Level Developments Ltd
Spencer Place
97-99 Gloucester Road
Croydon
Surrey CR0 2DN
United Kingdom

t. +44 (0)20 86841400
f. +44 (0)20 86841422
sales@leveldevelopments.com
www.leveldevelopments.com

LCP-USB Inclinometer sensor

USB interface description

Level Developments
17.06.2020

Description

The LCP-USB sensor connects to a USB host (PC) with a standard 4 pin USB A connector. It is USB 2.0 compatible.

The sensor samples the two inclination axes, digital filters them and outputs the readings calibrated in degrees.

The host controller requests the readings over the USB interface, and the sensor replies according to the parameter requested. Parameters are detailed later in this document. The sensor has a maximum USB message rate of 100 messages per second, thus if the x and y axes are requested separately, the samples will be at 50Hz each. There is a combined x and y message which gives both axes in a single message. This message is recommended for most applications.

Level Developments supply a free C# sample application with source code for communicating with and controlling the sensor.

USB device type

The LCP-USB has a HID USB interface. This type of interface does not require a custom device driver for Windows, and is relatively simple to program into the users application code.

Vendor and Product ID

The Vendor ID is 0x04D8

The Product ID is 0xEC73

Endpoints

The sensor has a single endpoint, number 0, size 64 bytes of data in each direction.

Messaging

The sensor does not send any data unless requested by the USB host (PC). There is a simple protocol used to request data. The PC sends a message with commands in the first few bytes.

The byte fields are defined:-

Byte 0 Command code

0 – Read parameter
1 – Write Parameter
2 – Parameter reply

Byte 1 parameter number

0 – product ID.
1 – Serial Number
2 – X axis angle
3 – Y axis angle
4 – X & Y axis angles
5 – 360 Angle
6 – Sensor Temperature
7 – Filter index
8 – X axis direction
9 – Y axis direction
10 – 360 direction
11 – Single/dual axis

Byte 2 Data

1- n bytes of data depending upon parameter

Message examples

Request X	PC sends:	0x00 0x02 // read, parameter 2
	Sensor replies:	0x02 0x02 0x00 0x01 0x02 0x03 // response, parameter 2, value
Request Y	PC sends:	0x00 0x03 // read, parameter 3
	Sensor replies:	0x02 0x03 0x00 0x01 0x02 0x03 // response, parameter 3, value
Request X&Y	PC sends:	0x00 0x04 // read, parameter 4
	Sensor replies:	0x02 0x04 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07
		// response, parameter 4, red is x value, blue is Y value
Set Filter	PC Sends:	0x01 0x07 0x05 // write, parameter 6, value 5 = 4hz
	Sensor replies:	0x02 0x070x05 // response, parameter 6, value
Set X Direction	PC Sends:	0x01 0x08 0x01 // write, parameter 7, reverse
	Sensor replies:	0x02 0x08 0x01 // response, parameter 7, reverse

Parameter 0 – Product ID

A 16 bit code, msbyte first. The LCP-USB sensor is code 2. This is not adjustable.

Parameter 1 - Serial Number

32 bit value in 4 bytes, msbyte first. Set at the factory, not adjustable.

Parameter 2 – X axis angle

32 bit value, msbyte first of X axis angle in degrees, scaled by 1000. i.e. a value of 11234 equals 11.234 degrees.

Parameter 3 – Y axis angle

32 bit value, msbyte first of angle in degrees, scaled by 1000. i.e. a value of 11234 equals 11.234 degrees.

Parameter 4 – X & Y axis angles

Two 32 bit values of angle, scaled in the same way as parameters 3 and 4, contained in a single reply message from the sensor. This is to allow faster sample rates as the message rate from the sensor is limited to 100 messages/second.

Parameter 5 – 360 degree (single axis) angle

32 bit value, msbyte first of single axis angle in degrees, scaled by 1000. i.e. a value of 11234 equals 11.234 degrees.

Parameter 6 – Sensor Temperature

For future expansion

Parameter 7 – Filter index

The index to select one of 9 low pass filters used on the X and Y readings.

Index	Frequency
0	0.125Hz
1	0.25Hz
2	0.5Hz
3	1.0Hz
4	2Hz
5	4Hz
6	8Hz
7	16Hz
8	32Hz

Parameter 8 – X axis direction

Single byte to select normal or reversed X axis direction.

0 = normal
1 = reversed

Parameter 9 – Y axis direction

Single byte to select normal or reversed Y axis direction.

0 = normal
1 = reversed

Parameter 10 – Single 360 axis direction

Single byte to select normal or reversed single axis direction.

0 = normal
1 = reversed

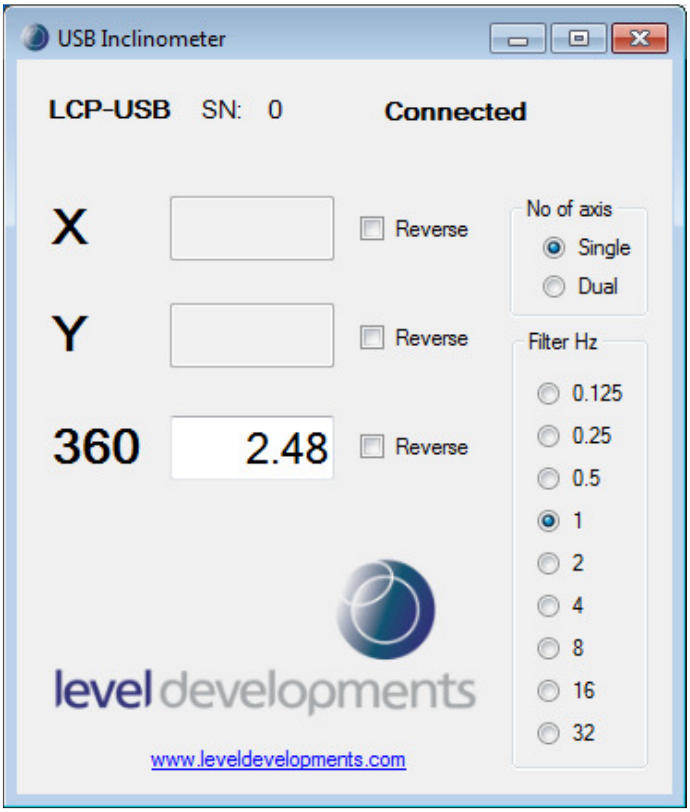
Parameter 11 –Number of axis

Single byte to select 1 or 2 axes.

1 = single 360 degree axis
2 = separate X and Y axes

C# Sample app notes

The sample app is a simple dialog app which connects to the sensor and displays values in a dialog window.



The USB interface is a HID model(Human Interface Device), and thus does not require a custom device driver to be loaded.

The App uses a 3rd party open source library to access the HID device driver. The library is called HIDLibrary, and is available at https://github.com/mikeobrien/HidLibrary.

The application code is not complex, and the sending of messages is very simple.

Sending a message

```
// poll the xy values
sendarray[0] = 0;       // endpoint #
sendarray[1] = 0;       // read
sendarray[2] = 4;       // x&y
_device.Write(sendarray);
```

Receiving a message

This is straightforward, as the HIDLibrary and App framework do all the work for you. The message arrives in $datarray$, and $processResponse$ is called automatically. Then it's simply a case of examining the values. E.g.

```
byte function = datarray[0];
byte parameter = datarray[1];

if ( parameter == 0)
   int productID = (datarray[2] << 8) + datarray[3];
```

Functions in Form1.cs

Form1_Load()

This connects to the sensor with the predefined Vendor ID 461 and Product ID 20. If a device with these IDs is not found then it does not connect.

Once connected, the function requests product ID, Serial number, filter, and the two direction settings from the sensor so it can initialize the controls with the current settings from the sensor.

OnReport()

This is an event driven function, triggered when a message arrives from the sensor. It copies the message into the application buffer $datarray,$ and then calls $processResponse$ to handle the message.

processResponse()

Parses the message and loads values into the appropriate controls.

timer1_Tick

This is the event function for a 0.1 second tick. It polls the sensor for the X&Y parameter (4). When the sensor replies, $OnReport()$ will be triggered, and thus $processResponse()$

Change actions

These are all similar, so we will not describe them all here. They are triggered by the user changing a control value. The changed value is then transmitted to the sensor. The sensor replies with the changed value, and this triggers $OnReport$