



## Controller Support

Revision 1.1 – 18<sup>th</sup> Aug 2009
[www.touch-base.com/documentation/technical](http://www.touch-base.com/documentation/technical)
[Technical info](#)
[UPDD listing](#)
[Hardware specifics](#)
[Data packet format](#)
[Firmware](#)
[EEPROM](#)
[Info sources](#)
[Contact](#)

Our UPDD driver supports 100's of touch controllers, mainly USB, serial and PS/2 devices and we try to keep support current with new controllers as they are released or made available.

### Technical information

To support a controller we either need access to a controller or technical details needed to configure support for the controller.

A [separate document](#) is available to advise how to identify a device connected to a system.

In some cases we work directly with the manufacturer and are supplied all the information we need to be able to fully support the device. In other cases it might be distributors, integrators or end uses that need to use our driver on a device that is not supported. We document here the information required to support a controller and also advise how to obtain that information if technical documents are not readily available.

The two most common touch controller types are USB or Serial. Adding controller support is normally undertaken free of charge as we are keen to ensure UPDD supports as many pointer devices as possible. Once a controller is configured in our production system we can generate the full range of drivers for the new controller.

***It is important to understand that any technical information supplied to us which is not in the public domain will be treated with the utmost confidentiality and never disclosed. If required we will enter into a joint NDA agreement to formalize this undertaking.***

The minimum information needed to support a controller is as follows:

- **Controller name as listed in UPDD**

We use three components in the naming of the controllers, company name, controller id and interface, e.g if we created our own USB controller, called the TC001 then the controller would be named Touch-Base, TC001, USB.

- **Interface settings**

**USB Vendor identification (VID)** USB identification unique to each vendor.

**Product identification (PID)** Allocated by the vendor and unique to each USB controller within the vendor's product range.

Drivers register vendor and product ids that they support. When a USB device is connected the vendor id and product id is used by the PnP sub-system to select the driver responsible for the device.

**Endpoint usage** Most USB controllers deliver touch data on endpoint 1

**Serial Communication parameters** Baud rate, parity, data bits, stop bits e.g. 9600,N,8,1

**For new controllers under development we have the following recommendation for USB:**

- 1) If HID compatible the controller has the advantage of being used both UPDD and alternatively with the HID driver supplied in many OS where some touch utilities are also available, such as calibration etc.
- 2) Interrupt transfer mode on endpoint 1 is recommended.
- 3) We would also recommend that the controller be designed so that it is not necessary to send vendor specific requests to activate it. This means that the controller can be verified easily using 3rd party tools to examine the data stream.
- 4) The controller must not make assumptions about the order of commands seen from Windows, but instead comply to the USB spec, otherwise it is likely not to work with future Windows versions and very likely not to work with non Windows systems.
- 5) We recommend that you execute the [Wingual](#) tests for pointer devices as these check various aspects of the controller. You will need to have HID, UPDD or another touch driver installed to make these checks. These tests can be undertaken by Touch-Base if requested. See [WHQL documentation](#) of more information.
- 6) It is recommended that each controller should carry a unique serial number in the string descriptor indexed by iSerialNumber. This allows for predicatable association of touch devices with monitors in a multi monitor environment. SString descriptors are explained at <http://www.beyondlogic.org/usbnutshell/usb5.htm#StringDescriptors>

- **Single touch data packet format**

When a single pointer device is in use it delivers a data packet to the system that carries co-ordinate information (X and Y and sometimes Z - pressure data) along with other device information, such as lift off trigger indicator. We need to know the EXACT structure of this packet.

- **Multi touch data packet format**

When a multi pointer device is in use it delivers data packets to the system that carries co-ordinate information (X and Y and sometimes Z - pressure data) along with other device information, such as lift off trigger and stylus information. We need to know the EXACT structure of this packet.

**For new controllers under development we have the following comments regarding packet structure:**

Generally speaking we are able to configure any given packet structure within our driver as we are able to define the packet structure at individual bit level so normally we expect to be informed of the packet structure implemented in the controller and we then define it as appropriate. However, for readers wanting advice as to a suitable packet structure, here is a example 12 bit packet structure that offers a status byte and 2 bytes per x and Y co-ordinate:

Bit	1	2	3	4	5	6	7	8
-----	---	---	---	---	---	---	---	---

Byte 0	1	0	<b>T</b>	0	0	0	0	0
Byte 1	0	X6	X5	X4	X3	X2	X1	X0
Byte 2	0	0	0	X11	X10	X9	X8	X7
Byte 3	0	Y6	Y5	Y4	Y3	Y2	Y1	Y0
Byte 4	0	0	0	Y11	Y10	Y9	Y8	Y7

Where **T** =1 when touching and 0 when untouch is detected.

This packet offers unique sync pattern (1<sup>st</sup> bit of each byte) such that lost bytes (say during transmission) will allow the driver to discard incomplete packets and resync on the 1<sup>st</sup> byte of the next packet (bit 1 = 1)

#### Notes:

HID may dictate packet structure format.

Ensure sync bits are defined, especially in serial devices.

Use unsigned integers for co-ordinate data (avoid FFFF = -1!).

Report (to Touch-Base) the exact numbers of co-ord bits used in the packet as UPDD definition relies on the correct definition to aid 'auto-calibration'. E.g 8 bits = 256 co-ord range, 9 bits = 512, 10 bits = 1024, 11 bits = 2048, 12bits = 4096 etc.

For multi-touch some of the other status bits can be used to indicate stylus id, e.g. bits 7 and 8 :- 00 = stylus 1, 01 = stylus 2, 10 = stylus 3 and 11 = stylus 4.

**Generally the above information is all we need to add support for a controller. However, there is some additional information that we made need to complete support or that allows us to implement support beyond mouse emulation as discussed below:**

- **Firmware commands**

Some controllers have a command set to adjust internal controller settings or to define settings that define a controller's functionality, which could be needed to work with our definition of the controller (e.g. some controllers can generate different data packet formats and UPDD would need to send the code to set the packet structure expected by UPDD)

The UPDD Console program could also expose firmware settings to the end user so that they can be adjusted as required. The console could, for example, also show the version of the firmware if it is available along with any other information considered relevant.

For USB devices we also need to know the end point used to send the firmware commands and the end point used to return the data.

- **EEPROM storage**

UPDD normally stores calibration data locally in the computer but there can be times when it is advantageous to store calibration data in the controller and if a controller supports EEPROM or other forms of storage we need technical details to implement this functionality.

## Information sources

### Official Documentation

In most cases we would hope we could be supplied the official technical information relating to the information above and controller against which to test the newly configured controller. However, we acknowledge this is not always possible.

Failing that a technical contact within the company able to assist with our queries is very useful.

If neither of these is available then, if you have the controller to hand, here are ways to determine some of the required information:

### Information gathering

Where official technical information is not readily available here are some approaches to identifying the required information if you have the controller to hand.

#### Serial

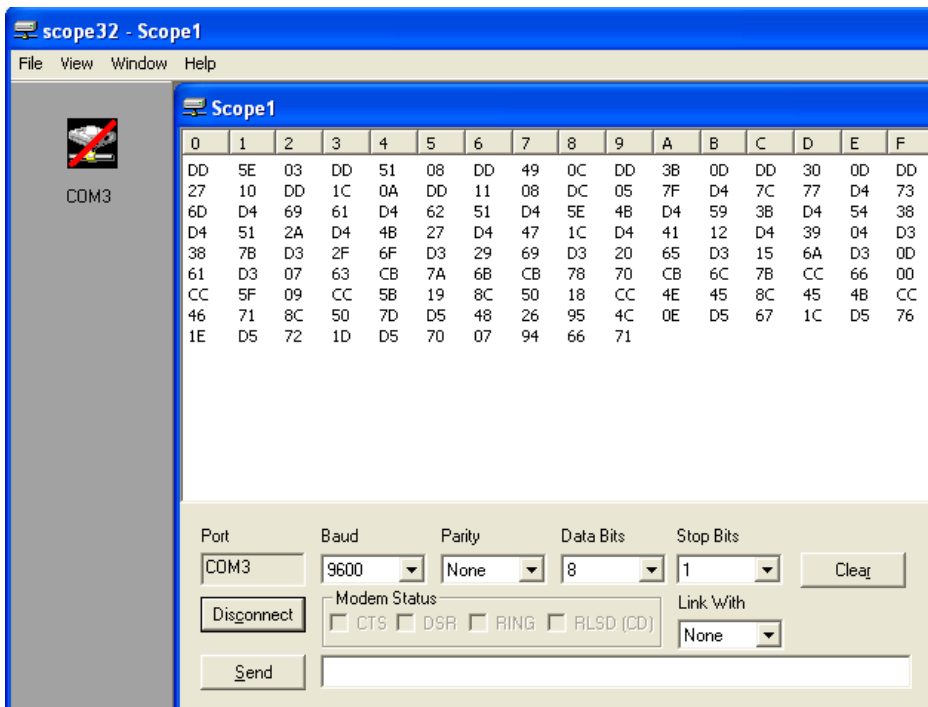
Given that the minimum information required is the serial interface parameters and the data packet format there are two approaches we can suggest, serial port data scope or UPDD capture tool.

#### Data scope

We have written a Windows serial port data scope that can be used to connect to a serial port and display the serial data. The data can then be saved to a file for analysis. We request that the data from a touch in each corner be saved in separate files and sent to us for further analysis.

The data scope program is available in the [utilities section](#) of our support page along with full [documentation](#).

The following example shows a touch screen on com 3, connected at 9600,n,8,1 and the data displayed from a prolonged touch on the screen:



### UPDD capture tool

We can configure a UPDD serial driver that can be installed and used to capture data packet information. The TBcalib program is used in 'test' mode to capture incoming data to a file that is sent to us for further analysis. See [TBcalib test parameter](#) details for more information.

See example below.

### USB

Given that the minimum information required is the Vendor and product identification and the data packet format we document ways to identify the VID and PID and two approaches to viewing the USB data via a USB analysis and UPDD data capture tool.

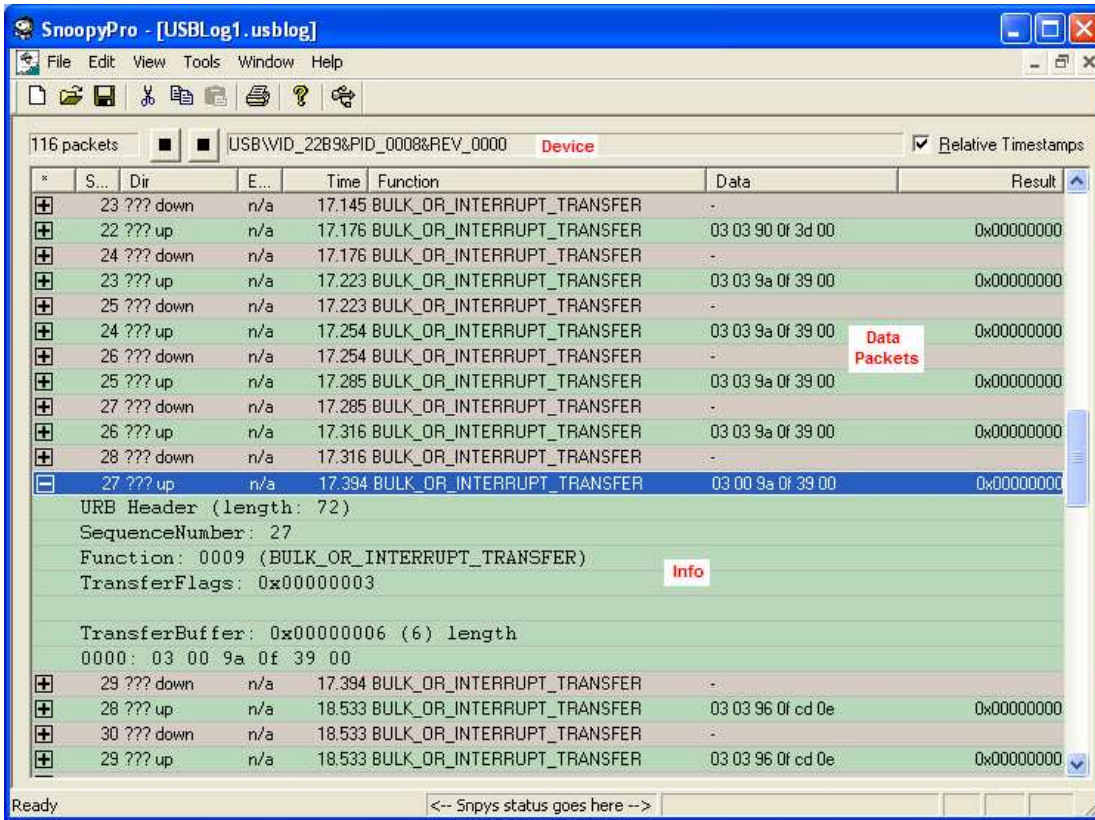
#### Vendor and Product id

There are numerous ways to [identify a USB device's vendor and product id](#) as documented in our 'Identifying touchscreen controllers' document.

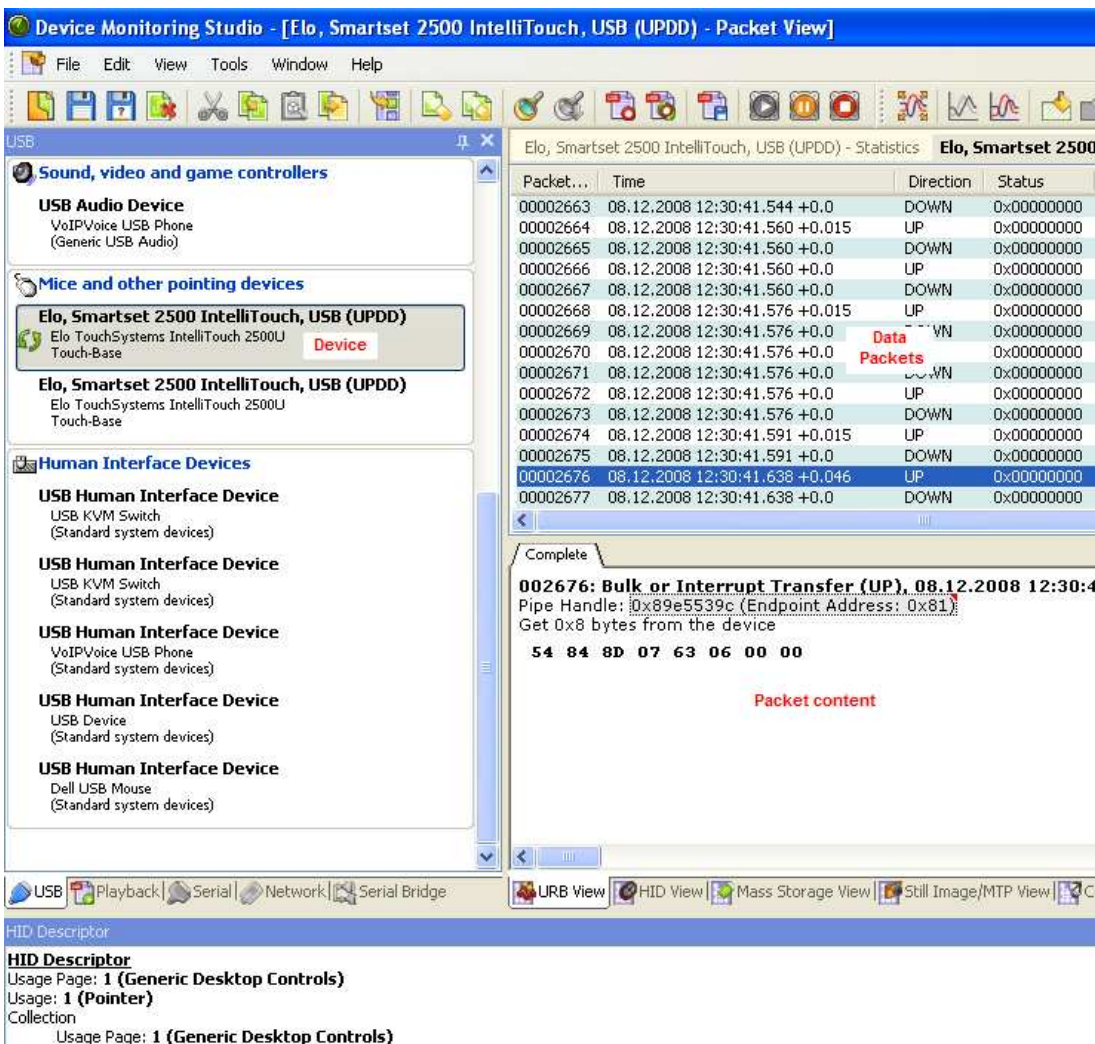
#### USB analysis tools

Various USB hardware and software USB data analysis tools are available, such as [SnoopyPro](#), [USBlyzer](#) and Device Monitoring Studio from [HDD Software](#). These tools can be used to select the USB device, record transfers and show transfer data, as seen below:

In this example a customer has captured a log using SnoopyPro from touching a touch screen USB device 22B9\0008 (eTurboTouch) and sending the log for analysis. From this we can see a 6 byte data packet, the sync byte = 03, the pen down / up byte is 03 / 00 and the X and Y co-ords follow in the next 4 bytes:



In this example we are capturing traffic on an ELO USB touch controller being handled by UPDD driver:

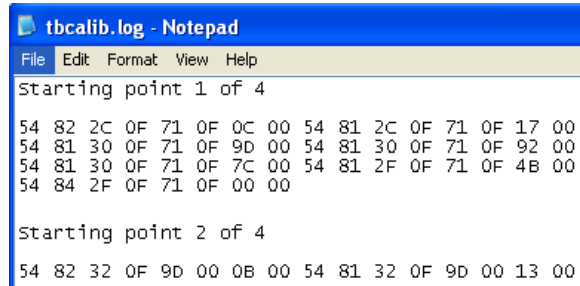


We request that the data from a touch in each corner be recorded and sent to us for further analysis.

### UPDD capture tool

We can configure a UPDD USB driver that will interface to a defined 'vendor and product id' that can be installed and used to capture data packet information. The TBCalib program is used in 'test' mode to capture incoming data to a file that is sent to us for further analysis. See [TBCalib test parameter](#) details for more information.

In this example, TBCalib test has been run against the same ELO USB controller as above and the calibration crosses touched. The log is shown in Windows Notepad:



```
tbcilib.log - Notepad
File Edit Format View Help
starting point 1 of 4
54 82 2C 0F 71 0F 0C 00 54 81 2C 0F 71 0F 17 00
54 81 30 0F 71 0F 9D 00 54 81 30 0F 71 0F 92 00
54 81 30 0F 71 0F 7C 00 54 81 2F 0F 71 0F 4B 00
54 84 2F 0F 71 0F 00 00

starting point 2 of 4
54 82 32 0F 9D 00 0B 00 54 81 32 0F 9D 00 13 00
```

### Contact

For further information or technical assistance please email the technical support team at [technical@touch-base.com](mailto:technical@touch-base.com)