Synaptics PS/2 TouchPad Interfacing Guide

PN: 511-000275-01 Rev. B

Copyright

Copyright © 2001–2011 Synaptics Incorporated. All Rights Reserved.

Trademarks

Synaptics, the Synaptics logo, OneTouch, ClearPad, EdgeMotion, LightTouch, ScrollStrip, TouchPad, and TouchStyk are trademarks of Synaptics Incorporated.

All other brand names or trademarks are the property of their respective owners.

Notice

Information contained in this publication is provided as-is, with no express or implied warranties, including any warranty of merchantability, fitness for any particular purpose, or non-infringement. Synaptics Incorporated assumes no liability whatsoever for any use of the information contained herein, including any liability for intellectual property infringement. This publication conveys no express or implied licenses to any intellectual property rights belonging to Synaptics or any other party. Synaptics may, from time to time and at its sole option, update the information contained herein without notice.

Conventions used in this document

The table below describes the documentation conventions. These conventions are used in all Synaptics technical literature.

Term	*Meaning*
$	Hexadecimal numbers are marked with a leading '$' sign: The number $7FF is equal to 2047 decimal.
—	Bits shown as "— " in register diagrams are equivalent to bits marked *Reserved.*
italics	*Italicized* words introduce a term described in the adjacent text or in the Glossary.
Reserved	*Reserved* is used to signify a bit or bit-field not currently used in any (published) way.
`Courier`	Courier font is used for text to be entered on a command line or in a program, or for text output from a device.
§	In the Glossary and index, this symbol is a reference to the section of this document where a word or concept is discussed.

 Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

Contents

Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

 Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

1. Introduction

This document describes the PS/2 protocol that Synaptics TouchPads and other pointing devices use to communicate with a host computer. This consists of the industry-standard "mouse" protocol plus a number of TouchPad-specific extensions. System architects and developers can read this document to learn how to interface to the TouchPad™ hardware. (For detailed mechanical and electrical data, refer to the Product Specification for your product.)

Most operating systems provide driver software to handle the TouchPad at the hardware level. Software developers will be more interested in the **TouchPad Driver API**, a high-level interface that Microsoft® Windows® applications can use to take advantage of all the special abilities of the Synaptics TouchPad and the Synaptics drivers. Appendix A describes the API.

Synaptics PS/2 applications are optimized for notebook computer implementations. Although the PS/2 protocol can be used for embedded systems, Synaptics offers other protocols, such as the Register Mapped Interface (RMI), which are better suited to embedded systems. See the *Synaptics RMI4 Interfacing Guide* (PN 511-000136-01) for more information.

The Glossary and Index in Appendix B defines the technical terms that appear in this document.

1.1. TouchPad features

The Synaptics TouchPad is a pointing device for computers and other electronic devices. To the user, the TouchPad is a flat, usually rectangular area of the computer that is sensitive to finger touch. By putting the finger on the TouchPad sensor and moving the finger around, the user can maneuver a cursor around the computer screen. By clicking a button or tapping directly on the pad, the user can select and drag objects on the screen. The TouchPad serves the same role in a computer system as a mouse or pointing stick, but its compact size, low cost, and lack of moving parts makes it ideal for portable computers.

Special applications might use the TouchPad for moving a cursor or as an absolute positioning tablet, or they might define touch-sensitive zones activated by finger motions or taps. *Dual Mode* TouchPads toggle between a cursor control mode and an application launch and control mode. A ClearPad™, which is a TouchPad made of transparent materials, can be used to construct compact and simple touch screens.

Some TouchPad models include other input functions in addition to finger sensing on a TouchPad surface. These functions may include physical "mouse" buttons or inputs to be connected to physical buttons, capacitive buttons, scrolling wheels or buttons, and touch-sensitive scrolling zones.

The advanced features of the Synaptics TouchPad make it the solution of choice for a variety of applications above and beyond simple mouse replacement. Synaptics offers a family of TouchPad models of various shapes and sizes, which connect to the rest of the computer system (the "host") using several different protocols. All Synaptics TouchPads do the following:

- support the same base set of features and modes,
- offer the same base set of commands and queries to the host, and
- operate according to the same principles.

Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

The subsections below provide some useful background information on how the Synaptics TouchPad senses fingers and how it translates finger actions into useful pointing behavior.

1.1.1. Sensing finger presence

Synaptics TouchPads use *capacitive sensing* to detect fingers. Whenever two electrical conductors are placed flat against each other and are separated by a thin insulator, an electrical capacitor is formed. The human body is a good conductor of electricity. In a TouchPad, a capacitor is formed by the human finger and a grid of electrical conductors with plastic or a thin insulating Mylar label between them. By measuring the capacitance of each conductor in the grid, the TouchPad can accurately determine the position of the finger as horizontal and vertical (X and Y) coordinates on the pad surface. The TouchPad can also determine the approximate finger pressure "Z."

1.1.2. Sensing finger location

The TouchPad uses various interpolation methods to determine the X and Y finger positions. Because the interpolation method used can subtly affect the behavior of the TouchPad, the various methods are briefly described here.

Most modern Synaptics TouchPads use the *peak interpolation method* to locate the finger. This method first estimates the finger position by finding the grid conductor with maximum finger capacitance. Then it refines the position by comparing to the capacitances of adjacent conductors. If a second finger or an accidental palm contact touches the TouchPad, this method preferentially tracks the original finger to keep the X and Y coordinates stable. Only if the original finger leaves the pad while another finger remains will the X and Y coordinates jump to track a different finger. A host driver using *Absolute mode* may wish to suppress cursor motion on any packet for which the finger count changes (see section 3.2.6) to prevent this jump in coordinates from affecting the cursor.

Many older Synaptics TouchPads used a different *centroid interpolation method*. This method uses formulas similar to physical "center of gravity" to find the center of overall capacitive touch on the pad. With two fingers present, the centroid of finger contact lies midway between them. If a second finger touches a centroid-based TouchPad, the X and Y coordinates will tend to slide toward the midpoint position.

Other quantities such as Z and W are also computed differently in the peak and centroid methods. For example, the centroid method computes Z as the total finger capacitance on the pad, but the peak method uses a more localized formula for Z. As a result, the centroid Z will keep increasing all the way to 255 if a whole palm presses against the pad, whereas the peak-based Z will tend to top out in the low 100s of Z units even as W increases to indicate a wider area of contact.

1.1.3. Filtering position data

The TouchPad uses a variety of "filtering" algorithms to convert the raw X and Y computations into smooth, pleasing motion even when electrical noise and physical effects have introduced some jitter into the capacitance measurements of the TouchPad. In *Relative mode*, the pad applies several filtering and acceleration algorithms as described in this section and section 1.1.4. In *Absolute mode*, the X and Y coordinates receive some basic filtering and are then passed on to the host with no further processing. For host software that uses Absolute mode, it may be worthwhile to apply some extra filtering to the X and Y values before using them for fine positioning.

 Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

Many filtering algorithms exist; no one algorithm is perfect for all applications. One simple method, the windowed average algorithm, computes each filtered coordinate value as the average of the last two unfiltered values. If *U* stands for the unfiltered finger position and *X* for the result of the filtering operation, then

$$X_{new} = (U_{new} + U_{prev}) / 2 \qquad \textit{(eq. 1)}$$

is the simple windowed average algorithm. (Here U_{new} is the most recent finger position, and U_{prev} is the previous finger position from the previous packet in the sequence of packets during a finger stroke.)

Another method is the decaying average algorithm:

$$X_{new} = (U_{new} + X_{prev}) / 2 \qquad \textit{(eq. 2)}$$

On the very first packet of a finger stroke, there is no value available yet for U_{prev} or X_{prev}. For this initial packet, it is reasonable to let $X_{first} = U_{first}$.

To increase the degree of smoothing, simply average three or more recent *U* values in the windowed average filter, or use a weighted average such as ¼ U_{new} + ¾ X_{prev} in the decaying average filter.

In Absolute mode, the TouchPad internally applies an unweighted decaying average filter to the X and Y data from each finger stroke. The filter is applied whenever the finger count is unchanged from the previous packet. Normally, this filtering is enough to produce usable position data with no undesirable artifacts. In special applications, the host can apply its own additional stage of windowed average or decaying average filtering. Also, in very special applications, the host can undo the effect of the built-in decaying average filter by "solving" *eq. 2* above for U_{new} as a function of X_{new} and X_{prev}.

1.1.4. Sensing motion

When no finger is on the pad, the Z value is normally zero. To detect a finger motion, the TouchPad looks for Z to increase beyond a "touch threshold," then for the X and Y positions to change in some way from each packet to the next, and finally for Z to return to nearly zero. This sequence of events makes up one *finger stroke* across the surface of the pad. The successive changes in (X, Y) during a stroke are translated into a succession of motion *deltas* (ΔX, ΔY) which are sent to the host. When the host translates these deltas into cursor motions, the cursor moves in a way that closely mimics the motion of the finger on the pad.

In principle, the motion deltas are simply the differences of successive positions multiplied by a suitable factor to cause a pleasing speed of cursor motion for a normal speed of finger motion. In other words,

$$\Delta X = S_T \times (X_{new} - X_{prev}), \quad \text{and} \qquad \textit{(eq. 3a)}$$

$$C_{new} = C_{prev} + (S_C \times \Delta X), \qquad \textit{(eq. 3b)}$$

where X is the finger position on the pad, C is the cursor position on the screen, and S_T and S_C are the speed factors employed by the TouchPad and the host mouse driver, respectively. If S_C were exactly 1.0, then ΔX would be measured in units of screen pixels of motion. Because S_C is not 1.0 in typical mouse drivers, ΔX is measured in arbitrary units of mouse motion which are known as *mickeys*.

Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

Most mouse drivers include an *acceleration* feature which varies S_C, based on the mouse speed, to help fast mouse motions cover more distance on the screen. The Synaptics TouchPad also includes its own acceleration feature which varies S_T based on the speed of finger motion. The TouchPad's acceleration feature acts mainly at low speeds, and is designed to complement the high-speed acceleration seen in mouse drivers.

In the Synaptics TouchPad, S_T is approximately 0.11 mickeys per Absolute position unit at normal finger speeds. S_T drops to about 0.04 at very low finger speeds. Section 3.2.3 describes Absolute position units. For a typical TouchPad, the overall speed factor is about 240 mickeys per inch or 9 mickeys per millimeter at full speed. Figure 3-7 of section 3.2.4 gives the Absolute mode resolutions for various other Synaptics TouchPad models. Multiply by 0.11 to get the expected number of mickeys per inch in Relative mode.

1.1.5. Sensing tapping gestures

For the purposes of this document, a *gesture* is any motion or action of a finger or fingers that the system interprets as something other than regular cursor motion. The most basic gesture is the ability to tap on the TouchPad for a *virtual button* click (action activated by a *tap* on the TouchPad instead of by a physical left mouse-click). The TouchPad itself supports *tap*, *double tap*, and *tap and slide*. The Synaptics TouchPad drivers add several more gestures, such as:

- Virtual scrolling, which allows scroll regions or zones along the side or bottom of the TouchPad
- EdgeMotion™ that allows the cursor to continue moving when the user's finger reaches the edge of the Touchpad surface.

These features are described in the driver control panel Help, the TouchPad Tutorial, and on the Synaptics public website. Although this functionality is provided by the Synaptics TouchPad driver, the driver may ship with advanced features turned off by default, and some features may be turned off by the OEM integrating the Synaptics device in its product. Access the Synaptics tab of the driver control panel to activate and adjust the features.

To detect a tapping gesture, the TouchPad looks for Z to increase beyond the touch threshold and then to return to nearly zero after only a fraction of a second, with little or no X or Y motion during this time. Note that a tap is actually detected shortly *after* the finger has left the pad, so the virtual button click, as reported to the host, begins after the actual tapping action is finished. Figure 1-1 illustrates tap and slide detection.

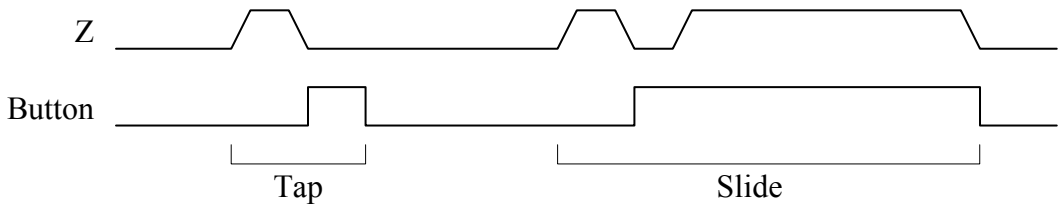


Figure 1-1. Tap and slide gestures

 Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

In this figure, the Z value and the state of the "virtual" left button are plotted against time as the user executes first a simple tap, then a slide gesture. Higher Z indicates the presence of a finger on the pad. A high level on the Button signal represents a simulated left button click as reported by the TouchPad. This figure is not to scale. Also, depending upon the quickness of the tap, there may be a delay between finger up and button click.

To move the cursor a long distance, the user may need to make several finger stokes. But because a slide gesture ends when the user lifts the finger, the entire slide must occur in a single stroke. Synaptics TouchPads offer the *EdgeMotion* feature to assist with long-distance slides. If, during a slide, the finger reaches the edge of the pad, the TouchPad begins to send a constant-speed motion signal to the host which continues as long as the finger stays at the edge.

Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

2. PS/2 Interface

This section of the *Interfacing Guide* describes the communication between the host and the TouchPad. The PS/2 protocol allows synchronous, bidirectional bit-serial communication between the host and the pointing device. Either side may transmit a command or data byte at any time, although only one side can transmit at one time. During initialization, the host sends command bytes to the device. Some commands are followed by argument bytes. The device acknowledges each command and argument byte with an ACK ($FA) byte, possibly followed by one or more data bytes. If the host has enabled "Stream mode" transmission, then the device may send spontaneous data packets to the host describing finger motions and button state changes.

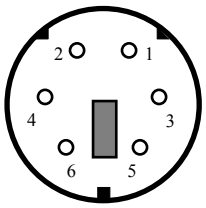
TouchPads integrated into notebook computers typically use the PS/2 protocol.

2.1. Electrical interface

The PS/2 protocol includes two signal wires as well as +5V power and ground. The signal wires, CLK and DATA, are bidirectional "open-drain" signals. They are normally held at a high (+5V) level by a 5–10K pull-up resistor on the host. By default, Synaptics enables pull-ups on the TouchPad, which allows the system to operate even if a non-compliant host fails to supply pull-ups. Either the host or the TouchPad can pull CLK or DATA low at any time. When the port is idle, both signal wires are floating high. The host can inhibit the device at any time by holding CLK low.

Note that neither side ever actively pulls CLK or DATA high. To output a logic 1, the wire is left undriven and is allowed to float high. The CLK and DATA lines should have a total capacitance of no more than 500 pF to ensure that the 5–10K pull-up resistor is able to drive them to a high voltage level in a reasonable time.

An integrated pointing device such as the Synaptics TouchPad typically consists of a circuit board module that brings the PS/2 signals out on a connector. For example, an external PS/2 mouse port uses a mini-DIN-6 connector with the following pinout (male connector view):



1	PS/2 DATA
2	N/C
3	Ground 0V
4	Power +5V
5	PS/2 CLK
6	N/C

Figure 2-1. PS/2 cable pinout

Synaptics offers a wide variety of board types, each with its own connector and pinout configuration. Please refer to the Product Specification for your sensor for specific connector and pinout information.

The following diagram shows the interconnections between the host and the Synaptics PS/2 TouchPad module:

 Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

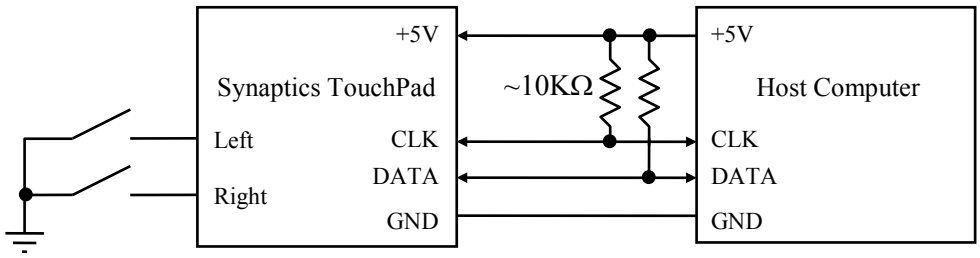


Figure 2-2. PS/2 system diagram

2.2. Button considerations

Most TouchPads support the two traditional mouse buttons, Left and Right. Some TouchPads support additional buttons in various configurations, such as:

- two standard buttons (Left and Right), and up to eight optional extended buttons,
- three standard buttons (Left, Middle, and Right), and up to eight optional extended buttons, and
- four buttons with a specific (Up-Down, Right-Left) coding.

When a TouchPad has a middle button, it reports the state of that button in the Middle button bit in the PS/2 Relative mode packet. In Absolute mode, the middle button status is reported only if the W mode feature has been turned on as described in section 3.2.8.

Extended buttons are any buttons other than the standard Left, Right, and Middle buttons. For extended buttons, also known as multi-button configurations, up to 11 buttons can be reported (although the actual number is dependent on the hardware configuration). To find the actual number of buttons supported by a particular TouchPad model, refer to the Product Specification for that model or use the queries described in section 3.2.8.

2.3. Byte transmission

Each byte transmitted between the device and the host includes a start bit (a logic 0), eight data bits (LSB first), a parity bit (odd parity), and a stop bit (a logic 1). Odd parity means the eight data bits and the parity bit together contain an odd number of 1s. During transmission, the device pulses the CLK signal low for each of the 11 bits, while the transmitting party (either the host or the device) pulls the DATA wire low to signal a logic 0 or allows DATA to float high to signal a logic 1.

Between transmissions, the bus can be in one of three states:

- **Idle.** If CLK and DATA are both high, there is no activity on the bus.
- **Inhibit.** If the host is holding CLK low, the device is inhibited from transmitting data. However, internal TouchPad processing continues to occur.
- **Request to Send.** If the host is holding DATA low and allowing CLK to float high, the host is ready to transmit a command or argument byte to the device.

Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

Note: Timing is extremely important for devices using the PS/2 protocol, for the following reasons:

- Packets are not self-synchronizing; timing delays between bytes may serve to indicate packet boundaries.
- In Stream mode, the device might start clocking at any instance, so the host must always be ready.
- In Remote mode, there is no way to tell if data is ready, so the host must poll constantly.

2.3.1. PS/2 waveforms

This section contains timing diagrams outlining the interaction between the host and the TouchPad. The input (host to TouchPad) consists of commands, while the output (TouchPad to host) consists of packets and responses. The TouchPad generates the clock for both output and input.

In this section:

- TouchPad signals are marked in black.
- Signals driven by the host are marked in blue (light gray in non-color renditions of this document).
- The portion of the timing diagram under discussion is marked with a red box (dark gray in non-color renditions).

The CLK signal is low for 30–50 µs and high for 30–50 µs in each bit cell. Notice that clocking is asymmetric in the PS/2 protocol. For TouchPad-to-host communication (packets and response bytes), the host samples DATA when CLK is low. DATA will be valid at least 5 µs before the falling edge of CLK and stays low at least 5 µs after the rising edge of CLK. For host-to-TouchPad communications (command bytes), DATA is sampled when CLK is high, and must be valid no later than 1 µs after the rising edge of CLK. In the line control bit, DATA goes low at least 5 µs before the falling edge of CLK and stays low at least 5 µs after the rising edge of CLK.

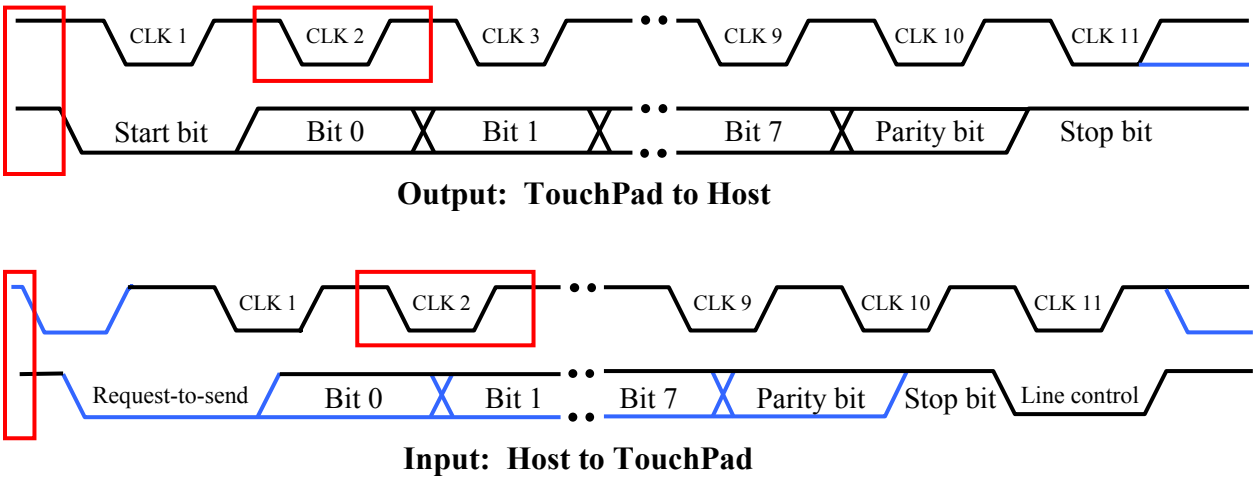


Figure 2-3. Idle state and clock waveform

Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

Both DATA and CLK wires are at logic 1 when idle. Either the host or the TouchPad can initiate a transmission whenever the bus is idle. The TouchPad cannot transmit if the bus is inhibited or in the request-to-send state.

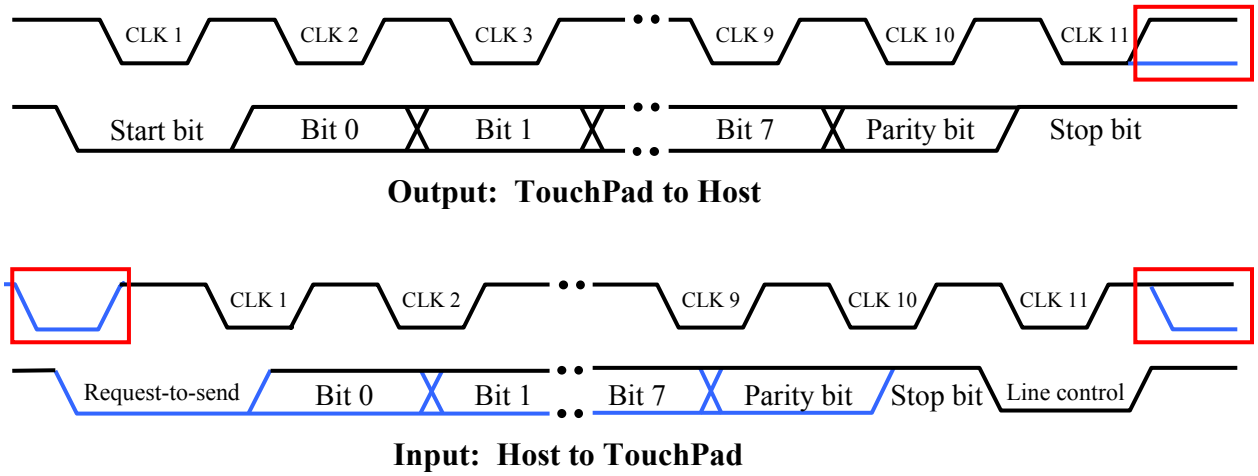


Figure 2-4. Host inhibition

If the bus is inhibited, the TouchPad waits for the bus to leave the inhibit state before transmitting to the host. The TouchPad is guaranteed to wait at least 50 µs after the inhibition ends before pulling CLK low to begin the start bit. (The TouchPad *may* wait considerably longer before beginning its transmission; the host's raising of the CLK wire is not a command to the TouchPad to begin transmission, but rather a signal that the TouchPad is now allowed to transmit as soon as it is ready to do so.)

The host signals its intent to transmit a command or argument byte by holding CLK low for at least 100 µs, then pulling DATA low and releasing CLK, thus putting the bus into the host request-to-send state. The TouchPad checks for this state at least every 10 ms.

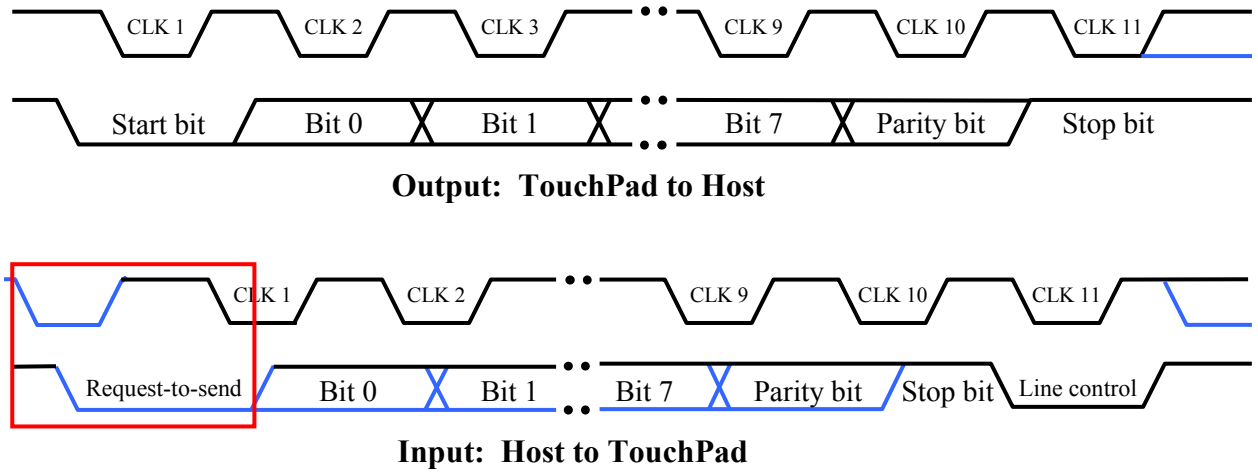


Figure 2-5. Request-to-send

Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

The TouchPad transmits a byte of data by pulsing CLK low and then high a total of 11 times, while transmitting the start bit, data bits, parity bit, and stop bit on the DATA wire. The host is expected to sample the DATA wire each time the CLK wire is low. The TouchPad changes the state of the DATA wire during the CLK high period.

If the bus is in the host request-to-send state, the TouchPad discards its pending transmission and receives and processes the host command or argument byte. (The one exception is the Resend ($FE) command, which responds by resending the most recent transmission even if that transmission was interrupted by the Resend command itself.) The Parity bits (odd parity), stop bit, and line control bit are responsible for error checking.

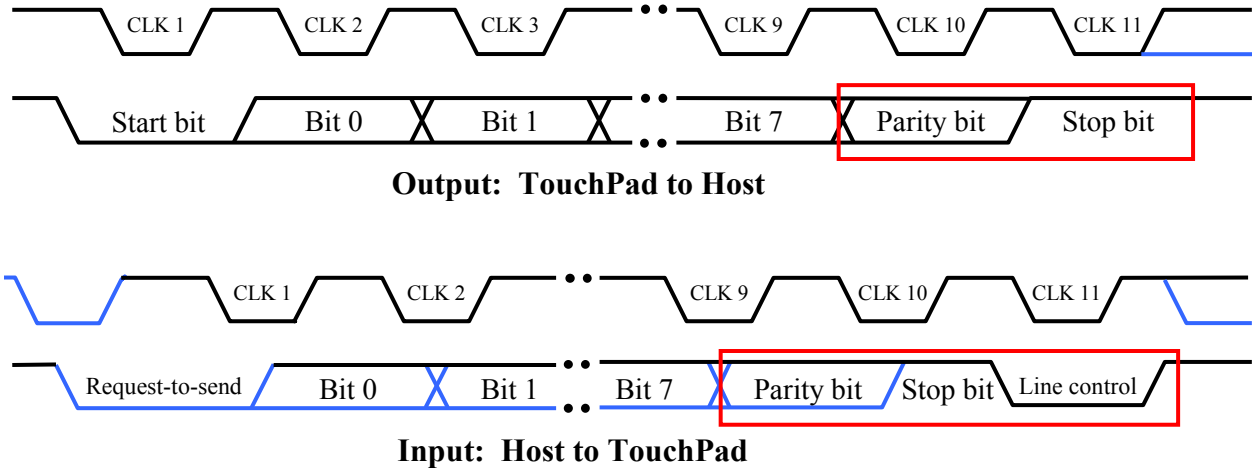


Figure 2-6. Waveform error checking

After the tenth clock in a host-to-TouchPad transmission, the TouchPad checks for a valid stop bit (DATA line high), and responds by pulling DATA low and clocking one more time (the line control bit). The host can then hold CLK low within 50 µs to inhibit the TouchPad until the host is ready to receive the reply. If the TouchPad finds DATA low during the stop bit, a framing error has occurred. The TouchPad continues to clock until DATA goes high, then sends a Resend to the host.

The host may hold CLK low after the transmission, effectively extending clock 11, to inhibit the TouchPad from sending further data while the host processes the transmission. When the *Absolute* and *Rate* mode bits are both 1, the TouchPad reports 6 × 80 = 480 bytes per second, which allows for about 2 milliseconds per byte. Since the waveform takes about one millisecond, the host should inhibit the bus for less than one millisecond per byte on average to achieve the maximum packet rate.

If the host inhibits the bus by holding CLK low for at least 100 µs during a TouchPad-to-host transmission, the TouchPad will recognize this and abort the transmission. The TouchPad recognizes an inhibit by noting that the CLK wire remains low during the high portion of the clock cycle. If the inhibit occurs before the rising edge of the tenth clock (the parity bit), the transmission of the byte is cancelled and the TouchPad will resend the interrupted byte as soon as the inhibit is released. (An ACK ($FA) reply to a command or argument byte is thrown away if cancelled, although the command being acknowledged is not cancelled, nor are the additional response bytes, if any, that follow the ACK.) If the inhibit begins after the tenth clock, the transmission is considered complete and the host must accept the transmitted byte.

 Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

2.3.2. Acknowledgement of commands

Each command or argument byte produces at least one response byte from the TouchPad. For every command or argument byte except the Resend ($FE) command, the response always begins with an "Acknowledge" or ACK ($FA) byte. For example:

Host:	***TouchPad:***
FF "Reset"	FA "Acknowledge"
(350 ms delay)	AA 00 "Ready"
E8 "Set Resolution"	FA "Acknowledge"
03 (argument)	FA "Acknowledge"
F4 "Enable"	FA "Acknowledge"
	09 00 00 (packet)
	08 04 02 (packet)

Depending on the command, the ACK byte may be followed by additional data bytes to make up a complete response. For the Resend ($FE) command, the response sometimes does not begin with an ACK.

The TouchPad responds within 25 ms, unless the host prevents it from doing so by inhibiting the bus. In multi-byte responses, the bytes of the response will be separated by no more than 20 ms. The Reset ($FF) command is an exception, where the $FA and $AA bytes are separated by up to 500 ms of calibration delay. The host must wait for the complete response to a command or argument before sending another byte. If the host *does* interrupt the response from a previous command with a new command, the TouchPad discards the unsent previous response as described in section 2.3.1.

If the TouchPad receives an erroneous input (an invalid command or argument byte, a byte with incorrect parity, or a framing error), the TouchPad sends a Resend ($FE) response to the host instead of an ACK. If the next input from the host is also invalid, the TouchPad sends an Error ($FC) response. When the host gets an $FE response, it should retry the offending command. If an argument byte elicits an $FE response, the host should retransmit the entire two-byte command, not just the argument byte.

On many PCs, the PS/2 port will also report a manufactured $FE response if the TouchPad does not send a response after a suitable timeout, or if the TouchPad does not respond to the request-to-send signal at all. Thus, an apparent $FE response from the TouchPad may also indicate that the TouchPad has been disconnected from the PS/2 port.

2.3.3. Host interface to the TouchPad

In a typical computer system, there are many levels of hardware and software between the TouchPad and the application software. This section will present an overview of the various steps in the path from TouchPad to application in an IBM PC-compatible computer.

Figure 2-7 illustrates a typical path that motion data travels on its way from a PS/2 TouchPad to a Windows application. Gray lines indicate paths that are not used as frequently.

Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

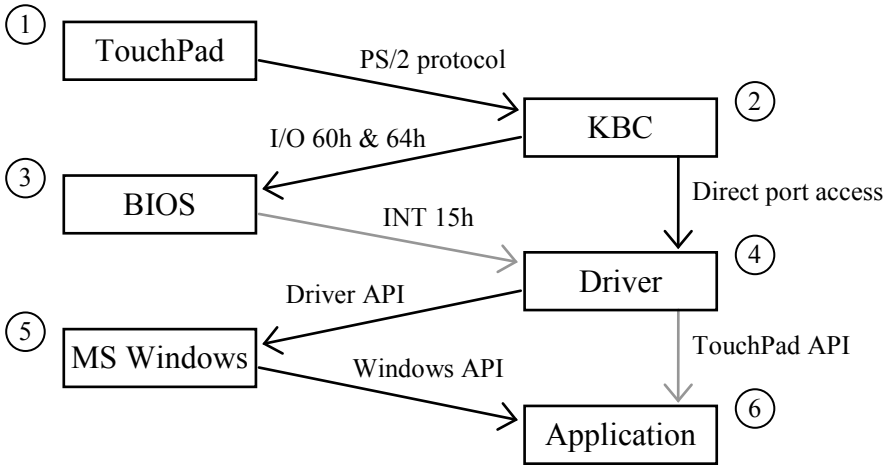


Figure 2-7. TouchPad to host data path (PS/2)

In step 1, the TouchPad contains a sensing mechanism and a microcontroller which converts raw sensor data into a form suitable for communication to the host.

In step 2, the keyboard controller (KBC) chip implements the host side of the PS/2 interface for the pointing TouchPad as well as the keyboard. The KBC communicates with the TouchPad via CLK and DATA wires. The KBC often has other responsibilities as well, such as power management. When the KBC is occupied with other duties (power management, keyboard processing), it holds the TouchPad's CLK wire low to make sure the TouchPad does not try to talk.

The KBC communicates to the TouchPad driver, the mouse driver, or the BIOS via I/O ports 60h and 64h and IRQ 12. When a byte of motion data arrives from the TouchPad, the KBC posts this data to port 60h, asserts IRQ 12, and pulls CLK low to prevent the TouchPad from sending more data. When the CPU responds to IRQ 12 by reading from port 60h, the KBC releases the CLK line and the TouchPad sends the next motion byte. (See section 6.1 for more discussion of the KBC.)

The standard PC BIOS software (step 3) provides a set of high-level mouse operations on INT 15h, function C2h. The original purpose of the BIOS was to isolate driver software from the details of I/O ports and IRQs, but many modern drivers talk directly to the KBC instead of using the BIOS facilities. (Note the direct port access arrow in Figure 2-7 between the KBC and the driver.)

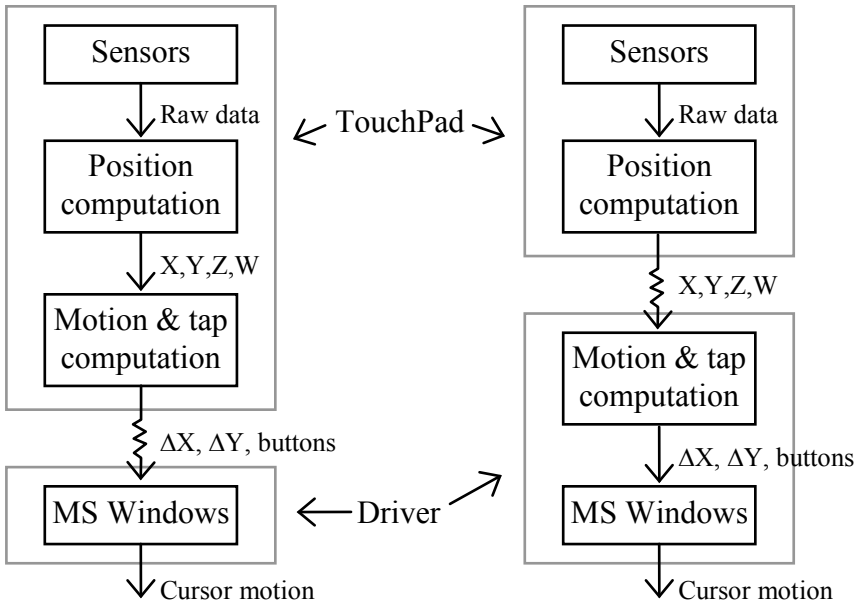
Frank van Gilluwe's book, *The Undocumented PC,* is an excellent reference to the KBC and the BIOS from the point of view of host software.

The driver (step 4) may be either a general-purpose mouse driver or the Synaptics TouchPad driver. The TouchPad driver does all the work of a mouse driver, as well as offering a variety of features which take advantage of the TouchPad's special abilities. The driver translates the information from the TouchPad into cursor motion and button click events in a form that Windows can use.

In step 5, the Windows operating system uses the motion information from the driver to display and move a cursor image on the screen. When the driver reports that a button has been clicked, Windows sees which application's window holds the cursor and forwards a Windows API (Application Programming Interface) message to that application (step 6).

 Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

Figure 2-8 compares the processing steps taken when the TouchPad is used with a standard mouse driver and with the Synaptics Windows TouchPad driver.



(a) Standard Mouse Driver *(b) Synaptics TouchPad Driver*

Figure 2-8. Motion processing in TouchPad and driver

Note that the processing steps are substantially the same, but the steps occur in different places depending on which kind of driver is used. With the standard mouse driver, the TouchPad is responsible for all processing, including converting (X, Y, Z, and W) position information into (ΔX, ΔY, and button) motion and tap gesture information. With the Synaptics TouchPad driver, the TouchPad operates in Absolute mode and reports (X, Y, Z, W) directly to the driver, which then takes over the motion and tap gesture processing itself.

Putting more of the processing in the driver has three advantages:

- The driver executes on a powerful CPU and so is able to use better algorithms with better pointing performance.
- The driver can implement a greater variety of advanced features.
- The driver now has access to the raw absolute data, which it can then provide to TouchPad-aware applications even though the driver still interfaces to Windows itself as a simple mouse driver.

The Synaptics TouchPad driver's API allows applications to get X, Y, Z, W, and other TouchPad-specific information, as shown by the second gray arrow between steps 4 and 6 in Figure 2-7. Appendix A describes the Synaptics TouchPad driver API.

Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

2.4. Power-on reset

At power-on, the PS/2 TouchPad performs a self-test and calibration, then transmits the completion code $AA and ID code $00. If the TouchPad fails its self-test, it transmits error code $FC and ID code $00. This processing also occurs when a software Reset ($FF) command is received. The host should not attempt to send commands to the TouchPad until the calibration/self-test is complete.

Power-on self-test and calibration normally takes 300–1000 ms. Self-test and calibration following a software Reset command normally takes 300–500 ms. In most Synaptics TouchPads, the delays are nominally 750 ms and 350 ms, respectively. But some newer TouchPads support an option in which the TouchPad transmits $AA and $00 immediately, and is immediately ready to accept host commands, without waiting for self-test and calibration to complete. This option should be ordered only if the KBC (keyboard controller) is compatible with it.

The Synaptics TouchPad never sends an $FC power-on/reset error code. Because the calibration algorithm is designed to adapt to environmental conditions rather than signal a hard failure, the power-on/reset response is always $AA, $00.

Officially, the host must not attempt to communicate with a PS/2 pointing device until the device has sent the power-on $AA, $00 announcement. For convenience, Synaptics TouchPads allow the host to put the bus into the request-to-send state immediately after powering up the TouchPad. The TouchPad will respond by clocking in the host's first initialization command as soon as it is ready. This command will override and discard the $AA, $00 announcement. The power-on calibration proceeds as usual, but in the background. If the host sends a Reset ($FF) command before the initial $AA, $00 announcement, then the $AA, $00 response to the Reset command may be delayed by the full 300–1000 ms required for power-on calibration.

To sense the finger's capacitance accurately, the TouchPad must perform an initial step called *calibration*. The TouchPad automatically calibrates itself upon power-up, and it also recalibrates itself in response to a PS/2 Reset ($FF) command. Calibration, which may take as long as one second, runs completely automatically; the only user-visible consequence is that the TouchPad will be unable to sense fingers until calibration is finished. Also, if a finger was present on the pad *during* calibration, then the TouchPad may miss the very first finger stroke after calibration. After the first stroke ends and the finger is lifted, the TouchPad operates normally.

The reset state of the TouchPad is as follows:

- Reported sample rate is 100 samples per second (see page 35).
- Reported resolution is 4 counts per mm (see page 37).
- Scaling is 1:1.
- Stream mode is selected.
- Data reporting is disabled.
- Absolute mode is disabled.

 Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

Note that only the *Absolute* bit of the TouchPad mode byte is cleared by a Reset ($FF) or Set Defaults ($F6) command. The other seven bits of the TouchPad mode byte are initialized to $00 only at power-on; these bits are preserved by the Reset and Set Defaults commands.

On rare occasions, the TouchPad may experience a spurious reset, often due to a power supply brownout or an electrostatic discharge (ESD). If this happens, the TouchPad will mostly reset itself as if after a power-on reset. If data reporting was enabled before the spurious reset, the TouchPad will attempt to come up re-enabled and without an $AA, $00 announcement so that the host does not experience an interruption of service. However, any other PS/2 settings or TouchPad mode byte settings will be lost. In particular, note that a spurious reset will cause the TouchPad to spontaneously revert from Absolute to Relative mode. If the host notices the TouchPad spontaneously reverting to the Relative mode packet format, it should reinitialize the TouchPad in the same manner as at power-up.

Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

3. PS/2 Modes

The Synaptics TouchPad supports two formats, or modes, for motion data packets. The default Relative format is compatible with standard PS/2 mice. The Absolute format gives additional information that may be of use to TouchPad-cognizant applications.

Data packets are sent in response to Read Data ($EB) commands. If Stream mode is selected and data reporting is enabled, data packets are also sent unsolicited whenever finger motion and/or button state changes occur. Synaptics recommends using Stream mode instead of Read Data commands to obtain data packets.

During transmission of a motion packet, the individual bytes of the packet will be separated by no more than 20 ms (assuming the host does not inhibit the bus). While PS/2 motion packets are lacking in explicit synchronization bits, if the host sees a delay of more than 20 ms between bytes it can assume the delay comes at a packet boundary.

3.1. Mouse-compatible Relative mode

Relative mode is the default mode for most Synaptics TouchPads. When power is applied, the TouchPad identifies itself to the host computer as a regular mouse. This allows the TouchPad to be used with standard mouse drivers. This mouse-compatible mode is called Relative mode because finger actions are reported to the host in terms of relative mouse-like motions across the sensor surface. The TouchPad reports this relative motion to the host in mouse-compatible *packets*.

Each packet reports the amount of motion in the X (horizontal) and Y (vertical) directions that has occurred since the previous packet. These amounts of motion are called *deltas,* and are written “ΔX” and “ΔY”. Left-to-right motion is reported with a positive ΔX and right-to-left is reported with a negative ΔX. Upward motion is reported with a positive ΔY and downward motion is reported with a negative ΔY. The packet also reports information about the “mouse” buttons.

Field	*Size (bits)*	*Range*	*Meaning*
ΔX	9	-256 to +255	Amount of horizontal finger motion
ΔY	9	-256 to +255	Amount of vertical finger motion
Left	1	0 or 1	State of left physical button or tap/slide gesture
Middle	1	0 or 1	State of middle physical button
Right	1	0 or 1	State of right physical button

Figure 3-1. Contents of Relative packet

Because the Relative packet is designed to be compatible with the existing mouse protocol, the exact contents of the Relative packet vary from one protocol to another. While the two-button mouse (left and right) is the default, Synaptics provides many methods to support multiple buttons. See Section 3.2.8 for details.

 Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

In Relative mode, placing the finger on the TouchPad does not automatically cause packets to be sent. However, moving the finger in any direction produces a sequence of packets that describe the motion. Pressing or releasing a mouse button causes the TouchPad to send a packet reporting this change in the state of the buttons. Tapping the finger quickly on the TouchPad also simulates a brief click of the left mouse button, and the slide gesture simulates an extended motion with the left button held down. (Figure 1-1 of section 1.1.5 illustrates these gestures in a technical way; the on-line help that comes with Synaptics' driver software has more user-oriented descriptions of gestures.)

The relative motion deltas generated by the TouchPad have a variety of smoothing and acceleration algorithms applied to produce pleasing motion when used for cursor positioning.

To improve the pointing precision of a slowly moving finger, a deceleration algorithm is used to generate smaller deltas for finger motion. Due to this feature, the resolution of the relative motion deltas is not fixed. However, for typical motions on typical TouchPads, the deltas report about 240 units per inch.

When there are no finger motions or button state changes to report, the TouchPad ceases to transmit packets and remains silent until the next motion or button activity.

3.1.1. Relative packet format

In the default Relative mode, each motion packet consists of three bytes. The first byte encodes various status bits, and the other two bytes encode the amount of motion in X and Y that has occurred since the previous packet.

	Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Byte 1	Y ovfl	X ovfl	Y sign	X sign	1	Middle	Right	Left
Byte 2	X delta							
Byte 3	Y delta							

Figure 3-2. PS/2 relative motion packet

Y ovfl: 1 = Y delta value exceeds the range –256…255, 0 = no overflow. When this bit is set, the reported Y delta will be either –256 or +255.

X ovfl: 1 = X delta value exceeds the range –256…255, 0 = no overflow. When this bit is set, the reported X delta will be either –256 or +255.

Y sign: 1 = Y delta value is negative, 0 = Y delta is zero or positive.

X sign: 1 = X delta value is negative, 0 = X delta is zero or positive.

Middle: 1 = Middle button is currently pressed, 0 = released.

Right: 1 = Right button is currently pressed, 0 = released.

Left: 1 = Left button is currently pressed (or gesture in progress), 0 = released.

X delta: This is the amount of motion ΔX that has occurred in the X (horizontal) direction since the last motion data report. This byte and the "X sign" bit of byte 1 combine to form a nine-bit signed, two's-complement integer. Rightward motion is positive, leftward is negative.

Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

Y delta: This is the amount of motion ΔY that has occurred in the Y (vertical) direction. Upward motion is positive, downward is negative. This is opposite from most OS interpretations of these bytes.

The three button-state bits reflect a combination of physical switch inputs and gestures. The left button bit is set if either the left physical switch is closed, or a tap or slide gesture is in progress. (If the *DisGest* mode bit is set, then the left button bit reports only the state of the physical left switch.) The right button bit is set only if the right physical switch is closed. The middle button bit is set only if the middle physical switch is closed. The TouchPad module usually supports two buttons, but modules may be ordered that support three buttons, if required.

The X and Y deltas report an accumulation of all motion that has occurred since the last packet was sent, even if host inhibition has prevented packet transmission for some time. Also, any host command except Resend ($FE) clears the motion accumulators, discarding any motion that had occurred before the command but that had not yet been sent in a packet.

The X and Y deltas have a resolution of about 240 DPI on a typical Synaptics TouchPad. See section 1.1.4 for further details.

3.2. Absolute mode

Synaptics TouchPads also support an Absolute mode of operation, where the TouchPad transmits an extended packet that reports the absolute finger position on the pad (X, Y), the finger pressure or contact area (Z), and other information, including the state of the buttons. The Synaptics Windows drivers operate the pad in Absolute mode. They use advanced algorithms to transform the absolute (X, Y, Z) data into smooth relative cursor motion, plus a wide variety of tapping and scrolling gestures and other features such as EdgeMotion.

In Absolute mode, the TouchPad reports packets continuously at the specified packet rate, either full or half speed, whenever the finger is on or near the pad. (Specifically, the TouchPad begins sending packets when Z is 8 or more.) The TouchPad also begins sending packets whenever any button is pressed or released. Once the TouchPad begins transmitting, it continues to send packets for one second after Z falls below 8 and the buttons stop changing. The TouchPad does this partly to allow host software to use the packet stream as a time base for gesture decoding, and also to minimize the impact if the system occasionally drops a packet.

3.2.1. Absolute packet format

When Absolute mode is enabled, each motion report consists of six bytes. These bytes encode the absolute X, Y location of the finger on the sensor surface, as well as the Z (pressure) value and various other measurements and status bits.

Synaptics TouchPads support two different Absolute packet formats, depending on the setting of the *Wmode* bit of the TouchPad mode byte (section 4.3). Note that if the *Absolute* and *Rate* mode bits are both set, then the TouchPad transmits up to 480 bytes per second over the PS/2 port. The PS/2 protocol in principle has plenty of bandwidth available to transmit data at this rate, provided the host takes care not to inhibit the bus for too long between bytes. See section 2.3.1 for further information.

 Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

The Absolute X/Y/Z packet format when *Wmode* = 0 is shown in Figure 3-3:

	Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Byte 1	1	0	Finger	*Reserved*	0	Gesture	Right	Left
Byte 2	Y position 11..8				X position 11..8			
Byte 3	Z pressure 7..0							
Byte 4	1	1	Y pos 12	X pos 12	0	Gesture	Right	Left
Byte 5	X position 7..0							
Byte 6	Y position 7..0							

Figure 3-3. PS/2 absolute X/Y/Z motion packet (Wmode = 0)

Note that the *Gesture*, *Left*, and *Right* bits appear twice in the Absolute packet. These bits are guaranteed to be identical in bytes 1 and 4 for a given packet. This and other aspects of the packet design allow low-level host software to interpret an Absolute packet as a sequence of two mouse-compatible three-byte packets. As high-level host software receives these three-byte half-packets, it can examine the upper two bits of the first byte to determine how to combine consecutive half-packets into full six-byte packets.

The Absolute X/Y/Z/W packet format when *Wmode* = 1 is shown in Figure 3-4:

	Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Byte 1	1	0	W value 3..2		0	W val 1	Right	Left
Byte 2	Y position 11..8				X position 11..8			
Byte 3	Z pressure 7..0							
Byte 4	1	1	Y pos 12	X pos 12	0	W val 0	R/D	L/U
Byte 5	X position 7..0							
Byte 6	Y position 7..0							

Figure 3-4. PS/2 absolute X/Y/Z/W motion packet (Wmode = 1)

In this packet, the four-bit W value replaces the *Finger* and *Reserved* bits and both *Gesture* bits. All other bits of the packet remain the same regardless of the *Wmode* setting. Section 3.2.6 describes the various purposes and interpretations of the W value.

On typical TouchPads, the L/U bit is identical to the Left button bit, and the R/D bit is identical to the Right bit. On MultiSwitch pads with the *capFourButtons* capability bit set (see section 4.4 for information on this bit) and *Wmode* enabled, the L/U and R/D bits also report the states of the Up and Down buttons, respectively. The L/U bit reports the logical XOR of the Left and Up button states. Viewed another way, L/U is the same as the Left bit, unless the Up button is pressed, in which case L/U is the complement of the Left bit. The R/D bit similarly reports the XOR of the Right and Down buttons. This encoding ensures that the packet will be backward compatible (and robust against meddling by "too-smart" keyboard controllers) whenever the Up and Down buttons are not pressed. Some MultiSwitch TouchPads use a different format that can accommodate up to 11 buttons; see Section 3.2.8.

Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

3.2.2. Absolute mode state bits

The Absolute mode packet, like the Relative mode packet, contains several bits that report the state of the buttons. An important difference is that in Absolute mode, the physical buttons are reported separately from tap and slide gestures. In mouse-compatible Relative mode, gestures and buttons are mixed together and there is no way for the host to distinguish them. (Naturally, if the host wishes for taps to act like left button clicks even in Absolute mode, the host is free to mix the separate state bits together itself.)

The button bits report the current state of the respective buttons. Each bit is 1 if the button is currently pressed, or 0 if the button is not pressed. Note that most Synaptics TouchPad models do not contain buttons mounted directly on the TouchPad board, but supply two or more external connector pins to which the system designer can attach buttons. These pins are labeled Left, Middle, and Right. It is up to the system designer to attach the pins to appropriately placed buttons. If the TouchPad supports MultiSwitch buttons, their state is also reported in the Absolute packet.

The *Finger* bit reports the state of the internal finger-presence check. This is a simple test based on comparing Z against a threshold of 25–30 units.

The *Gesture* bit reports the state of the "virtual" button; it is 1 during tap and slide gestures. (See section 1.1.5 for more details on the virtual button.)

The *Finger* and *Gesture* bits are fully redundant with the basic (X, Y, Z) information reported in the packet. In fact, the Synaptics drivers ignore these two bits and do their own more sophisticated finger and tap detection by examining Z directly. The Synaptics TouchPad provides these bits to simplify the use of the TouchPad in special applications where the Synaptics drivers cannot be used. Synaptics drivers set *Wmode* = 1 to replace these bits with a W field. For more information on the W value, see section 3.2.6.

3.2.3. Absolute X and Y coordinates

The X and Y values report the finger's absolute location on the TouchPad surface at any given time. When Z is zero, X and Y cannot be calculated and so are reported as 0. When Z > 0, X and Y are calculated and scaled to lie in the ranges shown in Figure 3-5. Most Synaptics TouchPad products are designed to scale their coordinates and pressure information to the same standard range regardless of the actual size of the sensor. This allows host software to interpret the coordinate data without needing to know the physical type of the TouchPad.

	X axis	*Y axis*
Absolute reportable limits	0–6143 ($0000–$17FF)	0–6143 ($0000–$17FF)
Typical bezel limits	1472–5472 ($05C0–$1560)	1408–4448 ($0580–$1160)
Typical edge margins	1632–5312 ($0660–$14C0)	1568–4288 ($0620–$10C0)

Figure 3-5. Absolute X and Y coordinates

In this table, the *absolute reportable limits* are guaranteed bounds on the values reported by the TouchPad under any circumstances. The *typical bezel limits* are approximate bounds on X and Y when fingers of typical size are used on TouchPads mounted in typical bezels or the analogous limits for TouchPads mounted under flat surfaces without bezels. The *typical edge margins* are suitable limits for deciding whether the finger is on the edge or in the interior area of the sensor surface; the finger is in the interior if X and Y lie within the edge margin limits. The edge margins are useful for hosts that wish to implement gestures such as EdgeMotion, scrolling zones, and corner taps.

 Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

The following figure illustrates the various coordinate limits:

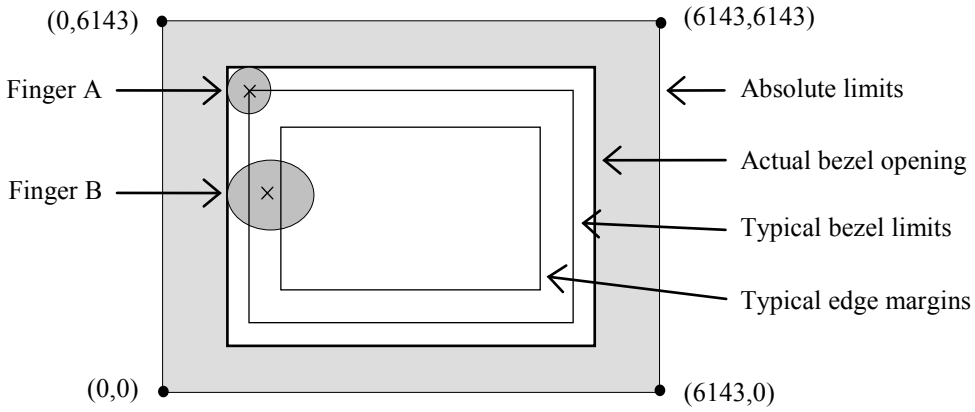


Figure 3-6. Coordinate limits (not to scale)

The typical bezel limits are inset a small distance from the "true" coordinates of the ideal bezel opening, because the TouchPad reports the coordinates of the *center* of the finger whereas the bezel constrains the *perimeter* of the finger. For any finger of reasonable size, the center will be inset a bit from the perimeter. For example, see finger A in the figure above. Similarly, the typical edge margins are inset somewhat from the bezel limits so that fingers of all sizes, such as the larger finger B shown above, will be able to fit within the edge zone.

For "portrait" oriented TouchPads, the X and Y axes limits in Figure 3-5 are interchanged. For example, the X bezel limits for a portrait TouchPad would be 1408–4448. Figure 4-11 (*b*) illustrates the portrait orientation.

The coordinate ranges in Figure 3-5 imply a resolution of 2000 DPI or more, depending on the physical size of the TouchPad. (Section 3.2.4 lists the actual resolutions for some TouchPad models.) In practice, the usable X and Y resolution is often somewhat reduced by the effects of electrical noise and physical jitter. Host software may need to apply filtering or averaging to the X and Y values before using them for fine positioning; section 1.1.3 gives some examples. In general, remember that a TouchPad is *not* a graphics tablet. Designers should not expect a compact, finger-operated TouchPad to match the stability, linearity, and repeatability of a precision pen-operated tablet.

3.2.4. Coordinate resolution

Synaptics TouchPads allow the host to query for the resolution of the X and Y coordinates in Absolute-mode units per millimeter. The resolutions are reported as 8-bit numbers called *infoXupmm* and *infoYupmm*. For example, a typical module has an *infoXupmm* of 85 units per millimeter, which means that moving the finger by one centimeter (10 mm) on the TouchPad surface results in a change of approximately 850 units in the X coordinate as reported in Absolute mode.

Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

The table below lists examples of reported resolutions for various TouchPad models:

Model	*infoSensor*	*Units per mm*	*Units per inch*	*Sensor area (mm)*
Typical	1	85 × 94	2159 × 2388	47.1 × 32.3
Mini	2	91 × 124	2311 × 3150	44.0 × 24.5
Super	3	57 × 58	1448 × 1473	70.2 × 52.4
UltraThin	8	85 × 94	2159 × 2388	47.1 × 32.3
Wide	9	73 × 96	1854 × 2438	54.8 × 31.7
Stamp	11	187 × 170	4750 × 4318	21.4 × 17.9
SubMini	12	122 × 167	3099 × 4242	32.8 × 18.2

Figure 3-7. TouchPad resolutions

The "units per mm" values in this table are the *infoXupmm* and *infoYupmm* resolution numbers reported by the various TouchPad models. Units per inch (DPI) are computed by multiplying units per mm by 25.4. In each case, the resolution is shown in the form *X* × *Y*. These resolutions are only approximate.

The sensor area is computed based on the bezel limits in Figure 3-5, which span 5472 – 1472 = 4000 units in X and 4448 – 1408 = 3040 units in Y. Dividing the total width of the TouchPad in units by the number of units per millimeter gives the width in millimeters, as shown in the rightmost column of Figure 3-7. Note that this represents the comfortable range of motion of a typical finger *within* a typical bezel, not the size of the bezel opening. See section 3.2.3 for more discussion of coordinate limits.

The values shown in Figure 3-7 are for TouchPads with landscape orientation; for portrait-oriented TouchPads (with *infoPortrait* = 1), the axes are exchanged. For example, a portrait UltraThin would have a 32.3 × 47.1 mm sensor area and report a resolution of 94 × 85 units per millimeter.

Custom-sized TouchPads also report a coordinate resolution. For some sensors, the coordinate stretching needed to get to a standard range can be severe. It is important to remember that even if the coordinates have been stretched to the point where a certain product reports 1 X unit = 3 microns, that does not imply in any way that X is *accurate* to 3 microns. Rather, 3 microns is merely the chosen unit for reporting positions, and the low few bits of the position values will almost certainly be noise.

Note that the resolution described in this section applies *only* to Absolute mode; in Relative mode, the resolution is variable. See section 1.1.4 for information about the resolution in Relative mode.

3.2.5. Absolute mode Z values

The *Z value* reports a measure of the amount of finger contact, which is affected by the contact pressure and by the angle at which the finger is held. The following table illustrates some typical Z values.

 Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

Value	*Interpretation*
Z = 0	No finger contact
Z = 10	Finger hovering near the sensor surface
Z = 30	Very light finger contact
Z = 80	Normal finger contact
Z > 110	Very heavy finger contact
Z = 255	Maximum reportable Z

Figure 3-8. Typical Z values

The measurement of Z is approximate; actual reported Z values will vary from one TouchPad to another and from one user to another. In fact, because capacitance is influenced by environmental effects such as the moisture of the skin, Z measurements may even vary from day to day for the same TouchPad and user. The measurement of Z also varies depending on the interpolation method used by the TouchPad. See section 1.1.2 for more about these methods.

When Z = 0, the position cannot be measured and will be reported as (0,0). W will also be 0. When Z is positive but very small, for example less than a module-specific threshold, then the position will be reported, but it may not be very accurate.

3.2.6. Absolute mode W values

Synaptics TouchPads support an optional value in the Absolute packet called *W*. The W value is reported only when the host enables a special *W mode*. The W value supplies extra information about the character of the contact with the sensor. The host can use W to distinguish among normal fingers, accidental palm contact, and multiple fingers.

The following table shows the W values defined for Synaptics TouchPads:

Value	*Capability bit set (bit = 1)*	*Interpretation*
W = 0	*capMultiFinger*	Two fingers on the pad
W = 1	*capMultiFinger*	Three or more fingers on the pad
W = 2	*capEWmode*	Extended W mode
W = 3	*capPassThru*	Pass-Through encapsulation packet (see section 5.1)
W = 4–15	*capPalmDetect*	Finger width; 15 is the maximum reportable width

Figure 3-9. Absolute mode W values

Sections 3.2.7 and 3.2.8 and the Read Model ID query information in section 4.4 show how the host can query for multi-finger and palm detection capability in a particular pad, as well as for the capability to report W.

If the *capPalmDetect* capability bit is set, then W values from 4 to 15 indicate that the TouchPad has sensed a single finger of a particular width. The host can watch for especially wide “fingers” as evidence that the TouchPad was activated by an accidental brush of the hand or palm rather than deliberate finger contact. If *capPalmDetect* is clear, the W value is reported as 4, regardless of actual finger width.

Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

Note: The finger width measurement is very approximate. Actual widths will vary from one TouchPad to another and from one user to another.

If the *capMultiFinger* capability bit is set, then W values 0 and 1 indicate a multi-finger touch. The TouchPad still reports a single pair of X and Y coordinates even when multiple fingers are on the sensor, as described in section 1.1.2. If *capMultiFinger* is clear, the multi-finger contact W value is reported as if it were a single finger.

The *capPalmDetect, capMultiFinger,* and other capability bits are located in the first and third bytes of the Read Capabilities query ($02). The Extended Model ID query ($09) also contains a capability bit; namely, the *capEWmode* capability bit that is bit 2 of the top byte. If this bit is set to 1 and the W value in Absolute packet equals 2, then this device supports the Extended W mode. For more information on Extended W mode, see section 3.2.9.

When the capability bits are clear (0), the corresponding W values are reserved for future definition by Synaptics and should be given no special interpretation by host software.

3.2.7. Extended capability bits

Synaptics TouchPads support *extended capability* bits that report to the host the presence or absence of various advanced features. The features are encoded in a 16-bit word. These capability bits are accessed via the Read Capabilities ($02) query. Other extended capability bits are reported as part of the Extended Model ID ($09) and Continued Capabilities ($0C) query responses. See section 4.4 for information on these queries.

Some bits are marked *Reserved*, which means that Synaptics has not yet assigned these bits to a specific function, or the bit is being used internally. There is no *specific* plan to add meanings to those spare capability bits; they will be assigned meanings as new product options are developed. For example, the *capFourButtons* bit was assigned when Synaptics developed the four-button MultiSwitch pad. Until that time, the bit that is now *capFourButtons* was a reserved/always zero query bit. Retroactively calling it *capFourButtons* is consistent because those older TouchPads did not have the four-button MultiSwitch capability. When an older TouchPad reports a 0 in that reserved query bit, a modern driver interprets it as reporting *capFourButtons* = 0, which is correct.

Capability bits are usually defined so that if the bit is 1, the driver knows how to interpret information that it would otherwise disregard, or it knows it can issue a type of command that would otherwise be invalid. This allows Synaptics hardware to be both forward- and backward-compatible.

3.2.8. Extended buttons

The extended model ID query (Figure 4-16 of section 4.4) contains an option that identifies the number and type of extended buttons supported by the device. Bits 15-12, *nExtendedButtons*, report the number of extended buttons supported by this TouchPad. Taken together, they form a 4-bit number *n* which represents the total number of extended buttons. The number (*n*+1)>>1 shows how many bits must be masked from both X and Y in the Absolute mode packet. If *n* is zero, then this TouchPad does not support extended buttons and the *Ext* bit decoded from the packet in Figure 3-10 will never be set.

Similarly, the *capMiddleButton* bit described in Figure 4-8 reports whether the middle button is set.

The TouchPad only reports extended button and middle button input in Absolute mode when *Wmode*=1. The TouchPad does not report any extended button states in Relative mode, but middle buttons are

 Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

reported. For Absolute mode, Synaptics uses an alternate extended button packet format as shown in Figure 3-10.

	Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Byte 1	1	0	W value 3..2		0	W val 1	Right	Left
Byte 2	Y position 11..8				X position 11..8			
Byte 3	Z pressure 7..0							
Byte 4	1	1	Y pos 12	X pos 12	0	W val 0	R/Ext	L/M
Byte 5	X position 7..4				b7	b5	b3	b1
Byte 6	Y position 7..4				b8	b6	b4	b2

Figure 3-10. Extended button Absolute mode packet (Wmode = 1)

In the extended button packet format, if the middle button is present its state is always encoded in bit 0 of byte 4 by XORing it with the current state of the Left button bit. Up to eight extended button states (b1 to b8) are reported in the low order bits of the X and Y position, which means that the low order position bits are lost. However, the extended button states are only reported if any one of them is actually pressed, so that position data reporting is only compromised when an extended button is being pressed.

The presence of bits b1 to b8 is indicated by the *Ext* bit, which is XORed with the Right button state and encoded in bit 1 of byte 4. If *Ext* is 1 (that is, if bit 1 of bytes 1 and 4 disagree), then one or more extended buttons are being pressed, and some bits in byte 5 and byte 6 now represent button states. If *Ext* is 0 (if bit 1 of bytes 1 and 4 are the same) then there are no external buttons being pressed (or all external buttons have been released). The packet only compromises the minimum number of bits necessary to support the number of buttons attached, and the other bits remain as valid position data.

For example, a TouchPad that has four extended buttons masks off bits 0 and 1 of bytes 5 and 6, and reports the extended button states in those bits. If the TouchPad has only one extended button, then bit 0 of bytes 5 *and* 6 are still masked, but only bit 0 of byte 5 is relevant. If a TouchPad has *n* extended buttons, then (*n*+1)>>1 low order bits are masked off from X *and* Y.

Masking off some low bits of X and Y results in degraded pointer resolution, but in typical TouchPad usage the effect proves to be quite minor. Also, note that all bits of X and Y report valid position information when *Ext* is 0. Thus, the minor degradation in pointing resolution occurs only when one or more extended buttons are pressed.

Note 1: If *nExtendedQueries* is zero, then query number $09 is not supported and *nExtendedButtons* is implicitly zero.

Note 2: If *nExtendedButtons* is greater than eight, then for the purposes of this specification *nExtendedButtons* should be considered to be invalid and treated as zero. Later specifications may use these larger values to represent further configurations of the packet format.

Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

3.2.9. Extended W mode

The extended model ID query (Figure 4-16 of section 4.4) contains an option for Extended W mode. To enter Extended W mode, the TouchPad must be configured with the Absolute Mode bit, *Wmode* bit and *EWmode* bit all set to 1. The *EWmode* is bit 2 of the TouchPad mode byte, described in section 4.3. Extended W mode encapsulates data into an absolute mode 6-byte PS/2 packet. In this mode, a W value of 2 indicates that this is an encapsulated packet and that the X, Y, and Z fields have all been redefined by the EW packet code.

EW packet codes 0-7 return 30 bits of encapsulated data:

	Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Byte 1	1	0	0	0	0	1	Right	Left
Byte 2	Defined by EW packet code							
Byte 3	Defined by EW packet code							
Byte 4	1	1	Defined by EW		0	0	Right	Left
Byte 5	Defined by EW packet code							
Byte 6	EW packet code (0 – 7)				Defined by EW packet code			

Figure 3-11. Extended W mode packet codes 0-7

As the figure indicates, the definitions of the bits in Bytes 2, 3, 5, and parts of Bytes 4 and 6 are defined by the Packet Code in bits 7:4 of Byte 6. Three of the available packet codes are used:

- 0 – This code defines a scroll wheel packet, described in section 3.2.9.1.
- 1 – This code defines the secondary finger packet, described in section 3.2.9.2.
- 2 – This code defines finger state information, described in section 3.2.9.3.

EW packet codes $80-$FF return 26 bits of encapsulated data:

	Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Byte 1	1	0	0	0	0	1	Right	Left
Byte 2	Defined by EW packet code							
Byte 3	Defined by EW packet code							
Byte 4	1	1	Defined by EW		0	0	Right	Left
Byte 5	Defined by EW packet code							
Byte 6	EW packet code ($80 – $FF)							

Figure 3-12. Extended W mode packet codes $80 - $FF

Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

EW packet codes $08-$7F are reserved. The button bits in bytes 1 and 4 continue to report, including any special meanings that may be encoded in the copy of the buttons in byte 4, such as the 4-button, middle button or extended button encodings.

Note: Although they may report that extended buttons are pressed, the state of the extended buttons is not reported in the encapsulated Extended W packet.

The subsections below describe two EW packets, for a wheel encoder and for secondary finger information.

3.2.9.1. Wheel encoder data

This packet is generated whenever a wheel encoder has a non-zero delta to report.

	Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Byte 1	1	0	0	0	0	1	Right	Left
Byte 2	Wheel Encoder 1 Delta							
Byte 3	Wheel Encoder 2 Delta							
Byte 4	1	1	Wheel 4 Delta [5:4]		0	0	Right	Left
Byte 5	Wheel Encoder 3 Delta							
Byte 6	0				Wheel Encoder 4 Delta [3:0]			

Figure 3-13. Extended W mode packet code 0, wheel encoders

3.2.9.2. Secondary finger information

This packet is generated to report the X, Y and Z of a secondary finger upon the TouchPad. During the duration of the multiple finger contact, the TouchPad will interleave standard W mode packets (reporting the primary finger X, Y and Z, and a W value of 0 (two fingers present) or 1 (three or more fingers present) with this Extended W packet. When only a single finger is present on the TouchPad, it will be considered the primary finger and its X, Y, Z, and Width data will be reported in a standard W mode packet and no secondary finger EW packets will be sent.

If the primary finger lifts, then the secondary finger will become the primary finger. Host software may want to check for this occurrence by noticing the finger count going to 1 and the primary finger position reported moving from its previously reported position to one very close to the last reported secondary finger position.

To fit in the EW packet, the secondary finger data do not report in the full resolution of the primary finger. The X and Y positions have discarded their LSBs and only report bits 12:1. The Z field is only six bits wide. It will peg its internal Z to $7F and then discard its LSB to report Z[6:1].

Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 1	1	0	0	0	0	1	Right	Left
Byte 2	Secondary xPos [8:1]							
Byte 3	Secondary yPos [8:1]							
Byte 4	1	1	Secondary Z [6:5]		0	0	Right	Left
Byte 5	Secondary yPos [12:9]				Secondary xPos [12:9]			
Byte 6	1				Secondary Z [4:1]			

Figure 3-14. Extended W mode packet code 1, secondary finger information

3.2.9.3. Finger state information

The Finger State information packet provides finger count information:

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 1	1	0	0	0	0	1	Right	Left
Byte 2	Reserved				FingerCount [3:0]			
Byte 3	Primary finger index							
Byte 4	1	1	Reserved		0	0	Down/ Right	Middle/ Left
Byte 5	Secondary finger index							
Byte 6	2				Reserved			

Figure 3-15. Finger State Extended W mode packet

The *FingerCount* field in Byte 2 can notify the host of the presence of 0 – 15 fingers. In addition, this packet provides the index of the finger that is being reported as the primary finger and the secondary finger.

3.2.10. V field for reporting width when multiple fingers are on TouchPad

When multiple fingers are on the TouchPad, W reports finger count rather than width. To report width when you have two or more fingers on the TouchPad, Synaptics created a virtual field called V which is encoded in low order bits of X/Y/Z when W=0 or 1. See Figure 3-16 and Figure 3-17.

 Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

	Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Byte 1	1	0	W value 3..2		0	W val 1	Right	Left
Byte 2	Y position 11..8				X position 11..8			
Byte 3	Z pressure 7..1							V[2]
Byte 4	1	1	Y pos 12	X pos 12	0	W val 0	R/D	L/U
Byte 5	X position 7..2						V[0]	X pos 0
Byte 6	Y position 7..2						V[1]	Y pos 0

Figure 3-16. Primary Finger Packet W=0 or 1

	Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Byte 1	1	0	0	0	0	1	Right	Left
Byte 2	Secondary xPos [8:2]							V[0]
Byte 3	Secondary yPos [8:2]							V[1]
Byte 4	1	1	Secondary Z [6:5]		0	0	Right	Left
Byte 5	Secondary yPos [12:9]				Secondary xPos [12:9]			
Byte 6	1				Secondary Z [4:2]			V[2]

Figure 3-17. Secondary Finger Packet when W=2 and EWcode = 1 (2nd Finger Position)

Width equals V+8, where V=0 means that width is 8 or less. See Figure 3-18.

V	*Width value*
0	Width = 8 or less
1	Width = 9
2	Width = 10
3	Width = 11
4	Width = 12
5	Width = 13
6	Width = 14
7	Width = 15

Figure 3-18. Decoding V

Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

4. PS/2 Commands

This section describes the commands available to identify the available features for your TouchPad, configure your TouchPad, and customize its operation.

4.1. Command set

The Synaptics TouchPad accepts the full standard PS/2 “mouse” command set. This section describes the mouse commands, along with any special properties of those commands as they are implemented on the Synaptics TouchPad:

- $FF Reset
- $FE Resend
- $FC Error
- $FA Acknowledge
- $F6 Set Defaults
- $F5 Disable
- $F4 Enable
- $F3 Set Sample Rate (argument $00-$FF)
- $F2 Read Device Type
- $F0 Set Remote Mode
- $EE Set Wrap Mode
- $EC Reset Wrap Mode
- $EB Read Data
- $EA Set Stream Mode
- $E9 Status Request
- $E8 Set Resolution (argument 0-3)
- $E7 Set Scaling 2:1
- $E6 Set Scaling 1:1

If the TouchPad is in Stream mode (the default) and has been enabled with an Enable ($F4) command, then the host should disable the TouchPad with a Disable ($F5) command before sending any other command. However, if the host *does* send a command during enabled Stream mode, the TouchPad abandons any data packet or previous command response that was being transmitted at the time of the command. The TouchPad will not send any further data packets until the response to the new command is finished. The commands are described in detail below.

$FF **Reset.** Performs a software reset and recalibration as described in section 2.4. Response is ACK ($FA), followed by $AA, $00 after a calibration delay of 300–500ms.

 Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

$FE **Resend.** The host sends this command when it detects invalid output from the TouchPad. The TouchPad retransmits the last packet of data: for example, a three- or six-byte motion data packet, a one-byte response to the Read Device Type ($F2) command, or the two-byte completion-and-ID reset response ($AA, $00). The ACK ($FA) byte sent to acknowledge a command is not stored in any buffer or resent. If the last output from the TouchPad was an ACK with no additional data bytes, Resend responds with an ACK.

The TouchPad will send a Resend ($FE) to the host if it receives invalid input from the host; see section 2.3.2. The $FE command means the same thing sent either direction.

$FC **Error.** If the TouchPad receives an invalid command byte, it replies with a Resend ($FE) byte. If it immediately receives a second invalid command, it replies with an Error ($FC) byte.

$FA **Acknowledge.** The response for valid output is ACK ($FA). For every command or argument byte except the Resend ($FE) command, the response always begins with an "Acknowledge" or ACK ($FA) byte. See section 2.3.2 for more information.

$F6 **Set Defaults.** Restores conditions to the initial power-up state. This resets the sample rate, resolution, scaling, and Stream mode to the same states as for the Reset ($FF) command, and disables the TouchPad. This command disables Absolute mode, but it leaves the rest of the TouchPad mode byte unaffected.

$F5 **Disable.** Disables Stream mode reporting of motion data packets. All other TouchPad operations continue as usual.

$F4 **Enable.** Begins sending motion data packets if in Stream mode. To avoid undesirable bus contention, driver software should send the Enable as the very last command in its PS/2 initialization sequence.

Note that a PS/2 TouchPad includes two distinct state bits: the enable/disable flag controlled by commands $F4 and $F5, and the Stream/Remote flag controlled by commands $EA and $F0. These two flags are independent, and both must be set properly (enabled, Stream mode) for the TouchPad to send motion packets. The intention is that disabled Stream mode means the host is not interested in motion packets, while Remote mode means the host plans to poll explicitly for motion data. In practice, Remote mode and disabled Stream mode are identical in the Synaptics TouchPad.

$F3 **Set Sample Rate.** Followed by one argument byte, this command sets the PS/2 "sample rate" parameter to the specified value in samples per second. Legal values are 10, 20, 40, 60, 80, 100, and 200 (decimal) samples per second.

The Set Sample Rate command is a two-byte command. The command byte and argument byte each receive an ACK ($FA) from the TouchPad. Thus, a complete Set Sample Rate = 10 command consists of $F3 from the host, $FA from the TouchPad, $0A from the host, and $FA from the TouchPad.

The Synaptics TouchPad records the sample rate argument and will respond properly to a later Status Request ($E9) command, but this value does not actually affect TouchPad data reporting. Stream mode reporting occurs at either 40 or 80 samples per second, and is controlled by the *Rate* bit of the TouchPad mode byte; see section 4.3.

Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

$F2 **Read Device Type.** Responds with an ACK ($FA) followed by a $00 TouchPad ID byte.

$F0 **Set Remote Mode.** Switches to Remote mode, as distinct from the default Stream mode. In Remote mode, the TouchPad sends motion data packets only in response to a Read Data ($EB) command.

$EE **Set Wrap Mode.** Switches into special "echo" or Wrap mode. In this mode, all bytes sent to the TouchPad except Reset ($FF) and Reset Wrap Mode ($EC) are echoed back verbatim.

$EC **Reset Wrap Mode.** If the TouchPad is in Wrap mode, it returns to its previous mode of operation, except that Stream mode data reporting is disabled. If the TouchPad is not in Wrap mode, this command has no effect.

$EB **Read Data.** The TouchPad replies with an ACK ($FA) followed by a three- or six-byte motion data packet. This command is meant to be used in Remote mode (see command $F0), though it also works in Stream mode. In Remote mode, this command is the only way to get a data packet. The packet is transmitted even if no motion or button events have occurred. The host can poll as often as PS/2 bus bandwidth allows, but since the underlying motion data are updated only 40 or 80 times per second (according to the *Rate* bit in section 4.3), there is little point in polling more often than that.

$EA **Set Stream Mode.** Switches to Stream mode, the default mode of operation. In this mode, motion data packets are sent to the host whenever finger motion or button events occur and data reporting has been enabled. Maximum packet rate is governed by the current TouchPad sample rate, described below.

Stream mode is the recommended way to use a Synaptics TouchPad; nearly all PC-compatible computers operate their pointing devices in Stream mode.

$E9 **Status Request.** Responds with an ACK ($FA), followed by a 3-byte status packet consisting of the following data:

	Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Byte 1	0	Remote	Enable	Scaling	0	Left	Middle	Right
Byte 2	0	0	0	0	0	0	Resolution	
Byte 3	Sample rate							

Figure 4-1. Status request response

Remote: 1 = Remote (polled) mode, 0 = Stream mode.

Enable: 1 = Data reporting enabled, 0 = disabled. This bit only has effect in Stream mode.

Scaling: 1 = Scaling is 2:1, 0 = scaling is 1:1. See commands $E6 and $E7 below.

Left: 1 = Left button is currently pressed, 0 = released.

Middle: 1 = Middle button is currently pressed, 0 = released.

 Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

Right: 1 = Right button is currently pressed, 0 = released.

Resolution: The current resolution setting, from 0 to 3 as described under Set Resolution ($E8) below.

Sample rate: The current sample rate setting, from 10 to 200 as described under Set Sample Rate ($F3) above.

For example, after Reset or Set Defaults, a Status Request command will return the bytes

$FA $00 $02 $64

indicating no buttons pressed, Stream mode, Disabled mode, Scaling 1:1, Resolution $02, and Sample rate $64 = 100 decimal.

The Status Request command returns different data in the context of a TouchPad special command sequence; see sections 4.2 and 4.3.

$E8 **Set Resolution.** Followed by one argument byte, this command sets the PS/2 resolution parameter. Legal argument values are $00, $01, $02, and $03, corresponding to resolutions of 1, 2, 4, and 8 counts per mm, respectively.

The Synaptics TouchPad records the resolution argument and will respond properly to a later Status Request ($E9) command, but this value does not actually affect TouchPad data reporting. Sections 3.1.1 and 3.2.4, as well as the results from the Read Model ID query described in section 4.4, describe the actual resolution reported by the TouchPad.

$E7 **Set Scaling 2:1.** Sets the PS/2 scaling bit to enable a non-linear motion gain response. The Synaptics TouchPad records this value and will respond properly to a later Status Request ($E9) command, but this value does not actually affect TouchPad data reporting. The Set Scaling commands have special meaning for TouchPads using the transparent Pass-Through (GlassPad) feature; see section 5.3 for details.

$E6 **Set Scaling 1:1.** Clears the PS/2 scaling bit, as described above. The Set Scaling commands have special meaning for TouchPads using the transparent Pass-Through (GlassPad) feature; see section 5.3 for details.

4.2. TouchPad special command sequences

The standard PC BIOS does not allow system software to send arbitrary command bytes to a PS/2 pointing device. In fact, the BIOS supports only a subset of the commands listed in section 4.1. To be compatible with the BIOS, the Synaptics PS/2 TouchPad must express all TouchPad-specific information queries and other operations using only combinations of those commands that are supported by the BIOS. These combinations of commands are called *special command sequences*. They are designed to be relatively concise while still being distinctive enough so that non-Synaptics-aware drivers will not accidentally activate them. These special command sequences extend the command set to support TouchPad queries and extended capability bits, and the mode byte (and the Absolute mode and W mode contained within the mode byte).

Each TouchPad special command sequence consists of four Set Resolution ($E8) commands which together encode an 8-bit argument value, followed immediately by a Set Sample Rate ($F3) or Status

Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

Request ($E9) command. If the final command is not preceded by *exactly* four Set Resolution commands, it has only its usual effect as described in section 4.1 (either setting the sample rate or producing a standard status report. Note that neither the “resolution” nor the “sample rate” controlled by these PS/2 commands actually affect the Synaptics TouchPad’s pointing behavior.). When sending a special command sequence, precede the sequence with an “inert” command such as a second Disable or Set Scaling 1:1, just in case the most recent command sent to the device was a (fifth) Set Resolution.

The four Set Resolution commands encode an 8-bit argument by concatenating their individual 2-bit resolution arguments. If the four commands are

$E8 *rr* $E8 *ss* $E8 *tt* $E8 *uu*

where *rr, ss, tt,* and *uu* are numbers in the range $00–$03, then the full 8-bit argument for the special command sequence is

$(rr \times 64) + (ss \times 16) + (tt \times 4) + uu.$

Special command sequences are used for mode byte configurable features, described in section 4.3, and for information queries, described in section 4.4.

4.3. Mode byte

The Synaptics TouchPad has a small set of configurable features that are encapsulated in the *TouchPad mode byte,* an 8-bit field which the host can set to any value using a special command. If a Set Sample Rate 20 ($F3, $14) command is preceded by four Set Resolution commands encoding an 8-bit argument, the 8-bit argument is stored as the new value for the TouchPad mode byte.

For example, to set the mode byte to $C1 (Absolute mode, high packet rate, W mode enabled) one would use the sequence of commands,

$E8 $03 $E8 $00 $E8 $00 $E8 $01 $F3 $14

where the argument $C1 is encoded as follows:

$(\$03 \times 64) + (\$00 \times 16) + (\$00 \times 4) + \$01 = \$C1.$

All ten command and argument bytes receive the usual ACK ($FA) acknowledgments. Note that, as described at the beginning of section 4.1, it is important to ensure that the device is disabled ($F5) before sending this command sequence. To receive Absolute mode packets, follow this sequence with an Enable ($F4) command.

The power-on initial setting for the mode byte is $00. During normal operation, the mode bits are generally preserved except when explicitly changed by a host command. The PS/2 Reset ($FF) and Set Defaults ($F6) commands clear the *Absolute* bit to 0 but do not affect the other mode bits.

 Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

The mode byte is arranged as follows:

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Absolute	Rate	Transparent Mode	Guest ACPI Mode	Sleep	DisGest or EWmode	PackSize	Wmode

Figure 4-2. TouchPad mode byte

Some mode bits are "reserved" in some product configurations. The host should ignore the values of reserved bits when reading the mode byte. The host may either set the reserved bits always to zero, or preserve the last-read values of the reserved bits, when changing the mode byte; the host must not change a reserved bit from 0 to 1.

Absolute (bit 7)

This bit is 0 to select Relative (mouse-compatible) mode, or 1 to select Absolute mode. See sections 3.1 and 3.2.

Rate (bit 6)

The TouchPad typically generates 40 or 80 packets per second:

- 80 packets per second, also known as *full speed*, is the default. The higher packet rate is preferable because it leads to the smoothest cursor motion.
- 40 packets per second, also known as *half speed*, is used for slower hosts that cannot keep up with 80 packets per second. Also, the low packet rate mode does more internal data filtering and so may perform better in environments of extreme electrical noise.

This bit is 0 to select half speed, or 1 to select full speed. The default setting for this bit in most TouchPad devices is 1. This bit is valid in both Relative and Absolute mode, although it may not be supported in all Synaptics products.

Transparent Mode (bit 5)

This bit is reserved, except for TouchPads with the Pass-Through option where this bit allows information to pass directly from guest to host. See section 5, Pass-Through Option, for more information.

Guest ACPI Mode (bit 4)

This bit is reserved, except for TouchPads with the Pass-Through option. See section 5, Pass-Through Option, for more information.

Sleep (bit 3)

The *Sleep* bit is 0 for normal operation, or 1 for Sleep mode. When Sleep mode is enabled, the sensor goes into a low-power idle state in which it ignores finger activity. In Sleep mode, only button clicks cause the TouchPad to generate a motion packet. When the *Sleep* bit is changed from 1 to 0, the TouchPad may need to spend 300–1000ms recalibrating before finger sensing resumes. The Synaptics drivers use Sleep mode for ACPI power management support. The *Sleep* bit is defined on TouchPads whose *capSleep* capability bit is set. In TouchPads without this capability, the bit is reserved and should be left at 0. In some products, the *Sleep* bit affects only Relative mode; in others, it affects both Relative and Absolute modes. Note that *DisGest* should be set whenever *Sleep* is set.

Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

DisGest or *EWmode* (bit 2)

This bit has two different meanings, depending on whether or not *Gesture* is reported:
- if the *Wmode* bit is set to 1, *Gesture* is not reported and bit 2 automatically reports *EWmode* (also known as Extended W mode). See section 3.2.9 for more information about Extended W mode.
- If the *Wmode* bit is not set (0), then *Gesture* is reported and bit 2 refers to *DisGest*. In this case, the bit is 0 to enable tap and slide gesture processing, or 1 to disable detection of tap and slide gestures. When this bit is 1, the Relative mode mouse packet reports the true physical button states, and the Absolute mode packet's *Gesture* bit always reports as zero.

This bit is implemented only for 4.*x* and later TouchPads (i.e., when *infoMajor* ≥ 4). For older TouchPads, the bit is reserved.

PackSize (bit 1)

PackSize determines the expected packet size of guest packets.

Wmode (bit 0)

This bit is 0 to select simple Absolute mode packets, or 1 to select enhanced Absolute packets that contain the W value as well as X, Y, and Z. See section 3.2.1 for more information about these packet formats. This bit is defined only in Absolute mode on pads whose *capExtended* capability bit is set. In Relative mode and in TouchPads without this capability, the bit is reserved and should be left at 0.

The following table shows some typical values for the mode byte in the PS/2 protocol:

Value (hex)	*When to use*	*Effect*
$00	Always OK	Relative mode
$04	Always OK	Relative mode with gestures disabled
$40	Always OK	Relative mode with high packet rate
$80	*capExtended* = 0	Absolute mode
$81	*capExtended* = 1	Absolute mode with W
$C0	*capExtended* = 0	Absolute mode with high packet rate
$C1	*capExtended* = 1	Absolute mode with W, high packet rate
$0C	*capSleep* = 1	Low-power sleep mode

Figure 4-3. PS/2 mode byte values

4.4. Information queries

The host can query the TouchPad for information describing the size, model, and capabilities of the TouchPad.

If a Status Request ($E9) command is preceded by four Set Resolution commands encoding an 8-bit argument, then the 3-byte packet that is returned takes a special form where the three bytes encode special information chosen by the 8-bit argument. In some cases, the middle byte (normally the current resolution from $00 to $03) is replaced by a constant $47 byte which can be used to verify that the special command sequence was recognized. The 8-bit argument selects one of the following queries:

 Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

$00 **Identify TouchPad.** The Identify TouchPad query returns the following information to the host:

infoMajor
The primary or "major" version of the TouchPad device and firmware.

infoMinor
The minor version number starts over at 0 with each new major version, and increases by one whenever changes are made to the device or its firmware that are significant, yet compatible with previous versions of the firmware. For example, in a complete version number such as 8.5, the major version is 8 and the minor version is 5.

infoModelCode
This 4-bit field encodes very limited information about the TouchPad model. It is provided for compatibility only; this field is set at 1 for all current devices. New host software should not use the *infoModelCode* field.

The first byte of the response is the minor version number infoMinor. The middle byte is the constant $47. The third byte encodes the major version number infoMajor in the low 4 bits, and the (obsolete) infoModelCode in the upper 4 bits.

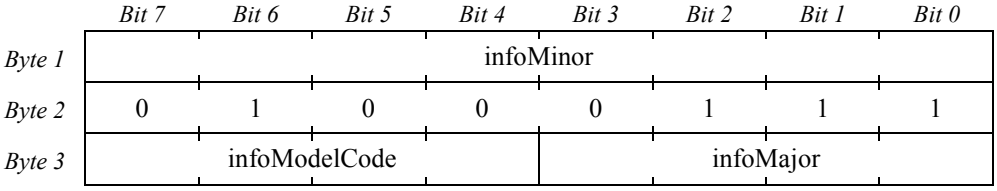


Figure 4-4. PS/2 Identify TouchPad response

All TouchPads ever shipped by Synaptics have supported the Identify TouchPad query. To check whether a PS/2 pointing device is a Synaptics TouchPad, send four Set Resolution 0 commands followed by a Status Request command,

$E8 $00 $E8 $00 $E8 $00 $E8 $00 $E9

and look at the second byte of the three-byte Status response. If the second byte is $47, the device is a Synaptics TouchPad. For non-Synaptics devices, the second byte will instead report the current resolution (which in this case would be $00).

$01 **Read TouchPad Modes.** Beginning with firmware version 7.5, this query has been redefined. See query $00 to obtain the firmware version information.

For firmware version 7.5 and above, this query contains the model number and the TouchPad mode byte. If the module supports a guest PS/2 device, Byte 1 bit 0 reports whether the guest is present.

Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

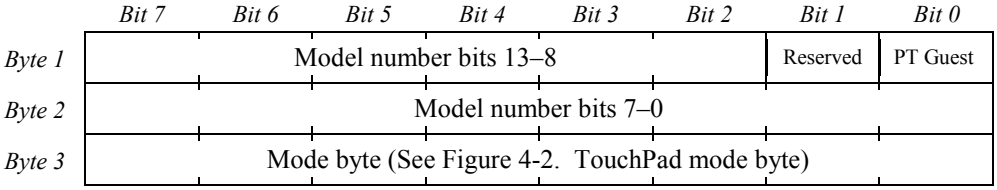


Figure 4-5. PS/2 Read Modes response for firmware version 7.5 and above

For firmware versions prior to 7.5, the first two bytes of the response are the constants $3B and $47, respectively, and the third byte is the TouchPad mode byte.

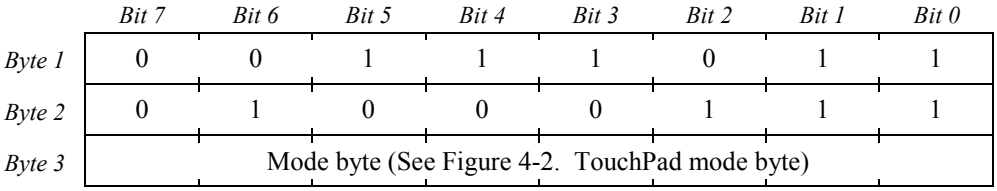


Figure 4-6. PS/2 Read Modes response for firmware versions prior to 7.5

Note: For a Pass-Through TouchPad with firmware versions prior to 7.5, Byte 1 = $00 if the guest is not present, and Byte 1 = $01 if the guest is present. For non-Pass-Through TouchPads, Byte 1 = $3B as documented above. See Section 5 for more information on the Pass-Through option. For more recent firmware versions, the guest presence is reported in bit 16.

$02 **Read Capabilities.** Beginning with firmware version 7.5, this query has been redefined. See query $00 to obtain the firmware version information.

For firmware version 7.5 and above, the first and third bytes hold the extended capability bits. The second byte is the model sub-number.

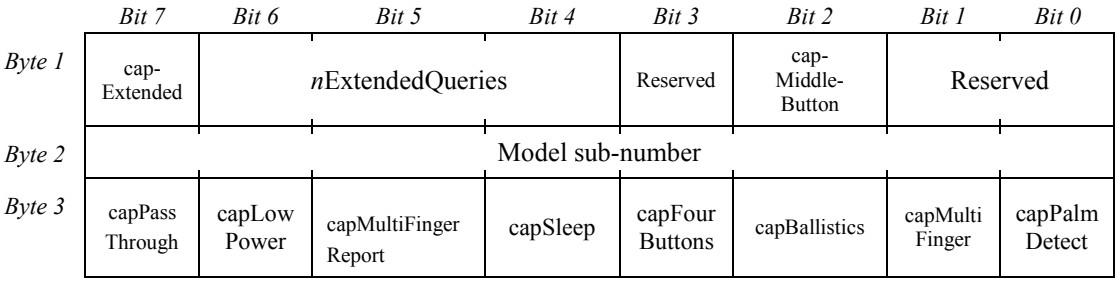


Figure 4-7. PS/2 Read Capabilities response for firmware version 7.5 and above

 Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

For firmware versions prior to 7.5, the first and third bytes hold the extended capability bits. The second byte is the constant $47.

	Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Byte 1	cap-Extended	*n*ExtendedQueries			Reserved	cap-Middle-Button	Reserved	
Byte 2	0	1	0	0	0	1	1	1
Byte 3	capPass Through	capLow Power	capMultiFinger Report	capSleep	capFour Buttons	capBallistics	capMulti-Finger	capPalm Detect

Figure 4-8. PS/2 Read Capabilities response for firmware versions prior to 7.5

Synaptics TouchPads support an extended capability query which reports to the host the presence or absence of various advanced features.

capExtended (bit 23)

This bit is set if the extended capability bits are supported. The host can examine this bit to see whether the other 15 extended capability bits are present and meaningful. The *capExtended* bit also signifies that the TouchPad supports W mode as described in sections 3.2.6 and 4.3.

nExtendedQueries[2..0] (bits 22..20)

These bits represent the number of extended queries the TouchPad supports. Taken together they form a 3-bit number which encodes the number of extended query pages that the TouchPad can report. If *nExtendedQueries* is 1 or greater, then the maximum meaningful query is $08 plus *nExtendedQueries*.

Note: *capFourButtons* and *capMiddleButton* / *nExtendedButtons* are mutually exclusive. If a TouchPad reports *capFourButtons* set then *capMiddleButton* will be clear and *nExtendedButtons* will be zero. If *capMiddleButton* is set and/or *nExtendedButtons* is non-zero, then the multi-button option is set.

capMiddleButton (bit 18)

This bit is set if the TouchPad has a middle button. If this bit is set, the TouchPad reports the middle button state in the appropriate place in the Relative mode packet, and in the Absolute mode packet (when Wmode=1). The middle button is not considered to be an extended button.

capPassThrough (bit 7)

This bit is set if the TouchPad uses the Pass-Through option. See section 5 for a complete description.

capLowPower (bit 6)

This feature allows the TouchPad to automatically reduce power when the device is not in use. Because it is automatic, there is no mode bit to control this option.

capMultiFingerReport (bit 5)

The *capMultiFingerReport* bit is set if the TouchPad supports Extended W mode and is able to report two finger locations in interleaved packets, as described in section 3.2.9 and in section 3.2.9.2.

Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

capSleep (bit 4)

The *capSleep* bit is set if sleep mode is supported. See the discussion of the Sleep bit in section 4.3.

capFourButtons (bit 3)

This bit is set if the TouchPad is a MultiSwitch that supports four mouse buttons labeled Left, Right, Up, and Down. The Up and Down buttons are reported only during Absolute mode with the *Wmode* bit set. A four-button TouchPad takes the current value to report for the two extra buttons (Up and Down) and XORs them with the current value of the Right and Left buttons, as shown in Figure 3-4. The driver can decode the state of the two extra buttons by reversing this operation. Note that many devices are multi-button, rather than four-button. See Figure 4-16 for a description of the *nExtendedButtons* query.

capBallistics (bit 2)

This bit is only used by TouchStyks, which are described separately in the *Synaptics TouchStyk Interfacing Guide* (PN 511-000003-01).

capMultiFinger (bit 1)

This bit is set if multi-finger detection is supported. The TouchPad is then able to count the number of simultaneous fingers on the sensor and report the finger count via the W field of the Absolute packet. If this bit is 0, the TouchPad does not support multi-finger detection; any finger contact will be assumed to be a single finger. If *capPalmDetect* is set and multiple fingers are present, W reports a large width for the assumed single finger.

capPalmDetect (bit 0)

This bit is set if palm detection is supported. In W mode, the TouchPad measures the apparent size or width of the finger and reports the width in the W field of the Absolute mode packet. The host can use this information to help distinguish between intentional finger contact and accidental palm or hand contact.

Reserved

Any capability bits not described above are reserved for future use. The host should ignore the values of reserved bits when reading the capability bits.

\$03 **Read Model ID.** The model ID query allows the host to learn information about the physical type of the TouchPad. The model ID consists of 24 bits divided into various bit-fields, as shown in the figure below.

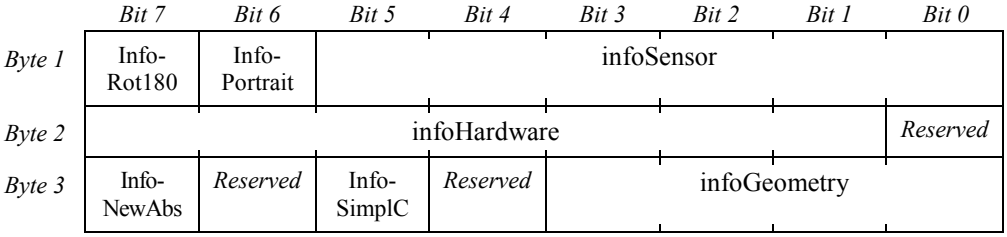


Figure 4-9. PS/2 Read Model ID response bits

The model ID fields are defined as follows:

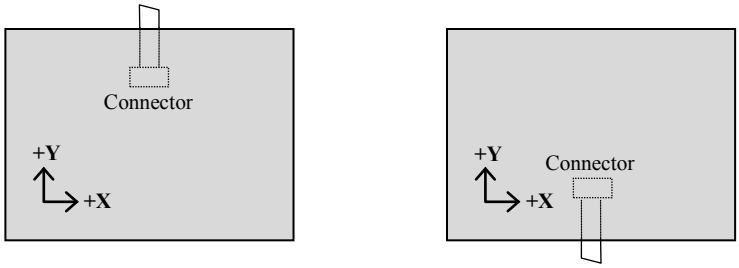
infoRot180 (bit 23)

This bit is 0 for typical “Up” TouchPads, or 1 for 180°-reversed “Down” TouchPads

 Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

designed to be mounted upside-down. (The X/Y coordinate system as experienced by the user is the same in both cases; thus, host software generally will not care about the *infoRot180* bit.)

The Up orientation is defined as the orientation in which the cable exits upwards from under the board. On most models, this is also the orientation with the connector near the top edge of the underside of the board. The Down orientation uses a physically identical board that is programmed to work properly when mounted with the cable exiting downwards.



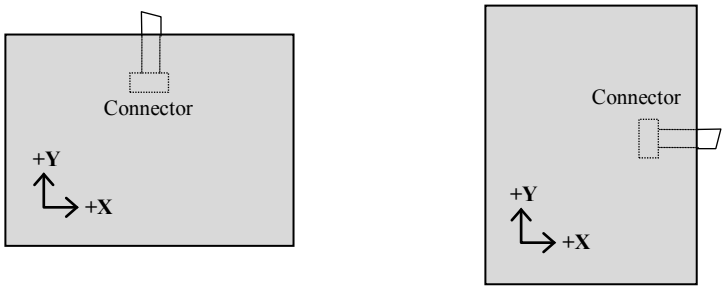
(a) Up
infoRot180 = 0

(b) Down
infoRot180 = 1

Figure 4-10. Up and Down orientations

infoPortrait (bit 22)

This bit is 0 for normal (landscape) TouchPads, or 1 for 90°-rotated (portrait) TouchPads in which the X and Y coordinates still represent the user's horizontal and vertical axes, respectively, but the TouchPad is oriented so that it is taller than it is wide. Hence, the typical bezel limits and typical edge margins of section 3.2.3 are approximately interchanged X-for-Y on a Portrait pad.



(a) Landscape
infoPortrait = 0

(b) Portrait
infoPortrait = 1

Figure 4-11. Portrait orientation

Note that the *infoRot180* and *infoPortrait* bits can be considered together as a two-bit field specifying a clockwise rotation of the TouchPad in multiples of 90°.

infoSensor (bits 21–16)

This 6-bit field identifies the type or model of TouchPad sensor; it allows the host to

Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

determine the size and physical type of the TouchPad. If Query $09 is present, it supplies two more bits for a logical 8-bit InfoSensor.

infoHardware (bits 15–9)
This 7-bit field is reserved for use by Synaptics.

infoNewAbs (bit 7)
This bit indicates the modern Absolute packet format is available. It is 1 except for certain very old PS/2 TouchPads.

infoSimpleCmd (bit 5)
This bit is 1 on all modern TouchPads.

infoGeometry (bits 3–0)
This 4-bit field identifies unusual sensor arrangements such as non-rectangular or non-flat TouchPads. This field is independent from the *infoSensor* type. For example, a sensor board might be mounted beneath a round bezel, in which case this custom TouchPad would identify the *infoSensor* type as 1 but the *infoGeometry* type as 2. The host driver and the TouchPad itself might then use the fact that *infoGeometry* = 2 to provide a round EdgeMotion zone instead of the usual rectangular zone.

infoGeometry	*Definition*
0	Unknown shape
1	Rectangle
2	Round
3	Rounded rectangle
4	Race track shape
5–15	*Reserved*

Figure 4-12. Geometry types

$04 Reserved

$05 Reserved

$06 **Read Serial Number Prefix.** This query returns the first twelve bits (bits 35–24) of the TouchPad's unique serial number. The serial number consists of a total of 36 bits, which are reported to the host in the form of a 12-bit prefix and a 24-bit suffix. Synaptics does not specify the internal structure of these 36 bits, but Synaptics does guarantee that the complete 36-bit serial number will be unique among all TouchPads produced. Synaptics does *not* guarantee that the serial numbers will be consecutive or otherwise related, even within the same manufacturing lot.

For TouchPads that have not been serialized, the serial number query will return a default value of zero. The bits shown as "—" in the figure are reserved and hold undefined data.

 Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

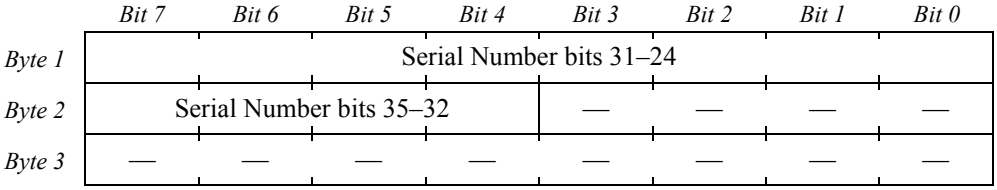


Figure 4-13. PS/2 Serial Number Prefix response

Note that bits 35–24 are guaranteed to be not all zero in a valid serial number, and are all zero for unserialized TouchPads. The host should consider the serial number to be valid only if this query returns non-zero data for bits 35–24.

\$07 **Read Serial Number Suffix.** This query returns the remaining 24 bits of the serial number. The results from this query are undefined if the Serial Number Prefix query returned zero for bits 35–24.

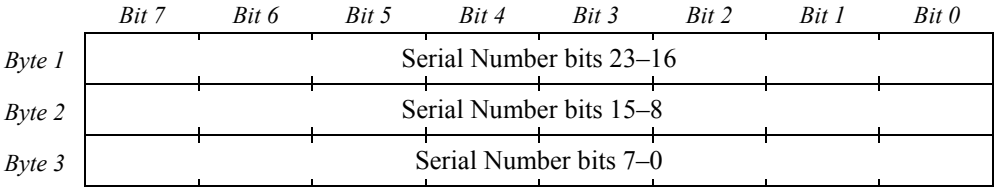


Figure 4-14. PS/2 Serial Number Suffix response

\$08 **Read Resolutions.** See section 3.2.4. This query returns the X and Y coordinate resolutions in Absolute units per millimeter. The second byte of the response is undefined except for the most significant bit, which is guaranteed to be 1.

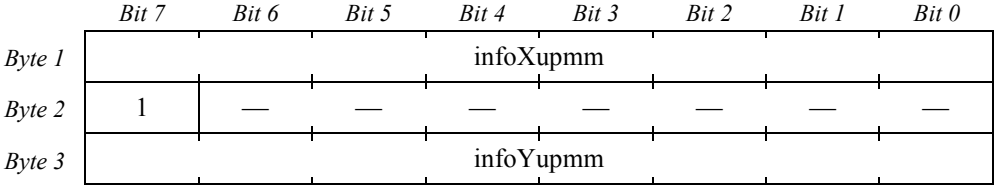


Figure 4-15. PS/2 Read Resolutions response

\$09 **Extended Model ID**. This query returns the product ID, additional capability bits, and additional bits to widen the *infoSensor* field. *infoSensor* is a part of the Model ID query (query \$03). The Extended Model ID query is only present if *nExtendedQueries* is greater than or equal to 1.

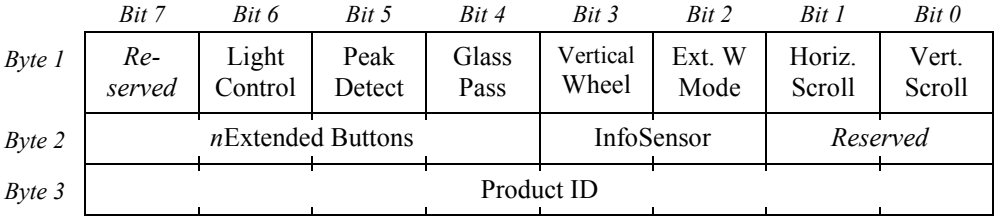


Figure 4-16. PS/2 Extended Model ID response

The extended model ID fields are defined as follows:

Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

Light Control (bit 22)
This bit is set to 1 for LuxPads. For all other TouchPads, this bit is set to 0.

Peak Detect (bit 21)
This bit is set to 1 for newer-model TouchPads. See section 1.1.2 for more information on the difference between newer-model TouchPads and centroid TouchPads. For centroid TouchPads, this bit is set to 0.

Glass Pass (bit 20)
This bit is set to 1 for TouchPads that support the transparent Pass-Through extension (see Section 5.3 for more information). For all other TouchPads, this bit is set to 0.

Vertical Wheel (bit 19)
This bit is set to 1 for TouchPads that have a physical wheel connected. For all other TouchPads, this bit is set to 0.

Extended W Mode (bit 18)
The *capEWmode* bit is set to 1 for TouchPads that support reporting an encapsulated data packet in Absolute mode, indicated by a W value of 2. See section 3.2.6 for more information.

Horizontal Scroll (bit 17)
This bit is set to 1 for virtual scrolling on the bottom edge of the TouchPad with the scrollzone separated from the rest of the TouchPad as a separate opening in the bezel.

Vertical Scroll (bit 16)
This bit is set to 1 for virtual scrolling on the right edge of the TouchPad with the scrollzone separated from the rest of the TouchPad as a separate opening in the bezel.

nExtendedButtons (bits 15-12)
These bits are used to identify the number of extended buttons supported by the TouchPad. See Section 3.2.8 for more information.

InfoSensor (bits 11-10)
These bits are used to widen the InfoSensor field of Read Model ID, query $03.

Product ID (bits 7-0)
These bits contain the TouchPad product ID.

$0A Reserved

$0B Reserved

$0C **Continued Capabilities**

 Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 1	Covered Pad Gest	MultiFinger Mode		ClkPad Bit 0	Advanced Gestures	ClearPad	Reports Max	TB Adj Thresh
Byte 2	Reserved	InterTouch	Reports Min	Uniform ClickPad	Reports V	No Abs Pos Filt	Deluxe LEDs	ClkPad Bit 1
Byte 3	InterTouch I^2C address							

Figure 4-17. PS/2 Continued TouchPad Capabilities

The continued capabilities fields are defined as follows:

Covered Pad Gesture (bit 23)

This TouchPad supports the Covered Pad Gesture. When the TouchPad is determined to be substantially covered, it reports this condition as a 15 in the W field. When this feature is present, the maximum finger width reported will be 14.

MultiFinger Mode (bits 22-21)

This 2-bit field encodes which MultiFinger algorithms are included in the TouchPad.

00 = MF 1.0
01 = MF 1.5
10 = MF 1.25
11 = Reserved

ClickPad bit 0 (bit 20)

This is the low bit, along with bit 8, of the 2-bit ClickPad Type field.

00 = Not a ClickPad
01 = One-button ClickPad
10 = Two-button ClickPad
11 = Reserved

Advanced Gestures (bit 19)

This bit reports whether the TouchPad supports Advanced Gestures.

ClearPad (bit 18)

This bit reports whether this TouchPad is a ClearPad.

Reports Max Coordinates (bit 17)

This bit reports whether query $0D, Maximum Coordinates, is supported.

TouchButton Adjustable Threshold (bit 16)

If this TouchPad includes TouchButtons, their touch threshold is adjustable.

InterTouch (bit 14)

This TouchPad supports InterTouch communication protocol.

Reports Min Coordinates (bit 13)

This bit reports whether query $0F, Minimum Coordinates, is supported.

Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

Uniform ClickPad (bit 12)

If this is a ClickPad, this bit describes the mechanical depression method used by the module.

0 = Hinged mechanism
1 = Uniform mechanism

Reports V (bit 11)

This device supports reporting the V field during multi-finger packets. See 3.2.10.

No Absolute Position Filter (bit 10)

This bit indicates that the TouchPad does not apply a position filter to the absolute postion data prior to reporting it.

Deluxe LEDs (bit 9)

This TouchPad supports the Deluxe LED query (query $0E) and the Deluxe LED command set.

ClickPad bit 1 (bit 8)

This is the high bit, along with bit 20, of the 2-bit ClickPad Type field. See *ClickPad bit 0* (bit 20) for more information.

InterTouch I^2C address (bits 7..0)

If the InterTouch bit is set (bit 14), this is the InterTouch I^2C address.

Reserved

Any capability bits not described above are reserved for future use. The host should ignore the values of reserved bits when reading the capability bits.

$0D **Maximum Coordinates**

If the ReportsMax capability bit is set, then this query will return the largest X and Y coordinates that the TouchPad should generate.

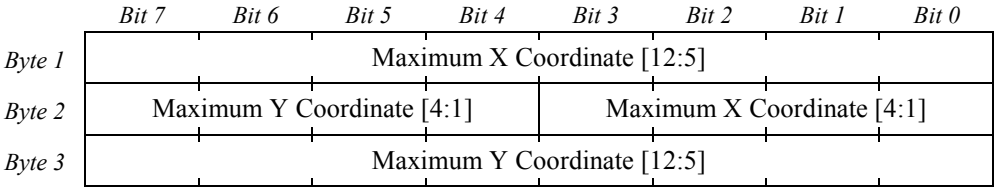


Figure 4-18. PS/2 Maximum Coordinates

$0E **Deluxe LED Info**

If the Deluxe LEDs capability bit is set, then this query will return information on the number of LEDs present and the number of built-in LED animation sequences.

 Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

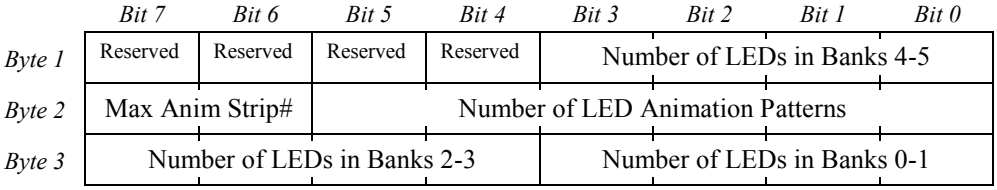


Figure 4-19. PS/2 Deluxe LED Information

Number of LEDs in Banks 4–5 (bits 19-16)

This field returns the number of LEDs implemented in Banks 4–5.

Max LED Animation Strip # (bits 15-14)

If the device supports LED Animations (*Number of LED Animation Patterns* > 0), this field returns the highest addressable LED strip that can be used by the LED Animate command

Number of LED Animation Patterns (bits 13-8)

Returns the number of LED animation patterns that are built into this device. If no LED animations are supported, this field will return 0.

Number of LEDs in Banks 4–5 (bits 7-4)

This field returns the number of LEDs implemented in Banks 2–3.

Number of LEDs in Banks 4–5 (bits 3-0)

This field returns the number of LEDs implemented in Banks 0–1.

other For any other value of the 8-bit argument, the values of the three result bytes are undefined.

$0F **Minimum Coordinates**

If the ReportsMin capability bit is set, then this query will return the smallest X and Y coordinates that the TouchPad should generate.

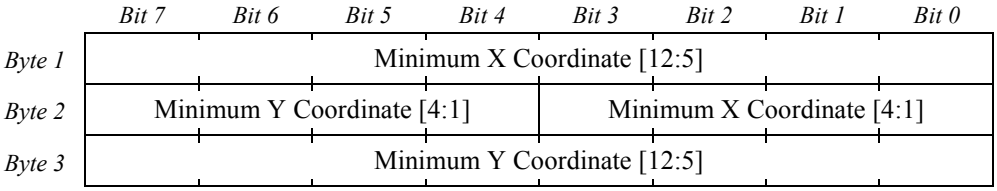


Figure 4-20. PS/2 Maximum Coordinates

Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

5. PS/2 Pass-Through Option

This section describes the Pass-Through option to the PS/2 protocol. This option is used to implement Dual Pointing solutions including a Synaptics TouchStyk or third-party pointing stick along with the Synaptics TouchPad.

In a Pass-Through system, the host connects to the TouchPad by a standard PS/2 port. The TouchPad is the master device in this context. The pointing stick, referred to as the guest device, connects to a second PS/2 port on the TouchPad master. The master is responsible for passing commands from the host to the guest, and for passing packet data from the guest to the host, as well as for all its normal TouchPad functions.

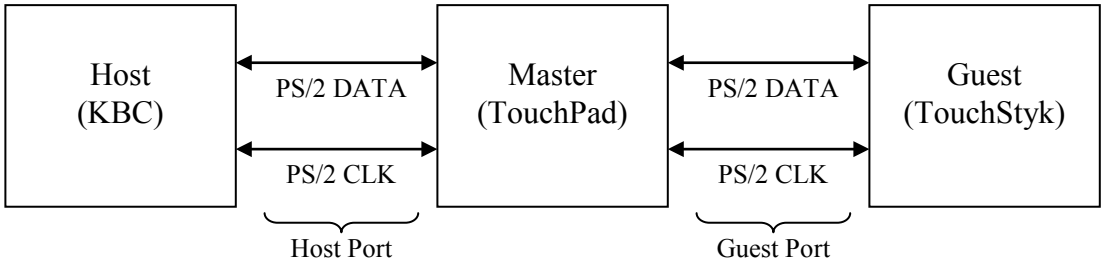


Figure 5-1. Pass-Through configuration

5.1. Host port communication

For PS/2 compatibility, the master appears to the host as a standard Synaptics TouchPad with certain PS/2 protocol extensions. All PS/2 timing requirements are observed. Like the standard TouchPad, the master is backward-compatible with a PS/2 mouse.

The master provides pull-ups of approximately 10KΩ on the CLK and DATA wires of its host port.

5.1.1. Operating modes

Synaptics TouchPads provide two operating modes, Relative and Absolute. A TouchPad serving as a Pass-Through master continues to support these two operating modes. In Relative mode, the master appears to the host to be a standard PS/2 mouse. The master also operates the guest as a standard PS/2 mouse. In Absolute mode, the master provides individual access to itself and the guest.

In Relative mode, the device simulates the command set and packet formats of a standard PS/2 mouse:

- The TouchPad essentially does its own “hidden multiplexing.”
- Guest deltas are added into TouchPad deltas.
- Guest buttons are mixed with TouchPad buttons.
- Special support is required for a guest wheel mouse.

Upon power-on reset, the master defaults to Relative mode. The master will reset the guest to ensure that the guest is also in its mouse-compatible or Relative mode.

 Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

In Absolute mode:

- The device uses a six-byte packet format that includes complete information about the state of the finger on the device.
- Three- and four-byte guest packets are encapsulated into 6-byte packets.
- Encapsulated packets have a special W value. Guest data are stuffed into X, Y, and Z fields.
- The TouchStyk supports a special 4-byte Absolute mode for use as a guest.

This absolute information requires considerable post-processing to emulate a mouse, so the TouchPad presumes that its Absolute mode will be used only by a Synaptics driver, or by a specialized host in a non-PC application.

For configuration, a TouchPad serving as a Pass-Through master may also support a transparent Pass-Through mode. This mode sends commands directly to the device, and speeds up the initialization process. For more information on transparent Pass-Through, including how to switch between normal and transparent Pass-Through modes, see section 5.3.

5.1.2. PS/2 command set

The master receives commands from the host and sends responses to the host in the usual way. The following notes describe additional behaviors of these commands in the Pass-Through setting.

Note that host commands are *not* propagated to the guest except as noted below.

$FF **Reset.** The master acknowledges with $FA to the host immediately, then with $AA $00 within approximately 350 ms. Shortly after sending the $FA to the host, the master begins the process of resetting and initializing the guest as described in section 5.2.2. Note that the timing of the $FA and $AA $00 responses to the host is independent of the interaction with the guest.

$FE **Resend.** Handled locally to the master; there is no special Pass-Through behavior aside from an additional packet format (the encapsulation packet) that may be re-sent when appropriate.

$F6 **Set Defaults.** Shortly after acknowledging the host with $FA, this command resets and initializes the guest as described in section 5.2.2.

$F5 **Disable.** The enable/disable state of the system is held in the master. However, if a guest is present, the master propagates the Disable command to the guest after sending its acknowledge ($FA) to the host. If for some reason the guest sends bytes to the master when disabled, the master discards the byte (if in Relative mode) or sends them to the host as encapsulated response bytes (if in Absolute mode).

$F4 **Enable.** The Enable command is propagated to the guest if present; see $F5 above.

$F3 **Set Sample Rate.** Handled locally to the master and not propagated to the guest; there is no special Pass-Through behavior except for an additional TouchPad special command sequence using this command (as described in section 5.1.3.2.).

Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

$F2 **Read Device Type.** Handled locally; there is no special Pass-Through behavior.

$F0 **Set Remote Mode.** Handled locally; this command is not propagated to the guest. If the master is in remote mode and the guest is enabled, the guest sends packets to the master but the master does not forward the packets to the host. In Relative (remote) mode, the master adds the motion and button information from the guest to its own motion deltas and button states for reporting in an $EB (Read Data) command. In Absolute (remote) mode, guest packets are simply discarded.

$EE **Set Wrap Mode.** Handled locally; there is no special Pass-Through behavior.

$EC **Reset Wrap Mode.** Handled locally; there is no special Pass-Through behavior.

$EB **Read Data.** Handled locally to the master; there is no special Pass-Through behavior. This command always forces a master (TouchPad) packet, not a guest packet. However, see notes above for $F0 (Remote Mode).

$EA **Set Stream Mode.** Handled locally; there is no special Pass-Through behavior.

$E9 **Status Request.** Handled locally; there is no special Pass-Through behavior. The button bits report the TouchPad button states only.

$E8 **Set Resolution.** Handled locally to the master and not propagated to the guest; no special Pass-Through behavior except for an additional TouchPad special command sequence using this command (as described in section 5.1.3.2.).

$E7 **Set Scaling 2:1.** Handled locally; there is no special Pass-Through behavior unless the TouchPad is using the transparent Pass-Through (GlassPass) mode. See section 5.3 for details.

$E6 **Set Scaling 1:1.** Handled locally; there is no special Pass-Through behavior unless the TouchPad is using the transparent Pass-Through (GlassPass) mode. See section 5.3 for details.

Invalid command codes or non-Synaptics extensions are not propagated to the guest. However, the host may use the tunneling feature (section 5.1.3.2) to send any desired command code to the guest.

5.1.3. Synaptics PS/2 extensions

This section describes additions to queries and commands described elsewhere in this document. The additions have been made specifically to support Pass-Through operations.

5.1.3.1. Queries

A new *capPassThru* flag is defined as bit 7 of the extended capability bits, as reported by query $02 (Read Capabilities). Pass-Through-enabled master devices report a *capPassThru* value of 1. All other queries operate as described in previous sections of this document. The queries report only the state and properties of the TouchPad (master), not of the guest.

 Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

5.1.3.2. Command tunneling

Pass-Through TouchPads accept a special PS/2 command sequence that forces a command or argument byte to be sent to the guest. This feature is known as *tunneling* a command to the guest.

To tunnel a command, first ensure that the TouchPad is disabled (command $F5), in Stream mode (command $EA, the default), and that the *Absolute* and *Wmode* bits of the TouchPad mode byte are set (bits 7 and 0). Then, encode the desired 8-bit command byte into four Set Resolution commands, and issue a Set Sample Rate with an argument of 40 decimal ($28 hex):

$E8 $*xx* $E8 $*xx* $E8 $*xx* $E8 $*xx* $F3 $28

The TouchPad acknowledges each of these ten bytes with the usual $FA. After the final $FA, the TouchPad sends the encoded byte to the guest. Each response byte from the guest is encapsulated in a six-byte packet as shown in Figure 5-2. For normal guests, the first encapsulated response byte is an $FA. However, the TouchPad does not enforce this in any way.

	Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Byte 1	1	0	0	0	0	1	Right	Left
Byte 2	Response byte							
Byte 3	*Reserved*							
Byte 4	1	1	*Reserved*		0	1	Right	Left
Byte 5	*Reserved*							
Byte 6	*Reserved*							

Figure 5-2. Encapsulated command response byte

See section 5.1.5. for further discussion of encapsulation packets. Note that the state of the PackSize mode bit is irrelevant for encapsulated command response bytes.

The tunneling and response encapsulation mechanism is sufficient for Synaptics' needs and is fully compatible with KBCs that reject non-standard PS/2 transactions. To make communication between the host and the guest faster and more efficient, Synaptics has a transparent Pass-Through mode. See section 5.3 for more information.

5.1.4. Relative mode operation

When the TouchPad is in Relative mode, the guest is assumed also to be in Relative mode. Guest packets are decoded based on the PS/2 standard into X and Y deltas and *Left*, *Middle*, and *Right* button bits. The guest's X and Y deltas are added into the TouchPad's own delta accumulators, and the guest's button states are logically OR'd into the TouchPad's button states. The TouchPad sends the resulting sequence of merged relative packets to the host in the usual way. It is not possible for the host to determine which device (TouchPad or guest) originated a given motion delta or button action.

Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

5.1.4.1. Relative mode support for wheel mice

Synaptics Pass-Through TouchPads do not support wheel mice in Relative mode. If the *PackSize* mode bit is set, the TouchPad expects four-byte packets from the guest. However, there is no provision for forwarding these four-byte packets on to the host, and the fourth byte of each guest packet is discarded.

5.1.4.2. Relative mode support for separate enables

Relative mode has limited support for separately enabling or disabling the TouchPad and guest pointing devices. The pointing and gesturing functionality of the TouchPad can be disabled by setting the *Sleep* and *DisGest* bits of the TouchPad mode byte, as described in section 4.3. This does not affect any mouse buttons attached to the TouchPad; there is no way to disable these buttons short of disabling the TouchPad and guest entirely.

The guest can be disabled by tunneling an $F0 (Set Remote Mode) command to the guest before enabling the TouchPad.

5.1.5. Absolute mode operation

Synaptics TouchPads support two packet formats in Absolute mode. The format is selected by the *Wmode* bit of the TouchPad mode byte.

When *Wmode* = 0, absolute packets use an older format that does not support the Pass-Through feature. In Absolute mode with *Wmode* = 0, guest packets are simply discarded. This mode is not recommended for use by newly-written host software.

When *Wmode* = 1, absolute packets include a four-bit W field. On devices with the *capPassThru* capability bit set, a W value of 3 signifies an *encapsulation* packet. Any packet with W ≠ 3 is a regular TouchPad packet. The state of the guest does not influence these packets. Any packet with W = 3 is an encapsulation packet, as shown in Figure 5-3.

	Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Byte 1	1	0	0	0	0	1	Right	Left
Byte 2	Packet byte 1							
Byte 3	Packet byte 4							
Byte 4	1	1	*Reserved*		0	1	Right	Left
Byte 5	Packet byte 2							
Byte 6	Packet byte 3							

Figure 5-3. Encapsulated data packet

If the PackSize bit is 0, guest packets are assumed to be three bytes long and byte 3 of the encapsulation packet, “Packet byte 4,” is $00.

In an encapsulation packet, the Left and Right bits in bytes 1 and 4 report the state of the TouchPad buttons. The state of the guest buttons is typically in the first byte of the guest packet, reported in byte 2 of the encapsulation packet. The TouchPad and guest buttons are not OR’d together as they are in Relative mode. Indeed, the guest packet is not interpreted in any way.

 Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

If the TouchPad is a four-button or MultiSwitch TouchPad, the button bits in byte 4 of the encapsulation packet are modified somewhat as described in section 3.2.1. Note that in the case of MultiSwitch TouchPads, the M and X bits are reported in encapsulation packets but the extended button bits are not. Packets with W ≠ 3 are fully compatible with the MultiSwitch protocol.

5.1.5.1. Packet interleaving

When the TouchPad and guest each have data to report, the TouchPad interleaves TouchPad and encapsulation packets to the host. The details of the interleaving algorithm are implementation dependent. However, the general goal of interleaving is to ensure that the TouchPad and guest each receive fair access to the host. Because the PS/2 protocol has limited bandwidth, the packet rate seen by each device is reduced considerably (by as much as half) when both devices talk at once.

The TouchPad inhibits the guest, as described in section 5.2.4., to limit it to the available bandwidth.

When only the TouchPad or only the guest is attempting to send data to the host, the sending device can use the full PS/2 bandwidth. It is implementation-dependent whether guest packets are limited to the TouchPad's native packet rate of 40 or 80 packets per second. It is highly recommended that the driver set the rate bit of the TouchPad's mode byte to enable the 80/second packet rate.

5.1.5.2. Absolute mode support for wheel mice

An Absolute mode driver can support Intellimouse-compatible guest devices using command tunneling and the PackSize bit. First, the host should tunnel the Intellimouse recognition sequence to the device and check the response, as outlined in section 5.1.3.2. If the device is an Intellimouse, the host should set the PackSize mode bit on the TouchPad using the special command sequences shown in section 4.3.

5.1.5.3. Absolute mode support for separate enables

In Absolute mode, the driver can effectively disable either device simply by ignoring packets from that device. However, because the packet rate suffers when both devices attempt to send packets at once, the driver may wish to disable the guest's packet stream by tunneling an $F0 (Set Remote Mode) command if the driver was planning to ignore guest packets.

Note: Synaptics does not support the Sleep feature in Absolute mode; therefore, the technique suggested in section 5.1.4.2. does not work in Absolute mode.

5.2. Guest port communication

5.2.1. PS/2 compatibility

The TouchPad expects the guest to appear as a standard PS/2 pointing device. The TouchPad attempts to be forgiving of idiosyncrasies in the guest's PS/2 implementation; particular requirements of the guest are noted below where possible.

The TouchPad does not guarantee to provide full pull-ups on its guest port. The TouchPad provides approximately 100 KΩ of pull-up which the guest should augment with its own pull-ups in the standard 5–20 KΩ range. Future TouchPads may include 10 KΩ pull-ups on the guest port, or they may contain no pull-ups at all on the guest port. Because not all PS/2 devices provide pull-ups, the TouchPad implementation usually includes at least a board stuffing option for 10 KΩ pull-ups on its guest port.

Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

5.2.2. Guest initialization

The TouchPad resets the guest in response to an $FF (Reset) or $F6 (Set Defaults) command from the host, or in response to a PS/2 protocol error on the guest port.

When the TouchPad resets the guest, it performs the following actions:

1. Sends an $F5 (Disable) command code to the guest port and receives the $FA acknowledge response.
2. Sends an $FF (Reset) command code to the guest port and receives the $FA acknowledge response.
3. Waits for $AA $00 from the guest.

When $AA $00 arrives from the guest at any time, whether after guest reset, after power-up of the guest, or at any other time due to hot-plugging or spontaneous reset in the guest, the TouchPad takes the following actions:

4. If the TouchPad is enabled, it sends an $F4 (Enable) command code to the guest port and receives the $FA acknowledge response. If the TouchPad is disabled, it leaves the guest in its default disabled state.
5. Marks the guest "present." In particular, this tells the TouchPad to propagate Enable and Disable commands to the guest in the future.
6. Clears the PackSize bit in the TouchPad's mode byte, causing the TouchPad to expect three-byte packets from the guest.
7. Clears the absolute bit in the TouchPad's mode byte, causing the TouchPad to send three-byte relative packets to the host. In the normal case that the $AA $00 was received from the guest in ultimate response to a propagated $FF (Reset) command from the host, the TouchPad's absolute bit was already clear. If the $AA $00 arrives spontaneously while the TouchPad is in Absolute mode because the Synaptics TouchPad driver is in use, this signals the driver that the TouchPad has experienced a spontaneous reset. The driver responds by resetting and reinitializing the TouchPad (and guest).

Pass-through command tunneling works even if the guest is marked "not present." However, if the guest truly is not present and fails to respond to the tunneled command, then the TouchPad resets the guest as described in section 5.2.3 below.

5.2.3. Guest protocol errors

If a PS/2 protocol error occurs during guest initialization or while sending any propagated or tunneled command to the guest, the TouchPad attempts to reset and reinitialize the guest up to four times in succession. If the TouchPad is repeatedly unable to initialize the guest without error, the TouchPad assumes the guest is not present and does not attempt to enable the guest. The host can send an $FF (Reset) to the TouchPad to force the TouchPad to try again to initialize the guest.

If a PS/2 protocol error occurs while receiving a packet or a response to a tunneled command from the guest, the TouchPad cancels any packet in progress but does not reset the guest.

For these purposes, a PS/2 protocol error includes:

- Failure by the guest to respond to the TouchPad's Request to Send signal on the bus.

 Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

- Failure to observe the 50 µs CLK high and low times during clocking (with generous tolerances).
- Parity error or framing error receiving from the guest.
- Failure to send a response to a command within a reasonable timeout. Currently the timeout is 3 ms, based on known characteristics of dual pointing guests. This document allows a PS/2 device up to 25 ms to respond, so this timeout will have to be increased when Pass-Through is generalized to arbitrary guests.
- Failure to send an $FA response when one is expected. (The TouchPad checks for an $FA response to all command bytes it sends to the guest, but not for tunneled command bytes.)
- Failure to send successive bytes of a command response or data packet within a reasonable timeout. Currently the timeout is 3 ms from one stop bit to the following start bit. This timeout is less than that allowed.

The 3 ms timeout allows the TouchPad to detect and correct loss of synchronization at every packet boundary. Increasing these timeouts to the 20–25 ms range called for would prevent the TouchPad from correcting a loss of synchronization until the end of a motion stroke or other continuous usage of the guest.

5.2.4. Guest packet communication

During normal operation, the guest can send packets to the TouchPad just as it would to any PS/2 host.

The host driver is responsible for setting the PackSize bit on the TouchPad according to the anticipated packet size on the guest. If PackSize is 0, guest packets are assumed to be three bytes long. If PackSize is 1, guest packets are assumed to be four bytes long. The host driver must ensure that PackSize accurately reflects the size of guest packets; if it does not, packet synchronization is lost and the TouchPad's efforts to achieve synchronization will inhibit proper communication with the guest.

The TouchPad may inhibit the guest at any time. In general, the TouchPad inhibits the guest whenever it is communicating with the host. Therefore, the guest must be able to tolerate regular brief inhibitions, especially if the guest is expected to be able to interleave packets with the TouchPad's own packets when both pointing devices are active simultaneously.

After the TouchPad receives a packet from the guest, it holds the guest inhibited until the TouchPad is ready to accept another guest packet. This inhibition lasts until the TouchPad has succeeded in propagating the encapsulated packet to the host.

5.3. Switching Pass-Through Modes

The Pass-Through operation has an alternate *transparent Pass-Though mode*. This mode enables commands to go from host to guest and from guest to host, without requiring TouchPad processing or tunneled command sequences. When in transparent mode (also sometimes referred to as GlassPass):

- The TouchPad is effectively disabled, although it continues internal processing.
- The driver "sees" the guest and only the guest. Every byte is sent quickly down to the guest. Every byte received from the guest is sent immediately to the host.

Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

- PS/2 commands, including those normally used to clear byte 2 of the TouchPad mode byte, are sent on to the guest and do not affect the TouchPad. A special command sequence must be used to return to the default Pass-Through mode.

To put the TouchPad into transparent Pass-Though mode:

1. Disable the TouchPad.
2. Set bit 5, *GlassPass*, of byte 2 of the TouchPad mode byte.

Commands are now sent directly to the guest.

To take the TouchPad out of transparent Pass-Through mode:

1. Send the Set Scaling 2:1 command ($E7).
2. Send the Set Scaling 1:1 command ($E6).
3. Enable the TouchPad.

If for some reason you need to use the Set Scaling 2:1 command ($E7) while in transparent mode, send the command twice. The first command will be absorbed, but the second command will be sent on. The second Set Scaling 2:1 command will not be interpreted as the beginning of a transparent mode escape sequence, so you may follow that command with a Set Scaling 1:1 command ($E6) and the device will stay in transparent mode.

The PS/2 Reset command ($FF) can be used in transparent mode, where it resets only the guest, not the TouchPad. To use the Reset command in transparent mode, you must precede the Reset command with the Set Scaling 2:1 command ($E7). A Set Defaults command, which has almost the same effect as a full Reset command, can be sent directly to the guest, and it does not need to be preceded by the Set Scaling 2:1 command. Other PS/2 commands, even those normally used to clear byte 2 of the TouchPad mode byte, are also sent on to the guest but do not affect the TouchPad.

A spontaneous reset of the TouchPad, due to internal sanity checks, ESD, or some other problem, will take the device out of transparent Pass-Through mode.

 Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

6. PS/2 Implementations

The next two subsections describe two implementations of the host side of the PS/2 interface. On standard PC-compatible computers, the *keyboard controller* chip is responsible for PS/2. On other types of systems, it may be necessary to implement the PS/2 host interface in system software. Section 6.2 shows the source code for such an implementation.

6.1. Keyboard controller

On a standard PC, the keyboard controller (KBC) chip implements the host side of the PS/2 interface. Host software can operate the TouchPad without any regard to (or awareness of) the PS/2 protocol at the level of the CLK and DATA signals. Section 2.3.3 discusses the role of the KBC in general terms.

A good reference book for the KBC is chapter 8 of Frank van Gilluwe's *The Undocumented PC.* As described in that book, the interface to the KBC takes the form of two I/O ports and an interrupt vector. The host can read I/O port 64h to check the status of the KBC's input and output buffers. The host can write to port 64h to send a command to the KBC. The KBC commands include A7h, which inhibits the pointing device by holding CLK low; A8h, which de-inhibits the pointing device; and D4h, which causes the next data byte written to I/O port 60h to be sent to the mouse as a command or argument byte.

When the pointing device sends a byte, the KBC raises the IRQ 12 interrupt to notify the host. The byte is then available to be read from input port 60h. Various bits of input port 64h tell whether a byte is available in port 60h, and whether this byte came from the keyboard or pointing device.

In principle, the KBC is a simple conduit for bytes going between the device and the higher-level host software. In practice, some KBCs differ from this ideal in various ways. Writers of PC software that interfaces directly with the KBC should be aware of these known quirks:

- As described in section 2.3.2, the KBC expects a response byte for every byte sent to the device. If the KBC times out waiting for the device to acknowledge the Request to Send condition, or if it times out waiting for the ACK byte to arrive, then the KBC will invent an $FE response and report it to the host as if the device had actually sent an $FE.

- The KBC may test for the presence of a pointing device at boot time and shut down the PS/2 port if no device is found. So, if a PS/2 device is hot-plugged onto a computer which was booted with no PS/2 device present, the computer may need to be rebooted for the device to be recognized.

- Some KBCs will only send the PS/2 commands listed in section 4.1 to the pointing device. For "invalid" commands, the KBC will reject the command locally rather than sending it to the mouse to be rejected. This is one reason why the TouchPad special commands (section 4.2) are disguised as sequences of valid section 4.1 commands.

- Some KBCs attempt to merge two or more active PS/2 devices on the same port. Thus, when a host sends a command to "the device," the KBC forwards the command to both devices. When either device sends a motion packet to the host, the KBC forwards it to the host as a new packet from "the device." If both devices try to talk simultaneously, the KBC must buffer up the data bytes so that the host will see a sequence of coherent three-byte packets with no interleaving. Hence, under some conditions, a byte sent by a device may not reach the host immediately.

Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

- If the KBC merges two PS/2 devices, and the user holds down the mouse button on device #1 and then moves both devices at once, the host would receive alternating packets with the button pressed (from device #1) and released (from device #2), which would lead to undesirable false clicking. So, the KBC sometimes edits the mouse button bits. For example, device #2's packets would be edited to report a button pressed even though device #2 thinks no buttons are pressed. To accomplish this feature, the KBC must keep a count of all the bytes it transfers to the host so that it knows which bytes hold button data. That is why the TouchPad's Absolute packet format (Figure 3-3) is exactly six bytes long with duplicate button data every three bytes.

- Some KBCs will edit bit 3 of the first byte of each three-byte packet. Hence, neither the Relative nor the Absolute packet formats store useful information in this bit. Even though this bit is documented as always 1 or always 0, the bit may be different as seen by high-level host software.

6.2. Sample PS/2 implementation

The Synaptics TouchPad can also be used in appliances, handheld devices and other special applications where the KBC and other facilities of the standard PC are not present. In such applications, you may need to implement the PS/2 host-side interface yourself. This section presents a sample C-language implementation of the PS/2 host interface.

These routines depend on the following functions, which you must define suitably for your environment.

```
typedef unsigned char byte;     /* All data values are 8-bit bytes. */
extern byte read_CLOCK();       /* Return state of clock pin, 0 or 1. */
extern byte read_DATA();        /* Return state of data pin, 0 or 1. */
extern void set_CLOCK(byte);    /* Pull clock pin low (0) or let it float high (1). */
extern void set_DATA(byte);     /* Pull data pin low (0) or let it float high (1). */
extern void wait_us(byte);      /* Wait a number of microseconds (approx). */
extern void PS2_error();        /* This gets called in case of error. */
```

The `PS2_error` function is a stand-in for error handling suitable to the application; in some cases, it may suffice to ignore error checking altogether.

The following function waits for a high or low level on the PS/2 clock pin. In a real application, this function should time out and call `PS2_error` if the desired clock level does not appear after some amount of time (such as 25 ms).

```
1  void wait_CLOCK(byte state)
2  {
3    while (read_CLOCK() != state) /* Do nothing */ ;
4  }
```

To read a byte from the PS/2 device, use the following function:

```
1  byte PS2_get()
2  {
3    byte i, bit, value = 0, p = 0;
4    set_CLOCK(1);               /* Release inhibit, if necessary. */
5    wait_CLOCK(0);              /* Wait for start bit clock. */
6    wait_CLOCK(1);              /* End of start bit clock. */
```

 Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

```
7     for (i = 0; i < 8; i++) {
8       wait_CLOCK(0);              /* Wait for clock pulse. */
9       bit = read_DATA();          /* Read data bit from pin. */
10      value = value + (bit << i);
11      p = p + bit;                /* Accumulate data bit into parity. */
12      wait_CLOCK(1);              /* Wait for end of clock pulse. */
13    }
14    wait_CLOCK(0);                /* Parity bit clock. */
15    p = p + read_DATA();          /* Accumulate parity bit into parity. */
16    if ((p & 0x01) == 0)          /* Check for odd parity. */
17      PS2_error();                /* Parity error! */
18    wait_CLOCK(1);                /* End of parity bit clock. */
19    wait_CLOCK(0);                /* Stop bit clock. */
20    if (read_DATA() == 0)         /* Check for valid stop bit. */
21      PS2_error();                /* Framing error! */
22    set_CLOCK(0);                 /* Pull low during stop bit to inhibit. */
23    wait_us(50);                  /* Wait out the final clock pulse. */
24    return value;
25  }
```

The `PS2_get` function reads one byte from the TouchPad. If the TouchPad is not ready to transmit, the `wait_CLOCK` call at line 5 will wait until it is. If you wish to do other things while waiting, release the inhibit as shown in line 4, then check the clock pin periodically (at least every 20 µs or so) or using interrupts, and call `PS2_get` as soon as the clock goes low.

Line 22 inhibits the bus as soon as the byte is received; this is the recommended way to use the PS/2 bus. Note the 50 µs wait in line 23, to make sure the device has finished the last clock pulse before returning. Otherwise, an immediately following call to `PS2_get` might mistake the tail end of this stop bit as the beginning of a new start bit.

Lines 11 and 15–17 check the parity of the received byte. Lines 20–21 check for framing errors. You can omit these lines if you don't want to check for transmission errors.

To send a byte to the PS/2 device, use the following function:

```
1   void PS2_send(byte value)
2   {
3     byte i, ack, p = 1;
4     set_CLOCK(0);               /* Begin inhibit, if necessary. */
5     wait_us(100);               /* Inhibit for about 100us. */
6     set_DATA(0);                /* Hold data pin low while still inhibited. */
7     set_CLOCK(1);               /* Establish request-to-send state. */
8     for (i = 0; i < 8; i++) {
9       wait_CLOCK(0);            /* Wait for clock pulse. */
10      set_DATA(value & 0x01);   /* Output i'th data bit. */
11      p = p + value;            /* Accumulate parity. */
12      wait_CLOCK(1);            /* Wait for end of clock pulse. */
13      value = value >> 1;       /* Shift right to get next bit. */
14    }
15    wait_CLOCK(0);              /* Parity bit clock. */
16    set_DATA(p & 0x01);         /* Output parity bit. */
17    wait_CLOCK(1);              /* End of parity bit clock. */
18    wait_CLOCK(0);              /* Stop bit clock. */
19    set_DATA(1);                /* Stop bit must be 1. */
20    wait_CLOCK(1);              /* End of stop bit clock. */
```

Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

```
21    wait_CLOCK(0);              /* Line control bit clock. */
22    if (read_DATA() == 1)
23      PS2_error();              /* Missing line control bit! */
24    wait_CLOCK(1);              /* End of line control bit clock. */
25    ack = PS2_get();            /* Receive acknowledge byte from device. */
26    if (ack != 0xFA)
27      PS2_error();              /* Probably got an FE or FC error code. */
28  }
```

The `PS2_send` function first inhibits the bus for at least 100 µs. This ensures that any transmission the device may have begun will be cancelled before it collides with the host's transmission. (If you leave the bus inhibited at all times between `PS2` calls, it is safe to skip lines 4 and 5.)

Next, the function asserts a request-to-send and releases the inhibit signal. The device will respond within 10 ms by clocking in a start bit, data bits, parity bit, and stop bit. Lines 3, 11 and 16 are responsible for sending a proper parity bit; while some older Synaptics TouchPads ignored parity, the current TouchPad does check for parity errors so it is essential for the host to transmit a correct parity bit.

Line 25 calls `PS2_get` to receive the $FA acknowledge byte. It is okay to move steps 25–27 out of the `PS2_send` function, but in this case make `PS2_send` inhibit the bus before returning. This forces the device to wait until you are ready to receive the acknowledge byte.

If there are asynchronous interrupts on your system that take more than a few microseconds to service, you should disable them inside the `PS2_get` and `PS2_send` routines. Once a transmission has begun, timing is critical and not under the host's control.

For the TouchPad special command sequences described in section 4.2, the following helper function is useful. It sends an 8-bit argument encoded as a sequence of four Set Resolution commands.

```
1  void send_tp_arg(byte arg)
2  {
3    byte i;
4    for (i = 0; i < 4; i++)
5      PS2_send(0xE8);
6      PS2_send((arg >> (6-2*i)) & 3);
7  }
```

Lines 5 and 6 send Set Resolution commands with arguments consisting of bits 7–6, 5–4, 3–2, and 1–0 of `arg`, respectively, as `i` counts from 0 to 3.

As described in section 2.4, the device will send an $AA, $00 announcement 300–1000 ms after power-up. However, it is safe to send your first PS/2 command before this time if you don't care about the $AA, $00. (The proper clocking and acknowledgement of the first command serves just as well as the $AA, $00 to verify that the TouchPad is present and running.) It is still a good idea to wait until 300 ms after power-up before talking to the device. Note that the host should be prepared to wait up to 1000 ms after power-up for the device to respond to the first request-to-send, although the current Synaptics TouchPad responds much sooner. Also note that the official PS/2 specification states that the host must not send to the device until after the $AA, $00 announcement, so the host must wait if it expects to be used with devices other than Synaptics TouchPads.

Here is a typical sequence to initialize the TouchPad and enable Stream mode using the Absolute data format of Figure 3-3.

 Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

```
1   PS2_send(0xFF);                 /* Reset command. */
2   if (PS2_get() != 0xAA)          /* Note:  This may need an extra-long timeout. */
3     PS2_error();
4   if (PS2_get() != 0x00)
5     PS2_error();
6   send_tp_arg(0x00);              /* Send "Identify TouchPad" sequence (section 4.4)*/
7   PS2_send(0xE9);                 /* Status Request command.  */
8   minor = PS2_get();              /* First status byte: TouchPad minor rev. */
9   if (PS2_get() != 0x47)          /* Second status byte: 0x47 == Synaptics TouchPad. */
10    PS2_error();
11  major = PS2_get() & 0x0F;       /* Third status byte: Major rev in low 4 bits. */
12  send_tp_arg(0x80);              /* Send "Set Modes" sequence (see section 4.2). */
13  PS2_send(0xF3);                 /* Set Sample Rate command. */
14  PS2_send(0x14);                 /* Sample Rate argument of 20. */
15  PS2_send(0xF4);                 /* Enable command. */
16  enable_interrupt_handler();     /* Ready to receive data. */
17  set_CLOCK(1);                   /* Release PS/2 bus inhibit. */
```

Lines 1–5 perform a Reset command. If this initialization sequence is used only right after power-up, lines 1–5 are not strictly necessary since Reset merely restores the power-up defaults.

Lines 6–11 perform the "Identify TouchPad" query using a special command sequence. This query tells you that you do indeed have a Synaptics TouchPad attached and not some other pointing device.

Lines 12–14 enable Absolute mode by setting bit 7 of mode byte 2. A more elaborate initialization sequence would enable W mode (bit 0) as well if the TouchPad identifies itself as supporting that mode. You can also enable the high packet rate (bit 6), though the higher rate will cause the host to spend a significant fraction of its time in the `PS2_get` routine whenever the finger is on the sensor.

Line 15 sends an Enable command, which enables the transmission of finger motion packets (actually position/pressure packets in Absolute mode). The device won't actually transmit a packet yet, since `PS2_send` and `PS2_get` leave the bus inhibited.

Lines 16 and 17 assume the host will operate the TouchPad in an interrupt-driven Stream mode. Line 16 stands for whatever steps are necessary to enable interrupts when the PS/2 clock pin goes low. Line 17 ends the inhibit signal. (If the interrupts are level-sensitive, these steps will have to be done in the opposite order.)

The interrupt handler installed by line 16 should respond to a low clock signal by calling `PS2_get` to receive the byte. Note that the start bit is only 30–50 µs long, so the interrupt latency must be small. If you need to disable interrupts at certain times, be sure the PS/2 bus is inhibited (by holding the clock wire low) during these times.

Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

Appendix A. Driver API

Most IBM PC-compatible computers use the PS/2 protocol for low-level communication with the TouchPad. On computers running the Microsoft Windows operating system, this low-level communication is normally managed by "mouse" driver software. Application software talks to the TouchPad hardware only indirectly via the driver.

As shown in Figure 2-8 of section 2.3.3, the Synaptics Windows TouchPad drivers operate the TouchPad in Absolute X/Y/Z/W mode. But because the Synaptics drivers are still "mouse" drivers from Windows' point of view, the Synaptics drivers normally process X, Y, Z, and W into ΔX, ΔY, and virtual buttons, effectively acting as if the TouchPad were in Relative mode.

Working only with the facilities provided by Windows, applications see the TouchPad as a regular Relative-mode mouse. The Synaptics drivers provide a special API (Application Programming Interface) that applications can use to get the original X, Y, Z, and other TouchPad-specific information. Applications can use the TouchPad API to take full advantage of the special abilities of the TouchPad. For example:

- A drawing application could use the TouchPad as a miniature, pressure-sensitive graphics tablet.
- A game could use the TouchPad as a customized game controller by decoding special zones or gestures.
- Several of the features and accessories that come with the Synaptics drivers are really just API applications. These include virtual scroll bars, "stop pointer at window borders," the animated tray icon, Pressure Graph, MoodPad, and Sketch.

For complete documentation, download the TouchPad SDK from the Synaptics Web site, http://www.synaptics.com.

 Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

Appendix B. Glossary and index

This section summarizes the definitions of many of the terms and notations used in the *Synaptics TouchPad Interfacing Guide*. The "§" symbol denotes a reference to the section of this document where a word or concept is discussed. If a word in the description is in italics, is defined elsewhere in the glossary.

$ In this document, the dollar sign signifies hexadecimal (base-16) notation: $7FF = 0x7FF = 7FFh = 2047 decimal.

— In bit-field diagrams, see *reserved.*

Absolute Mode In the PS/2 protocol, a mode of operation where the TouchPad transmits an extended *packet* which reports the absolute finger position on the *sensor* (X, Y), the finger pressure or contact area (Z), and various other information such as the state of the buttons. (§3.2)

Absolute Position The position of the finger on the *sensor* surface measured absolutely with respect to a coordinate system with the point (0,0) in the lower-left corner. See also *relative motion.* (§3.2.3, Figure 3-6)

Absolute Reportable Limits The most extreme coordinate values that the TouchPad can report under any circumstances. The physical nature of the *sensor* ensures that all actual coordinate values will fall in a much narrower range. (§3.2.3)

Acceleration Pointing devices typically offer a feature called "acceleration" or "ballistics" which increases the speed factor at higher speeds to help the user move the cursor a long distance with a small motion. (§1.1.4)

ACK A response byte with value $FA used to acknowledge each host command or argument byte in the PS/2 protocol. (§2.3.2)

ACPI The Advanced Configuration and Power Management Interface, a standard promoted by Intel, Microsoft, and Toshiba.

Announcement An unsolicited transmission from a device which tells the host that the device is present and powered on. In *PS/2* pointing devices, the announcement is $AA, $00. (§2.4)

API Application Programming Interface. Typically, this refers to a set of functions and data types offered by a piece of system software to allow access by application software. (§Appendix A)

Application Software Software that interacts directly with the user, usually that was deliberately run by the user to accomplish some task. Application software generally interacts with the TouchPad via the TouchPad *API.* (§Appendix A)

ASIC Application Specific Integrated Circuit. A chip specially designed for a particular task. TouchPads include a capacitive-sensing ASIC designed by Synaptics.

Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

Bezel A physical covering that surrounds some TouchPad sensors. The bezel keeps the finger from straying outside the active sensor area, and also keeps the TouchPad electronics safe from dirt and *ESD*. Synaptics publishes recommended bezel shape and position for TouchPad models that feature bezels. If the bezel opening is too large, it may expose area beyond the active sensor which will result in an undesirable "dead spot." If the bezel opening is too small, it may obstruct the *edge zones,* which will prevent the EdgeMotion feature from operating correctly.

Bezel Limits Coordinate values that would be reported by a finger held against the edge or corner of the bezel. Actual reachable limits depend on the bezel opening and the size of the finger, so the bezel limits shown in Figure 3-5 are "padded" to ensure that most fingers will be able to reach the bezel limits. The TouchPad does not clip its coordinates to the bezel limits, so the coordinates may sometimes vary outside this range especially when the *sensor* is used with small fingers. (§3.2.3)

BIOS Basic Input/Output System, responsible for starting up the computer system and acting as an intermediary between the CPU and the input/output devices. (§2.3.3, §Figure 2-7)

Calibration A process in which the TouchPad prepares its capacitive sensors for operation.

Capability A feature which is supported if an associated "capability bit" is reported as 1 by the TouchPad. In this *Guide,* capability bits have names beginning with "*cap*...". (§3.2.7)

Capacitance The electrical phenomenon that Synaptics TouchPads use to sense the presence of fingers. (§1.1.1)

CLK Also "clock." The PS/2 signal wire that carries timing information from the *device* to the *host,* as well as inhibit commands from the host to the device. (§2.1)

Command One or more bytes sent by the *host* to the *device* as a request for the device to perform some action or answer some *query.*

Connector A plug for connecting a cable to a machine or to another cable. Male connectors have pins which fit the corresponding holes in the female connector. The host has a male connector and the pointing device is female. (§2.1)

Coordinates A pair of numbers identifying an *absolute position* on the surface of the TouchPad. See *X coordinate* and *Y coordinate.* (§3.2.3)

DATA The PS/2 signal wire that carries data bits between the *device* and the *host*. (§2.1)

Deltas A pair of numbers measuring an amount of *relative motion*. Deltas are named for the Greek letter Δ which is used in mathematical notation to signify an amount of change in a variable. The "ΔX" value measures change in the *X coordinate*, i.e., horizontal motion. The "ΔY" value measures vertical motion. (§1.1.4)

Device In the discussion of TouchPad *protocols,* "device" refers to the TouchPad or other pointing device, as distinct from the *host*.

 Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

Device Side The "downstream" side of a *PS/2* port, usually a mouse.

DIN-6 The small, round six-pin connector used for *PS/2* mouse ports and some keyboard ports. Actually, this connector is correctly known as a "mini-DIN-6." The larger DIN-5 connector is sometimes used for keyboard ports, especially on older machines. (§2.1, Figure 2-1)

Down Orientation The "down" orientation for TouchPads is defined as the orientation in which the cable exits from behind the bottom edge of the module, i.e., the edge closest to the user's body. (§Figure 4-10 (*b*))

DPI Dots Per Inch. A measure of *resolution;* when applied to pointing devices, a "dot" is generally taken to mean one unit of position or one unit of motion. An absolute resolution of 2000 DPI means that a change in finger position by one inch will cause a change in the X or Y coordinate by 2000 units. A relative resolution of 200 DPI means that a change in finger position by one inch will produce a sequence of packets whose ΔX or ΔY values add up to 200 units. (§3.2.4)

Driver A piece of system software responsible for operating a hardware device on behalf of higher-level software. The TouchPad is generally operated either by a standard *mouse* driver supplied by the operating system, or by the Synaptics TouchPad Driver. (§2.3.3)

Dual Mode The Synaptics Dual Mode TouchPad is both a cursor control device and an application launch and control center. When the user taps a mode switch button, the TouchPad illuminates previously hidden icons on the TouchPad surface and transforms into a secondary control, allowing users to quickly launch programs, control media functions, or provide any other set of secondary input. To return to cursor control mode, the user taps the mode switch button again.

Dual Pointing A notebook computer that has a TouchPad and a pointing stick.

Edge Margins The coordinate limits that identify the dividing lines between the *edge zone* and the interior of the sensor. (§3.2.3)

Edge Zone The area comprising the parts of the TouchPad surface very near to the edge of the *bezel*. (Equivalently, the "interior" is a rectangular area covering the central part of the sensor, and the edge zone is the part of the sensor surface not in the interior.) Moving the finger into the edge zone triggers EdgeMotion, if the EdgeMotion feature is enabled in the Synaptics device driver software. (§3.2.3, §1.1.5)

Electrical Noise See *noise.*

Encapsulation Packet An Absolute-mode packet that passes guest data on to the host instead of reporting data from the master itself.

Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

ESD Electrostatic Discharge. When the user builds up a charge of static electricity and then touches a conductive object, a small spark can result. This spark can compromise sensitive electronic devices such as TouchPads. To minimize the risk of ESD disruption, designers should carefully follow Synaptics' recommendations on grounding.

Facesheet The layer over the sensor that acts as an insulator as well as cosmetic element. A facesheet can be made of plastic, film (such as Mylar®), or other non-conductive material. Sensors mounted under a product's plastic casing (TouchPad Under Plastic) do not require a facesheet.

Filtering A general term for data processing steps that try to reduce the effects of *noise* and produce smoother motion. The TouchPad includes various filtering algorithms built-in. When using the Absolute mode X and Y *coordinates* for precise work, software designers may wish to apply more filtering. (§3.2.3)

Finger Synaptics TouchPad technology assumes a conductive finger that appears as a ground. (§1.1.1, §3.2.6)

Firmware The software that operates the microcontroller built-in to the TouchPad module. See also *version number.* (§4.4)

Framing Error The PS/2 protocol sends bytes in several-bit "frames" consisting of a start bit, data bits, and *stop bit*. If the receiver detects the wrong value for the stop bit, it can deduce that transmission errors have caused the transmitter and receiver to lose agreement on which bit interval is the beginning of a new frame. (§2.3.1)

Guest The PS/2 device that connects to the system indirectly through the master.

Guest Port The PS/2 port between the master and the guest. The master is on the host side of this port and the guest is on the device side.

Host The computer or other system in which the TouchPad is a part. To a pointing device, the "host" is both an immediately connected piece of hardware (such as the *KBC* in the case of the *PS/2* protocol), and the larger computer system comprising drivers, the operating system, and application software. The host connects directly to the master and indirectly to the guest. (§2.3.3)

Host Port The PS/2 port between the master and the host. The host is on the host side of this port and the master is on the device side.

Host Side The "upstream" side of a PS/2 port, usually the computer.

Host Software Any software running on the host that interacts with the TouchPad; for example, *drivers, application software,* and the operating system. (§2.3.3)

 Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

Hot-Plugging The act of attaching a device to a computer which is already powered up and running. Hot-plugging a PS/2 device does not work without special attention from the driver, since the device needs an Enable command to be transmitting packets. Hot-plugging is *not* recommended for PS/2 devices because of the design of the connectors. Hot-plugging can cause signal wires to make contact before power supply wires, which can result in damage to the TouchPad's electronics.

Inhibit In the PS/2 protocol, the *host* can inhibit the *device* to prevent the device from sending new data until the host is ready. The host can also inhibit during a transmission to cancel the transmission in progress. (§2.3)

KBC Keyboard Controller. The part of the *host*, usually a subsidiary microprocessor, which manages the host side of the *PS/2* interface. (§2.3.3, §6.1)

IRQ Interrupt ReQuest. A hardware interrupt on a PC. The *KBC* communicates to the TouchPad driver, the mouse driver, or the *BIOS* via I/O ports and the IRQ line. (§2.3.3)

Lifting Raising the finger far enough away from the surface of the TouchPad that the sensor no longer registers the finger's presence (i.e., until the *Z value* falls below the *touch threshold*). (§1.1.1)

Line Control Bit The final bit of a PS/2 transmission from *host* to *device*. (§2.3.1)

Margin See *edge margins.*

Master The PS/2 device that communicates directly to the host, and that also connects to the guest.

Mickey One unit of mouse motion as reported by the pointing device to the host. (§1.1.4)

Mode Byte An 8-bit value held by the TouchPad which contains various bits that control the behavior of the TouchPad. Each *protocol* provides a way for the host to read and change the mode byte. (§4.3)

Model ID A 24-bit response to a certain *query* which describes the size and shape of the TouchPad. (§4.4)

Module A typical TouchPad product from Synaptics is a "module" consisting of a circuit board with components and connector on the back and a protective mylar label on the front surface. (§2.1)

Mouse A pointing device containing motion sensors which can move freely on a work surface. Because a mouse uses motion sensors, it can only deliver *relative motion* data to the host. Most system software assumes the pointing device is a mouse by default. Most pointing devices simulate a mouse by default.

Mouse Button See *virtual button* and *physical button*.

Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

MultiSwitch A type of Synaptics TouchPad which supports extra buttons. The four-button MultiSwitch TouchPad supports four *physical buttons* instead of just the usual two. The two additional buttons are specified "Up" and "Down." Other MultiSwitch TouchPads can support up to 11 buttons. (§ 3.2.1, §3.2.8, §4.4)

Noise Electrical noise is interference caused by fluctuations in the ground, the power supply, or the electric field surrounding the TouchPad. Synaptics TouchPads use a variety of methods to minimize the effects of noise, but extreme noise can cause jitter in the X, Y, and Z values. Reducing the packet rate is one way to combat noise (§2.3); filtering is another. (§3.2.3)

Open-Drain Also "open-collector." A type of digital signal where the transmitting circuit is able to drive the wire to 0V or to let the wire "float" to any voltage, but *not* to drive the wire to a high voltage. A pull-up resistor is attached to the wire to cause the wire to float to a high voltage when no other circuits are driving it low. (§2.1)

Packet One transmission from the pointing device to the host describing the user's actions. The device sends many packets per second in to form the illusion of continuous cursor motion. (§3.1, §3.2)

Packet Rate The rate at which the TouchPad potentially sends packets to the host. In PS/2 Remote mode, the packet rate is actually the rate at which new packets become available for polling. In Relative mode, the packet rate is merely the maximum possible number of packets transmitted per second. In Absolute mode, the packet rate is the guaranteed number of packets per second whenever transmission is in progress. Not to be confused with *PS/2 sample rate.* (§2.3)

Palm Detection A TouchPad which measures the width of finger contact to distinguish between true finger contact and accidental activation by the palm of the hand. (§3.2.6)

Parity A method for detecting transmission errors. The transmitter counts the number of 1 bits sent, then sends a parity bit which is either 1 or 0 in order to cause the total number of 1s to be odd (thus the name "odd parity"). The receiver then counts the number of 1 bits and detects an error if the number is not odd. Parity guarantees detection of any single-bit error, and some multi-bit errors. (§2.3)

Pass-Through (or "pass-thru"). A system in which two or more PS/2 devices are chained on a computer's PS/2 mouse port. Also, a type of PS/2 device suitable for use as a master in a Pass-Through system. (§5)

Physical Button A button or switch on a pointing device which sends a command signal to the host. On an IBM-compatible PC, the pointing device typically has two buttons labeled "left" and "right." The phrase "physical button" refers to an actual button, not a *virtual button* activated by *tap gestures*. (§3.2.2)

Pointing Device A mouse, pointing stick, TouchPad, or other device used to move the cursor on the computer screen.

Portrait A portrait-oriented TouchPad is rotated 90° from the typical orientation, so that the vertical axis is longer than the horizontal. The typical orientation, where the

 Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

	horizontal axis is longer, is sometimes called landscape orientation. (§ Figure 4-11 (*b*))
Pressure	As measured by a TouchPad, "pressure" is actually the total amount of capacitance over the contact area of the finger. Pressing harder causes the flesh of the finger to flatten out against the pad, thus increasing the contact area. Hence, in casual use "pressure" and "contact area" can be treated as the same. (§3.2.5, §1.1.1)
Propagation	A command is "propagated" if the master passes it on to the guest as well as processing it locally on the master. Propagated commands include enable, disable, and reset.
Protocol	A defined method for communicating between a *device* and the *host.* See *PS/2.*
PS/2	The protocol used by most internal pointing devices in notebook computers, and by many external pointing devices. First used for pointing devices in the IBM PS/2 computer, although the IBM PC AT used essentially the same low-level protocol for the keyboard. (§1)
PS/2 Port	A connection consisting of PS/2 CLK and DATA wires with suitable hardware and/or firmware on the host and device sides.
PS/2 Resolution	A parameter of the PS/2 mouse protocol; not fully implemented on Synaptics TouchPads. See *resolution.* (§4.1)
PS/2 Sample Rate	A parameter of the PS/2 mouse protocol; not fully implemented on Synaptics TouchPads. See *packet rate.* (§4.1)
Query	A command from the host which is a request for information about the properties or current state of the TouchPad. (§4.4)
Relative Mode	A mode in which the TouchPad reports the *relative motion* of the finger in each *packet*. In Relative mode, the TouchPad device behaves like a mouse, and emits 3-byte packets which encode the amount of motion in the horizontal and vertical direction made by a finger touching the sensor. This is the default mode for TouchPads using the PS/2 protocol. (§3.1)
Relative Motion	The change in position of the finger relative to the finger's previous position. A conventional mouse can report only relative motion, not *absolute position*, because the mouse has no way to sense an absolute point of reference (such as the lower-left corner of the desk or mouse pad). (§1.1.4)
Request To Send	Or RTS. In the PS/2 protocol, a condition on the bus that tells the *device* to read a command from the host as soon as possible. (§2.3)
Reserved	A bit or bit-field is "reserved" if the present TouchPad models do not use it in any (published) way. The host should not interpret a reserved bit. If the host writes to a reserved bit, it should write the default value (or zero if not shown), or it can read the reserved bit and then write back the same value.

Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

Reset An action or command to restore a *device* to its initial, default state. The TouchPad resets itself when power is first applied. Also, each protocol provides a form of software reset. In PS/2, the Reset ($FF) command. (§2.4)

Resolution The number of positioning units corresponding to a given physical distance on the pad. See also *DPI.* Not to be confused with *PS/2 resolution.* (§3.2.4)

Response One or more bytes sent by the *device* to the *host* in response to a host *command*.

Sensor The top surface of the TouchPad module, which senses the presence and position of a finger.

Serial Number A unique number assigned to each individual TouchPad. A TouchPad with an assigned serial number is said to be serialized. (§4.4)

Sleep A mode in which the device operates at reduced power in exchange for reduced functionality (namely, the ability to sense buttons but not fingers). (§4.3)

Stop Bit One or more bits of a known value at the end of each transmitted byte. The PS/2 protocol uses the stop bit to detect *framing errors*. (§2.3.1)

Stroke A finger stroke is one complete finger action involving placing the finger on the sensor, possibly moving the finger around for some amount of time, then lifting the finger away from the sensor. (§1.1.4)

Switch See *physical button.*

Tap Zone An area on a sensor designated for customizable tap gestures. Tap zones can be used to simulate mouse clicks, to launch applications, or to perform other programmable functions. For example, on a TouchPad, the four corners can be configured to perform actions such as launching an application. Users can also configure the size of each Tap Zone.

TBD To Be Determined. Contact Synaptics for further information.

Transparent Pass-Though Mode
A mode of operation where commands are sent from the host to the guest, bypassing the TouchPad. The TouchPad remains disabled in this mode, although it continues internal processing. This mode speeds up communication between a guest, such as a TouchStyk or other pointing device, and the host. (§5.3)

TouchPad A pointing device which translates motions of a finger on a sensor surface into cursor motions. TouchPads support complex pattern recognition such as multi-finger gestures and character recognition in addition to simple gestures. Custom gesture support is even possible.

TouchPad Under Plastic
A TouchPad that is mounted on the underside of a non-conductive plastic palm rest. The TouchPad Under Plastic is environmentally sealed around the TouchPad area to minimize contamination and moisture problems, and increase resistance to electro-static discharge**.**

 Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

Touching	The Synaptics TouchPad considers the finger to be "touching" if it is close enough to register a Z value of at least 25–30. For typical fingers, this represents light finger contact. (§1.1.1)
Touch Threshold	A value to which the *Z value* is compared to decide whether or not there is a finger on the sensor. (§1.1.4)
Tunneling	The process of sending a command from the host to the guest without interpretation by the master. Together with encapsulation packets, this mechanism allows the host to operate the guest directly, albeit with greatly reduced efficiency.
Undefined	Same as *reserved.*
Up Orientation	The usual "up" orientation for TouchPads is defined as the orientation in which the cable exits from behind the top edge of the module, i.e., the edge farthest from the user's body. (§ Figure 4-10 (*a*))
Virtual Button	A simulated mouse button activated by *tap gestures* instead of by a physical switch. The system typically treats the physical and virtual buttons the same so that tap/slide gestures can serve as a convenient alternative to button clicks. (§1.1.5)
Version Number	A two-part number identifying the revision level of the TouchPad, especially of its physical design and its *firmware.* In the version number "6.5", "6" is the *major version* and "5" is the *minor version.* (§4.4)
Wide Contact Check	A feature of Synaptics drivers that guards against unintentional cursor movement or taps caused by accidental contact of the palm or hand with the sensor. Wide Contact Check allows the sensor to recognize when a palm is resting on it or brushing its surface while typing.
Windows	Any of the Microsoft Windows family of operating systems.
W Mode	A variant of *Absolute mode* in which the TouchPad reports the *W value* in each packet as well as X, Y, and Z.
W Value	A four-bit code which identifies the character of finger presence, e.g., the width of the finger and the number of fingers. (§3.2.6)
X Coordinate	The *coordinate* identifying the horizontal position of the finger; low values are near the left edge of the sensor, and high values are near the right edge. (§3.2.3)
ΔX Value	The *delta* identifying the amount of horizontal finger motion.
Y Coordinate	The *coordinate* identifying the vertical position of the finger; low values are near the bottom edge of the sensor (closest to the user), and high values are near the top edge. (§3.2.3)
ΔY Value	The *delta* identifying the amount of vertical finger motion.

Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

Z Value A numeric value reported by the TouchPad which measures the *pressure* of the finger contact. (§3.2.5)

 Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.

Appendix C. Synaptics literature and websites

- *Synaptics Gestures for TouchPad Solutions Product Brief (PN: 190-000022-01)*
- *Synaptics TouchPad Product Profile (PN: 509-000005-01)*
- TouchPad Information
  `http://www.synaptics.com/solutions/products/touchpad`
- Synaptics API SDK V1.0
  `http://www.synaptics.com/resources/developers`
- Synaptics Drivers
  `http://www.synaptics.com/support/drivers`

Contact Us

To locate the Synaptics office nearest you, visit our website at www.synaptics.com.

Copyright © 2001—2011 Synaptics Incorporated. All Rights Reserved.