Multi-touch and UPDD

Revision 1.4 – 7th Apr 2010
www.touch-base.com\documentation\general

There has been a resurgence of interest in the concept of multi-touch devices since the demonstration by Jeff Han of his conceptual multi-touch interface in 2006 and more recently the successful launch of the Apple iPhone and iPod Touch and the most recent and highly publicised Windows 7 launch, all with multi-touch capabilities.

Multi touch devices have been around for a long time and an overview of legacy system and the background of multi touch development by Bill Buxton is available at http://www.billbuxton.com/multitouchOverview.html. Wikipedia's entry is http://en.wikipedia.org/wiki/Multi-touch.

As of version 4.1.4 our UPDD has support for multi touch enabled devices and supports up to 16 styli and this documentation covers in detail UPDD multi-touch implementation.

For those customers contacting us for our advice on implementing multi-touch devices or applications this document also addresses general multi-touch considerations and therefore it is hoped will also answer general multi-touch queries.

Multi-touch utilisation needs to consider four main components, the touch hardware, driver, operating system driver interface and applications:

Hardware considerations

Until recently, touch devices only processed one touch point (co-ordinate data) at a time due to limitations of the input technology. However, some technologies allow for multi-touch sensing and increasingly touch hardware is now available that is able to detect multiple touches and report a separate data streams per stylus.

Before multi-touch can be utilised it is imperative that the touch hardware is delivering touch data streams for each stylus in use and in a manner that is recognised by the driver, such as UPDD or HID. Multi-touch cannot be achieved with software only solutions and therefore cannot be achieved with tradition touch hardware generating a single stylus data stream. With traditional touch hardware, if you place two or more stylus on the touch screen the device will only output a single data stream representing the mean average position of all touch points.

Data packet considerations

One of the most important features of a multi-touch device is the unique identification of individual stylus. The device must uniquely identify each packet of data sent to the host with a stylus number representing the input stream. The means of identification is a matter for the hardware manufacture, but consideration must be given to the end application. For example an application that is looking at relative positioning of styli, perhaps to look at gestures such as "squeezing" need only identify a number of stylus inputs, the correspondence of a stylus to a particular physical input is perhaps unimportant. Another example is an application that takes different actions depending on the stylus, such as a whiteboard program that has several pens to represent different colours or users will also need to be able to associate a particular stylus with a stylus number in the data packet. The input technology in use will influence the means by which styli are reported to the host. Not all input technologies will be able to uniquely identify individual styli.

In line with our standard recommendations for touch packets we suggest stylus identification be by means of a positive real binary encoded number in each packet. i.e. if 2 bits are used for stylus information then 00 = S1, 01=2, 10=3 and 11=4

UPDD implements a flexible packet parser so can recognise a number of different formats. The driver can also pre-process received multi touch data packets so if systems are sending data in a manner not immediately catered for by the UPDD data packet parser the received data can be reformatted into a structure more readily handled by the parser.

For operating systems that implement an HID (Human Interface Device) driver with multi touch extensions, such as Windows 7, then any HID MT compatible device can take full advantage of the MT functions available via the HID interface.

Touch gestures

Whilst it is possible for a controller to calculate and report only gestures this is not a recommended approach as this limits the flexibility available to drivers or application software to extend and improve the gesture capabilities of the device. In most cases it is better to deliver the stylus co-ordinate information and let the driver interface determine the gesture which is fed to the OS and applications as appropriate.
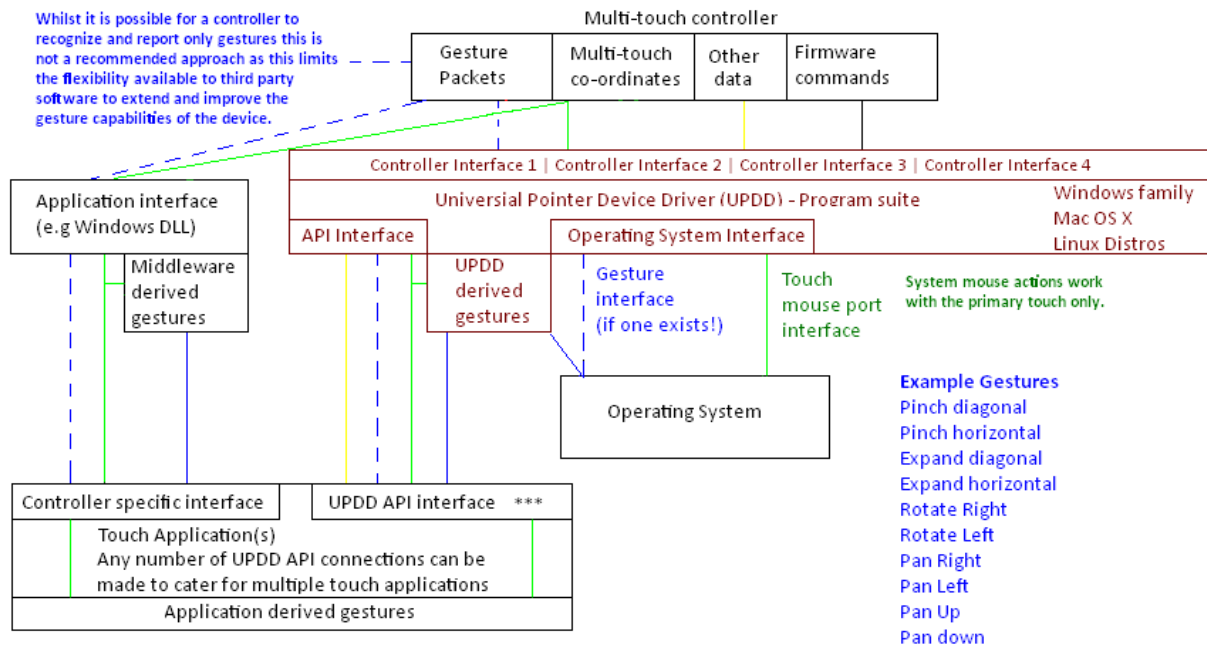
Driver operation

A driver receives the touch data stream from the hardware interface (USB, Serial etc) and makes that data available for use by the operating system mouse port (pointer) interface or application programs.

In the case of most touch drivers that work in standard desktop environments, the incoming co-ordinate data stream is processed to extract the co-ordinate data which is then scaled (based on calibration data) and passed to the mouse port interface of the operating system so that the system cursor is positioned under the point of touch. In addition to the cursor movement mouse clicks are emulated, either at the point of touch or lift-off, thus the touch screen functions as a pointer device by effectively emulating a mouse device. Although this usage reduces the touch screen function to that of a mouse it has the benefit of interacting with both the desktop manager and the system applications that have been developed to be controlled via a point and click device (normally a mouse or touch pad on a notebook) and therefore they also function with the touch device.

Our driver also makes touch data information available to system applications via the driver's Application Programming Interface (API). An application can register to receive data when certain events occur on the pointer device. One such event is the processing of a touch co-ordinate and the application code can receive the data directly and handle the data independent from the 'mouse emulation' interface. A UPDD API call also exists to enable / disable the system mouse port interface as required.

In a multi-touch environment the driver makes available the multi-touch stylus information to calling applications. In future versions of UPDD the driver could also derive gesture information and pass this to calling applications and optionally feed the gesture information to any future operating systems, or OS extensions, which can process multi-touch gestures.

Whilst it is possible for a controller to recognize and report only gestures this is not a recommended approach as this limits the flexibility available to third party software to extend and improve the gesture capabilities of the device.

Multi-touch controller

Gesture Packets | Multi-touch co-ordinates | Other data | Firmware commands

Controller Interface 1 | Controller Interface 2 | Controller Interface 3 | Controller Interface 4

Universial Pointer Device Driver (UPDD) – Program suite

Application interface (e.g Windows DLL)

API Interface | Operating System Interface

Windows family
Mac OS X
Linux Distros

Middleware derived gestures

UPDD derived gestures

Gesture interface (if one exists!)

Touch mouse port interface

System mouse actions work with the primary touch only.

Operating System

Controller specific interface | UPDD API interface   ***

Touch Application(s)
Any number of UPDD API connections can be made to cater for multiple touch applications
Application derived gestures

Example Gestures
Pinch diagonal
Pinch horizontal
Expand diagonal
Expand horizontal
Rotate Right
Rotate Left
Pan Right
Pan Left
Pan Up
Pan down

*** UPDD API offers a common interface to all single and multi-touch devices for co-ordinate and gesture information

Operating system mouse pointer interface

Touch co-ordinate data received by the driver is either delivered to the operating system to be processed by the underlying system pointer interface or passed directly to applications that have registered to receive co-ordinate data.

In operating systems designed for only 1 touch interface, such as XP, Vista, early Mac OS X etc the driver will only deliver the co-ordinate data from one stylus to the pointer interface. Where an OS can receive more than co-ordinate stream, such as Windows 7, them multiple data streams are passed to the OS. In the case of Windows 7 this is passed via the HID class interface.

In the case of Vista (single pointer) and Windows 7 (single and multi-touch) if the device is seen (real or virtual) as a compatible HID digitizer type device then extended touch functions are enabled within the OS and appropriate applications.

When processing multi-touch input UPDD considers stylus zero to be the primary stylus and associates this to the pointer interface in single touch environments.

Delivery of touch co-ordinate data to the OS can be disabled as required via a UPDD API function call.

Applications

Ultimately it is the individual applications that will utilise multi-touch input and as such will need an interface with the touch hardware. In multi-touch aware operating systems, such as Windows 7, applications can be written to receive both touch and gesture information via the appropriate interface and act accordingly. Where there is not a OS MT interface or where an application does not want to be restricted to one specific OS then direct access to the multi-touch data, such as via the touch driver's API or other interface method, is desirable. Once the touch data is received it is used to perform the multi-touch gesture as required, such as shrink or enlarge a photo or resize a browser etc.

To this end it is important that applications can interface with the driver to receive the touch data and it is also important that the application can identify individual stylus, either by individual stylus identification or code to identify the stylus based on received data.

With UPDD, input from all styli are processed and made available to client programs via the API callback mechanism. This extensible mechanism allows the device to be used for any purpose, such as concurrent drawing, touch typing, or gesture recognition.

We will add support for specific multi-touch hardware as it is made available.

Stylus recognition

Identification of a stylus is a function of the hardware; some input technologies, e.g. magnetic or infra-red pen devices might be able to associate a given physical stylus with a stylus number. Others might recognise the first touch as stylus 0, second as stylus 1, etc. Applications utilising the multi-touch features of UPDD should consider the details of the stylus recognition implementation of the hardware when choosing a suitable multi-touch input device.

API example

This simple example illustrates the use of a UPDD callback function to print co-ordinate and stylus information. The XY coordinates and touch events (touch / release) reported via the UPDD callback interface are identified as being for a given stylus with the new hStylus member.

```
HTBDEVICE hd = TBApiGetRelativeDevice(0);

TBApiRegisterDataCallback(hd,0,_ReadDataTypeXY,CBFunc);

void TBAPI CBFunc(unsigned long context, _PointerData* data)
{
```

```
    printf("device %d generated x=%d y=%d for stylus: %d\n",(int)data->pd.device,(int)data->pd.xy->rawx,(int)data->pd.xy->rawy, (int)data->pd.hStylus);
 }
```

Mouse port processing
Some applications might require that the default UPDD mouse processing be disabled. UPDD API function call TBApiMousePortInterfaceEnable() can be used for this purpose.

Development tools
There are a number of good multi-touch development tools on the market some of which are highlighted here:

Company / Web	Product	
Sotouch Intelligent Surfaces	So touch Presentation MT	Allows you to create MT presentations
Nuiteq	Snowflake MT development	Multi-touch SDK and demonstration programs
PyMT	PyMT	Open source library for developing MT applications
PQ Labs	SDK & Gesture Library	SDK for PQ Labs hardware
22Miles	Development	MT application software

UPDD test program with multi-touch

The driver's test program has been updated to handle multi-touch input. A simple, full screen, video demonstration of the test program in use with a multi-touch device is available here. Over time we will write example applications to receive multi-touch data and act accordingly and make the source code available for general reference.

UPDD supported or under development multi-touch hardware

This section will be updated as we add support for additional multi-touch devices:

Date	UPDD version	Manufacturer	Technology	Vid	Pid	Type
April 2008	4.1.4	NextWindow	Optical	1926	0003	Dual
Mar 2009	4.1.6	IST	Infra-red	Serial		Dual
April 2010	4.1.8	Quanta Computer	Optical	0408 (Pixart>>>>>)	3000 3001	Dual
May 2010	4.1.8	iNexio	Infra-red	1870	0100	Dual
May 2010	4.1.8	Zytronic	Projected Capacitive	14C8	0005	Dual

To be supported at some future date

		Manufacturer	Technology	Vid	Pid	Type
		Atmel - Maxtouch	Capacity			Multi - 10
		Touch International	Resistive			Multi - 10
		3M	Projected Capacitive	0596	0500	Multi - 20
		Stantum	Resistive	1F87	0002	Multi – 10+
		Wacom	Resistive			Multi – 5?
		KeeTouch	Resistive	1AAD	000F	Dual
		ELO	Surface Acoustic	04E7		Dual
		Next Window	Optical	1926	0064	Dual
		N-Trig DuoSense	Capacitive	1B96	0001	Multi - 6
		EETI/eGalax	Capacitive (?)			Dual
		Flat Frog	?			?
		Cando	Capacitive			Dual
		MosArt	?	0486	0185	Dual
		MosArt	?	062A	7100	Dual
		DiamondTouch	?	?	?	?

Contact

For further information or technical assistance please email the technical support team at technical@touch-base.com