now if the user customizes the top key to be “R SHIFT” (modifier) and USB code for “a” (04). If the user presses the top key, then the keyboard report sent to host would be:

Byte 0 – 20 This is Right Shift modifier.
Byte 1 – 0
Byte 2 – 52
Byte 3 – 0
Byte 4 – 0
Byte 5 – 0
Byte 6 – 0
Byte 7 – 0

The API makes the following functions available to developers

This is referenced in below functions:

```
  enum REQUEST_TYPE
  {
// message types
  DEVICE_STATUS = 1,            ///Device status message
  LED_BRIGHTNESS,
  LED_COLOUR,
  BUZZER_ON_OFF,
  LOAD_KEYCODE_TABLE,
  BUZZER_PERIOD,
  KEYPAD_TYPE,
  RESERVED_1,
  WRITE_CONFIG,
  FACTORY_DEFAULT,
  RESERVED_2,
  FIRMWARE_LOAD,
  STATUS_BUZZER
};
```

InitialiseStormUSBDevice

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

- Vendor ID – 0x2047
- Product ID – 0x0902

On successful finding the USB Encoder the manufacturer_local will be filled with “Storm Interface” and product_local will be filled with “USB Encoder”. 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 );


or


bool  InitialiseStormUSBDevice( int storm_vid, int product_pid);
```

Get Device Status

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

```
typedef struct
{
      unsigned char           led_brightness;
      unsigned char           led_color;
      unsigned char           buzzer;
      unsigned char           buzzer_period;
      unsigned char           keypad_type;
      unsigned char           keypad_table[32];
      std::string             FirmwareName;

} DEVICE_INFO;



///   \brief GetDeviceStatus Retrieves the keypad's status and information including:
///   Serial Number, Tamper Status, Firmware Version, 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 450 Encoder

///   \param _timeToWait is the time in milliseconds to wait for the data to be
      retrieved.

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

///     Possible error codes are:

///   DEVICE_INFO_STRUCTURE_NULL = User app passed in NULL pointer for DEVICE_INFO
structure

///   NO_USB_ENCODER_CONNECTED = No keypad is connected so cannot retrieve info

///   REQUEST_TIMEOUT   = Could not retrieve the info in the time alloted.
///

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

Set Led Brightness

This function sets the led brightness. The brightness can be set from level 0 to 9

0 will set the leds to off

```
///\brief SetLEDBrightness : Sets the LED brightness between 0 to 9, where 0 is off
///\param led_brightness             0 to 9, where 0 is off
///\return  0 on success, negative error code on failure
      //

int SetLEDBrightness(  int led_brightness,   int _timeToWait );
```

Enable/Disable Buzzer

```
This function will enable or disable the buzzer.

Buzzer = 1  enable
Buzzer = 0 Disable

///\brief EnableDisableBuzzer : Enables/Disables buzzer
      ///\param buzzer         0-disable, 1-enables
      ///\return  0 on success, negative error code on failure
      //

int EnableDisableBuzzer(  int buzzer,   int _timeToWait );
```

Beep Buzzer

This function will sound the buzzer for the buzzer_period. The buzzer period is defined between 0 - 9

```
///\brief SetBuzzerPeriod : Sets buzzer period betweeen 0 -9
      ///\param buzzer_period        0-off, 1 - 9
      ///\return  0 on success, negative error code on failure
      //

int   SetBuzzerPeriod(  int buzzer_period,   int _timeToWait );
```

Instructions for API		Date	Version	Details
		15 Aug 24	1.0	Split out from Eng Manual as separate doc
API	Date	Version	Details	
		1 Oct 2013	1.0	First Release
		2 Nov 15	1.1	