Synaptics TouchPad Interfacing Guide

510-000080 - A

Second Edition

Created:	2.0	March 25, 1998
Revision:	2.5	January 18, 2000
Printed:		January 22, 2001

Synaptics, Inc.
2381 Bering Drive
San Jose, California 95131
(408) 434-0110
Fax (408) 434-9819

1. Overview

This guide describes how computers and other hosts interface to the Synaptics TouchPad. The first section describes the TouchPad generally, including operating modes, features, host interactions, and principles of operation, with many historical notes comparing older versions of the Synaptics TouchPad with the present one, version 4.5. (See page 4.)

The Synaptics TouchPad family supports a variety of protocols for communicating with the host computer. The next few sections describe the available protocols:

- The **PS/2** protocol is the method that most portable computers use to interface with keyboards and pointing devices. (See page 27.)
- The **Serial** protocol connects the pointing device to the host using a standard RS-232 serial port. (See page 50.)
- The **ADB** protocol is used by Apple Macintosh family computers. (See page 64.)

In each case, the TouchPad supports the industry standard “mouse” protocol plus a number of TouchPad-specific extensions. This *Guide* describes the PS/2 and Serial protocols in complete detail, and the ADB protocol in all details not covered by Apple publications. System architects and developers can read these sections of the *Guide* to learn how to interface to the TouchPad hardware. (For detailed mechanical and electrical data, refer to the various *Model TM41xx Product Specification* sheets also available from Synaptics.)

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. (See page 73.)

The **Glossary/Index** (page 79) defines all the technical terms that appear in this *Guide.*

1.1. Table of Contents

2. 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 which 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 trackball, but its compact size, low cost, and lack of moving parts makes it ideal for portable computers. The Synaptics TouchPad's advanced features 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. However, there are also many things that all Synaptics TouchPads have in common: They support the same features and modes; they offer roughly the same set of commands and queries to the host; they operate according to the same principles. This first part of the *Interfacing Guide* describes the common aspects of the Synaptics TouchPad.

2.1. Mouse-compatible Relative mode

When power is applied, the Synaptics 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 pad. The TouchPad reports this relative motion to the host in mouse-compatible *packets*. The TouchPad generates roughly 40–80 packets per second. 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". The packet also reports information about the left and right "mouse" buttons.

Field	*Size (bits)*	*Range*	*Meaning*
ΔX	8	±127	Amount of horizontal finger motion
ΔY	8	±127	Amount of vertical finger motion
Left	1	0 or 1	State of left physical button or tap/drag gesture
Right	1	0 or 1	State of right physical button

Figure 2-1. 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. See the later sections of this *Guide* for details. (For example, in the PS/2 protocol, the packet actually reports 9-bit deltas, plus a third "middle" button which is not supported by current Synaptics products. Also, positive ΔY values correspond to upward motion in the PS/2 protocol, but to downward motion in the Serial and ADB protocols.)

In Relative mode, placing the finger on the pad 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 pad also simulates a brief click of the left mouse button, and the “tap-and-a-half” drag gesture simulates an extended motion with the left button held down. (Figure 2-17 of section 2.6.4 illustrates these gestures in a technical way; the on-line help that comes with Synaptics’ driver software has more user-oriented descriptions of the tapping gestures.)

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.

2.2. Packet rate

The Synaptics TouchPad reports approximately 40 packets per second by default. By setting the *Rate* bit of the mode byte (see section 2.5), the host can double the packet rate to approximately 80 packets per second.

The higher packet rate is preferable because it leads to the smoothest cursor motion. Versions 5.0 and later of the Synaptics TouchPad drivers for Windows® 95 and Windows NT® use the higher packet rate by default.

The TouchPad defaults to the lower packet rate for the benefit of 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.

2.3. Absolute mode

Synaptics TouchPads also support an *Absolute mode* of operation, where the TouchPad transmits an extended packet which reports the absolute finger position on the pad (X, Y), the finger pressure or contact area (Z), and various other information such as the state of the buttons. The Synaptics Windows 95 and Windows NT 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 Edge Motion™.

In Absolute mode, the TouchPad reports packets continuously at the specified packet rate, either 40 or 80 packets per second, 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.

The standard Absolute packet contains the following information:

Field	*Size (bits)*	*Range*	*Meaning*
X	13	0–6143	Horizontal finger position, 0 = far left
Y	13	0–6143	Vertical finger position, 0 = far bottom
Z	8	0–255	Pressure or contact area, 0 = no contact
W	4	0–15	Finger width and other state information
Left	1	0 or 1	State of left physical button, 0 = not pressed, 1 = pressed
Right	1	0 or 1	State of right physical button, 0 = not pressed, 1 = pressed
Gesture	1	0 or 1	Tap/drag gesture in progress, 0 = no gesture, 1 = gesture
Finger	1	0 or 1	Finger presence, 0 = no finger, 1 = finger

Figure 2-2. Contents of Absolute packet

2.3.1. 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 drag gestures, whereas 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 Left and Right button bits report the current state of the two 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 rather supply two external connector pins to which the system designer can attach buttons. These pins are labeled “Left” and “Right”; it is up to the system designer to attach the pins to appropriately placed buttons.

Some Synaptics TouchPads (the “MultiSwitch” pads) support two additional buttons labeled “Up” and “Down.” When these buttons exist, their state is also reported in the Absolute packet. (See section 3.6.2 for information on how these buttons are reported.)

The Finger bit reports the state of the firmware’s 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 drag gestures. (See section 2.6.4 for more details on the virtual button.)

Note that 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.

2.3.2. 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 2-3. All 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 2-3. Absolute X and Y coordinates

(Note: "$" indicates hexadecimal notation.)

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. The *typical edge margins* are suitable limits for deciding whether the finger is on the edge or in the interior area of the pad surface; the finger is in the interior if X and Y lie within the edge margin limits.

The following figure illustrates the various coordinate limits:

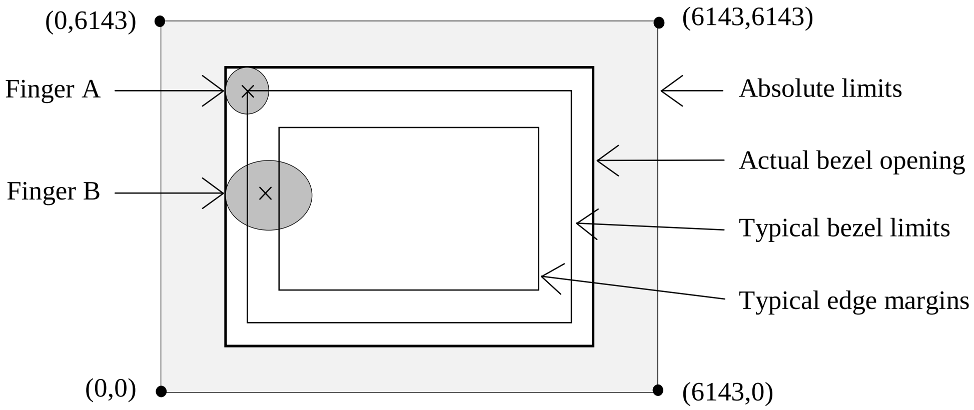


Figure 2-4. Coordinate limits (not to scale)

Note that 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 axis limits in Figure 2-3 are interchanged; for example, the X bezel limits for a portrait pad would be 1408–4448. Figure 2-11(*b*) of section 2.4.2 illustrates the portrait orientation.

The coordinate ranges in Figure 2-3 imply a resolution of 2000 dpi or more, depending on the physical size of the pad. (Section 2.4.3 lists the actual resolutions for different 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 2.6.2 gives some examples. In general, please remember that a TouchPad is *not* a graphics tablet; designers should not expect a compact, finger-operated device to match the stability, linearity, and repeatability of a precision pen-operated tablet.

2.3.3. Absolute mode Z values

The Z value reports the total finger capacitance, which is a function of the finger's contact area, which in turn is affected by the contact pressure and by the angle at which the finger is held. The following table illustrates some typical Z values.

Value	*Interpretation*
Z = 0	No finger contact.
Z = 10	Finger hovering near the pad surface.
Z = 30	Very light finger contact.
Z = 80	Normal finger contact.
Z = 110	Very heavy finger contact.
Z = 200	Finger lying flat on pad surface.
Z = 255	Maximum reportable Z; whole palm flat on pad surface.

Figure 2-5. Typical Z values

Note that 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.

For Synaptics TouchPad models that can sense pens as well as fingers, note that the Z scales for pens and fingers may be different. In fact, current pen TouchPads are unable to measure the pressure of pen contact; they report all pen strokes with a constant Z of 80.

2.3.4. Absolute mode W values

Newer Synaptics TouchPads support an optional value in the Absolute packet called "W." The W value is not available on all models of Synaptics TouchPads; when it is available, it 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 pad. The host can use W to distinguish among normal fingers, accidental palm contact, multiple fingers, and pens.

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

Value	*Needed capability*	*Interpretation*
W = 0	*capMultiFinger*	Two fingers on the pad.
W = 1	*capMultiFinger*	Three or more fingers on the pad.
W = 2	*capPen*	Pen (instead of finger) on the pad.
W = 3	—	*Reserved.*
W = 4–7	*capPalmDetect*	Finger of normal width.
W = 8–14	*capPalmDetect*	Very wide finger or palm.
W = 15	*capPalmDetect*	Maximum reportable width; extremely wide contact.

Figure 2-6. Absolute mode W values

Sections 2.4.2 and 2.4.4 show how the host can query for multi-finger, pen, and palm detection capability in a particular pad, as well as for the capability to report W at all.

If the *capPalmDetect* capability bit is set, then W values from 4 to 15 indicate that the pad has sensed a single finger of a particular width. The host can watch for especially wide “fingers” as evidence that the pad was activated by an accidental brush of the hand or palm rather than deliberate finger contact. (Note that 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 pad. In current TouchPads, X and Y will report the point on the pad midway between the fingers. (Future TouchPads may use a different convention, e.g., always following the first or the last finger to make contact with the pad.)

If the *capPen* capability bit is set, then a W value of 2 indicates that the pad is currently sensing a pen, not a finger. An object on the pad surface is considered a “finger” if it forms a significant contact area and is electrically attached to ground or to a large conductive body such as a human body. A “pen” is any other type of object, such as a non-conductive plastic stylus, that makes contact with the TouchPad surface. (Note that most Synaptics TouchPad products are unable to sense pens, and thus have *capPen* = 0; only certain “pen-input TouchPad” models are able to sense pens as well as fingers.)

When the *capPalmDetect, capMultiFinger,* or *capPen* 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. For example, if *capPen* = 0 and the TouchPad reports a W value of 2, the host should *not* treat the contact as a pen stroke, but rather as a normal finger stroke with no unusual properties.

When Z = 0, the X, Y, and W values cannot be measured and are reported as 0. When Z is positive but very small, e.g., less than 25, then the X and Y position, the finger width and count, and the finger/pen determination will be reported but they may not be very accurate.

Historical notes:

Older Synaptics TouchPads with the *capExtended* capability bit equal to 0 did not support "W mode." Those pads had no way to measure or report the width or count of fingers.

A very small number of early pen-capable TouchPads were built before the introduction of "W mode." On pen TouchPads with *capPen* = 1 but *capExtended* = 0, pen strokes are distinguished using the Z value: Z is 255 during a pen stroke, Z is in the range 1–254 during a finger stroke, and Z is 0 when no pen or finger is detected.

2.4. Information queries

The host can query the TouchPad for information describing the size, model, and capabilities of the TouchPad. The exact form of this query varies from one protocol to another, as described in later parts of this document. But the information itself is the same regardless of the protocol. The following sections describe the various available queries and the information they return.

2.4.1. TouchPad identification

The most basic query asks whether the device is a Synaptics TouchPad or some other mouse-compatible pointing device. In each protocol, this query is designed as a special command that can be sent to any mouse-compatible device, but which only a Synaptics TouchPad will recognize.

The Identify TouchPad query returns the following information to the host:

infoMajor

The primary or "major" version of the TouchPad device and firmware. Most older Synaptics TouchPads had a major version of 3; the modern Synaptics TouchPads described in this document have a major version of 4.

infoMinor

The minor version number starts over at 0 with each new major version, and increases by one whenever minor changes are made to the device or its firmware. In a complete version number such as "4.5", the major version is 4 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; modern TouchPads report much more detailed information about themselves in the queries described in the next few sections. New host software should not use the *infoModelCode* field.

When this *Guide* uses the phrase "versions $x.y$ and later" or "version $\geq x.y$," it refers to all TouchPads with greater major version, or equal major version and greater or equal minor version:

"version $a.b \geq x.y$" is equivalent to saying "($a > x$) **or** ($a = x$ **and** $b \geq y$)"

2.4.2. Model ID bits

Synaptics TouchPads starting with version 3.2 have supported a “model ID” query which allows the host to learn information about the physical type of the pad. The model ID consists of 24 bits divided into various bit-fields:

Bit 23	*Bit 22*	*Bit 21*	*Bit 20*	*Bit 19*	*Bit 18*	*Bit 17*	*Bit 16*
infoRot180	infoPortrait	infoSensor					

Bit 15	*Bit 14*	*Bit 13*	*Bit 12*	*Bit 11*	*Bit 10*	*Bit 9*	*Bit 8*
infoHardware							*Reserved*

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
infoNewAbs	capPen	infoSimplC	*Reserved*	infoGeometry			

Figure 2-7. TouchPad model ID bits

The model ID fields are defined as follows.

infoSensor (bits 21–16)

This 6-bit field identifies the type or model of TouchPad sensor; it allows the host to determine the size and physical type of the TouchPad. The following table lists the sensor types that have been defined as of this writing.

infoSensor	*Model no.*	*Definition*
0	—	Unknown sensor type
1	TM41*xx*134	Standard TouchPad
2	TM41*xx*156	Mini module
3	TM41*xx*180	Super module
7	*(discontinued)*	Flexible pad
8	TM41*xx*220	Ultra-thin module
9	TW41*xx*230	Wide pad module
11	TM41*xx*240	Stamp pad module
12	TM41*xx*140	SubMini module
13	*TBD*	MultiSwitch module
15	TM41*xx*301	Advanced Technology Pad
16	TM41*xx*221	Ultra-thin module, connector reversed
other	—	*Reserved*

Figure 2-8. Sensor types

See section 2.7 for more information about TouchPad model numbers.

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 standard sensor board might be mounted beneath an oval bezel, in which case the TouchPad would be ordered from Synaptics with special firmware that identifies 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 an oval-shaped Edge Motion zone instead of the usual rectangular zone.

infoGeometry	*Definition*
0	Unknown geometry
1	Standard rectangular geometry
2	Oval geometry
3–15	*Reserved*

Figure 2-9. Geometry types

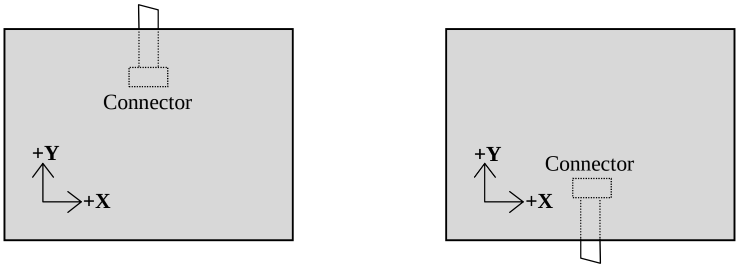
infoHardware (bits 15–9)

This 7-bit field is reserved for use by Synaptics.

infoRot180 (bit 23)

This bit is 0 for normal “Up” TouchPads, or 1 for 180°-reversed “Down” TouchPads designed to be mounted upside-down compared to the standard Synaptics product. (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 in order to work properly when mounted with the cable exiting downwards:



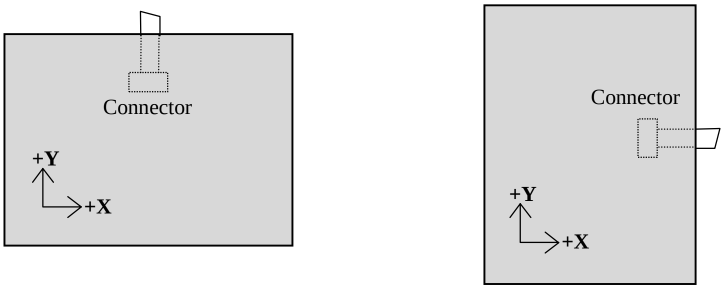
(a) TM41PU-134 (Up)
infoRot180 = 0

(b) TM41PD-134 (Down)
infoRot180 = 1

Figure 2-10. 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 2.3.2 are approximately interchanged X-for-Y on a Portrait pad.



(a) TM41PU-134 (Up)
infoPortrait = 0

(b) TM41PP-134 (Portrait)
infoPortrait = 1

Figure 2-11. Portrait orientation

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

infoNewAbs (bit 7)
This bit indicates that a new, improved Absolute packet format is available. It is 1 except for certain older PS/2 and Serial TouchPads; see the *historical notes* at the end of section 3.6.2, and the description of *PackSize* in section 2.5.

infoSimpleCmd (bit 5)
This bit is 1 except for certain older PS/2 TouchPads; see the *historical notes* at the end of section 3.5.2.

capPen (bit 6)
This bit is 0 for normal finger-only TouchPads, or 1 for TouchPads capable of sensing pens as well as fingers.

Historical notes:

Some very old TouchPads do not support the "model ID" query. Each protocol provides a way to tell whether or not a certain pad supports this query, as described below in the sections devoted to the respective protocols. If a pad does not support the model ID query, the pad can be assumed to have the following properties:

infoSensor = 0 (unknown)
infoGeometry = 1 (rectangular)
infoRot180 is unknown
infoPortrait = 0 (landscape)

infoNewAbs = 0
infoSimpleCmd = 0
capPen = 0

2.4.3. Coordinate resolution

Modern 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, the standard TM41*xxx*134 module has an *infoXupmm* of 85 units per millimeter, which means that moving the finger by one centimeter (10mm) on the TouchPad surface results in a change of approximately 850 units in the X coordinate as reported in Absolute mode.

The reported resolutions for various TouchPad models are shown in Figure 2-12.

Model	*infoSensor*	*Units per mm*	*Units per inch*	*Sensor area (mm)*
Standard	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 2-12. 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 (i.e., "DPI") are computed by multiplying units per mm by 25.4. In each case, the resolution is shown in the form "$X \times Y$". Please note that these resolutions are only approximate.

The sensor area is computed based on the bezel limits in Figure 2-3, which span 5472 – 1472 = 4000 units in X and 4448 – 1408 = 3040 units in Y. Dividing the total width of the pad in units by the number of units per millimeter gives the width in millimeters, as shown in the rightmost column of Figure 2-12. 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 2.3.2 for more discussion of coordinate limits.

The values shown in Figure 2-12 are for pads in the normal "landscape" orientation; for portrait-oriented pads (with *infoPortrait* = 1), the axes are exchanged. For example, a portrait UltraThin pad would have a 32.3 × 47.1mm sensor area and report a resolution of 94 × 85 units per millimeter.

Note that the resolution described in this section applies *only* to Absolute mode; in Relative mode, the resolution is variable and may depend on the protocol in use. See section 2.6.3 for information about the resolution in Relative mode.

Historical notes:

Older TouchPads do not support the resolution query. Each protocol provides a way to tell whether or not a certain pad supports the query. If the pad does not support the resolution query, use the table of Figure 2-12 indexed by the *infoSensor* code obtained from the model ID query (section 2.4.2), or simply assume a default resolution of 85 × 94 units per millimeter.

2.4.4. Extended capability bits

Modern Synaptics TouchPads support an “extended capability” query which returns to the host 16 bits indicating the presence or absence of various advanced features. In typical modern pads, the capability bits will be $8013 (hexadecimal), i.e., bits 15, 4, 1, and 0 are set and all other bits are clear.

The capability bits are arranged as follows:

Bit 15	*Bit 14*	*Bit 13*	*Bit 12*	*Bit 11*	*Bit 10*	*Bit 9*	*Bit 8*
cExtended	—	—	—	—	—	—	—

Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
—	—	—	cSleep	cFourBtn	—	cMultiFing	cPalmDet

Figure 2-13. TouchPad extended capability bits

Bits shown as “—” in Figure 2-13 are reserved for future use. The host should ignore the values of reserved bits when reading the capability bits.

capExtended (bit 15)

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; see the *historical notes* below. The *capExtended* bit also signifies that the TouchPad supports “W mode” as described in sections 2.3.4 and 2.5.

capSleep (bit 4)

For the PS/2 protocol, the *capSleep* bit is set if sleep mode is supported. See the discussion of the “Sleep” bit in section 2.5.

For other protocols, this capability bit is reserved.

capFourButtons (bit 3)

For the PS/2 protocol, this bit is set if the pad is a “MultiSwitch” pad which supports four mouse buttons labeled Left, Right, Up, and Down. In the PS/2 protocol, the Up and Down buttons are reported only during Absolute Mode with the *Wmode* bit set. See section 3.6.2.

For other protocols, this capability bit is reserved.

capMultiFinger (bit 1)

This bit is set if multi-finger detection is supported. The pad is then able to count the number of simultaneous fingers on the pad and report the finger count via the W field of the Absolute packet. If this bit is 0, the pad does not support multi-finger detection; any finger contact will be assumed to be a single finger, with the position reported as the midpoint between all actual fingers, and, if *capPalmDetect* is set, with W reporting a (typically 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.

Note that the *capPen* bit is used in the same way as an extended capability bit, but it is reported as part of the “model ID” query response (section 2.4.2).

Historical notes:

The extended capability bits are a relatively recent addition to the Synaptics TouchPad product line. All TouchPads starting with version 4.5 have extended capability bits; some models of older 4.*x* firmware also have capability bits, but there are many older pads in the field which do not support capability bits. In those older pads, the capability bit field was fixed at $5555 (hexadecimal). In very old pads (prior to version 3.2), this field was used to hold adjustable edge margin information, with $5555 as the power-up default value. Starting with version 3.2, the actual edge margins became fixed at the positions shown in Figure 2-4, and the host-readable $5555 value became a vestige.

To determine whether or not a Synaptics TouchPad supports capability bits, use the following procedure:

1. Read the *infoMajor* version number as described in section 2.4.1. If 3 or below, assume the capability bits are $0000. If 4 or above, continue with step 2.
2. Perform the “extended capability” query, to receive a 16-bit field.
3. If bit 15 of the received 16-bit field is 0, assume the capability bits are $0000.
4. If bit 15 of the received 16-bit field is 1, use the received field as the capability bits.

2.4.5. Serial numbers

Synaptics plans soon to include a unique host-readable serial number in each new TouchPad. Even though serial numbers are not yet in production, the TouchPad interface defines a protocol for querying a device for its 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 which have not been serialized (including all TouchPads produced so far at Synaptics), the serial number query will return a default value of zero.

2.4.6. Reading the mode byte

Synaptics TouchPads support a “mode byte” query which returns to the host the current value of the TouchPad mode byte. The mode byte is described in section 2.5 below.

2.5. Mode byte

The Synaptics TouchPad has a small set of configurable features which are encapsulated in the *TouchPad mode byte,* an 8-bit bit mask which the host can set to any value using a special command. The exact forms of the commands to set and read the mode byte vary from one protocol to another, as described in later parts of this document. The layout of the mode byte itself, however, is fairly consistent between protocols.

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. Also:

- In the PS/2 protocol, the PS/2 Reset ($FF) and Set Defaults ($F6) commands clear the *Absolute* bit to 0 but do not affect the other mode bits.
- In the Serial protocol, the RTS handshake clears the entire mode byte to $00.

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	—	—	Baud/Sleep	DisGest	PackSize	Wmode

Figure 2-14. TouchPad mode byte

Bits shown as "—" in Figure 2-14 are reserved for future use. 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 2.1 and 2.3.

Rate (bit 6)

This bit is 0 to select a low packet rate of approximately 40 packets per second, or 1 to select a high packet rate of approximately 80 packets per second. This bit is valid in all protocols; it is valid in both Relative and Absolute mode. See section 2.2.

Baud / Sleep (bit 3)

For the Serial protocol, this is the *Baud* bit. The *Baud* bit is 0 to select 1200 baud packet reporting, or 1 to select 9600 baud packet reporting. The *Baud* bit should be set whenever the *Rate* bit or the *Absolute* bit is set. See section 4.2 for more information about the Serial baud rate.

For the PS/2 protocol, this is the *Sleep* bit. The *Sleep* bit is 0 for normal operation, or 1 for "sleep" mode. When sleep mode is enabled, the pad goes into a low-power idle state in which it ignores finger activity. In sleep mode, only button presses will cause the pad to generate a motion packet. When the *Sleep* bit is changed from 1 to 0, the pad may need to spend 300–1000ms recalibating before finger sensing will resume. The Synaptics drivers use sleep mode for ACPI power management support. The *Sleep* bit is defined only in Relative mode on pads whose *capSleep* capability bit is set; in Absolute mode and in pads without this capability, the bit is reserved and should be left at 0. Also, *DisGest* should be set whenever *Sleep* is set.

The *Baud / Sleep* bit is defined only for the PS/2 and Serial protocols; for other protocols, the bit is reserved and hence should be left at its default value of 0.

DisGest (bit 2)

This bit is 0 to enable “tap” and “drag” gesture processing, or 1 to disable detection of tap and drag 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. The *DisGest* bit is implemented only for 4.*x* and later TouchPads (i.e., when *infoMajor* ≥ 4); for older pads, the bit is reserved.

PackSize (bit 1)

For the Serial protocol, this bit is 0 to select six-byte Absolute packets, or 1 to select seven- or eight-byte packets (per the *Wmode* bit). This bit is defined only in Absolute mode in Serial TouchPads with the *infoNewAbs* bit set (see section 2.4.2); in Relative mode, when *infoNewAbs* = 0, and at all times in non-Serial protocols, this bit is reserved and should be left at 0.

Wmode (bit 0)

This bit is 0 to select normal Absolute mode packets, or 1 to select enhanced Absolute packets that contain the “W” value as well as X, Y, and Z. See section 2.3.4 for more information about W; see the later sections on the particular protocols for more information about packet formats. This bit is defined only in Absolute mode on pads whose *capExtended* capability bit is set; in Relative mode and in pads 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	Version 4.*x* or later	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 2-15. PS/2 mode byte values

The following table shows some typical values for the mode byte in the Serial protocol:

Value (hex)	*When to use*	*Effect*
\$00	Always OK	Relative mode (1200 baud)
\$04	Version 4.*x* or later	Relative mode with gestures disabled (1200 baud)
\$48	Always OK	Relative mode with high packet rate (9600 baud)
\$88	Versions before 3.2	Absolute mode (9600 baud, 6-byte packets)
\$8A	*capExtended* = 0, ≥ v3.2	Absolute mode (9600 baud, 7-byte packets)
\$8B	*capExtended* = 1	Absolute mode (9600 baud, 8-byte packets with W)
\$C8	Versions before 3.2	Absolute mode (high packet rate, 6-byte packets)
\$CA	*capExtended* = 0, ≥ v3.2	Absolute mode (high packet rate, 7-byte packets)
\$CB	*capExtended* = 1	Absolute mode (high packet rate, 8-byte packets/W)

Figure 2-16. Serial mode byte values

Historical notes:

The *Absolute*, *Rate*, and *Baud* bits have always been present in Synaptics TouchPads. The other mode bits have been introduced over time as new optional features were added to the TouchPad product.

Older (3.*x*) versions of the TouchPad actually supported four bytes of mode information; the mode byte described in Figure 2-14 is the descendent of the original “mode byte 2.” The four original mode bytes provided considerably more control over the pad’s operation in Relative mode. The host could independently control the detection of taps, drags, and corner-zone taps; it could turn Edge Motion on and off and adjust the sizes of the edge zones; it could adjust the Z threshold for finger detection. In the modern TouchPad, all of these jobs are left to the host driver, which is in a position to provide a higher quality implementation of these features with even greater adjustability. The TouchPad itself implements only the simple gestures and other features necessary for basic mouse emulation in the presence of a non-Synaptics-aware driver. Section 7.1.1 describes the layout of the four original mode bytes.

The *DisGest* bit is now the only mode bit intended for use in Relative mode; note that because the PS/2 reset sequence does not affect this mode bit, a computer BIOS can set this bit to ensure that gestures are disabled even when a non-Synaptics mouse driver is used. (However, Synaptics recommends that *DisGest* be left at 0; experience shows that most TouchPad users prefer to have tapping gestures enabled.)

2.6. Principles of operation

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

2.6.1. Sensing finger presence

The standard Synaptics TouchPad uses *capacitive sensing* to detect fingers. Whenever two electrical conductors are placed flat against each other 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 a thin insulating Mylar label between. 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. By measuring the total amount of capacitance, the TouchPad can also determine the approximate finger pressure "Z". (This works because the harder the user presses down, the more the finger flattens out against the pad; larger contact area leads to larger total capacitance.)

2.6.2. 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 TouchPad's capacitance measurements. In Relative mode, the pad applies several filtering and acceleration algorithms as described in this section and section 2.6.3. 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.

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

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

in the simple windowed average algorithm. (Here U_{new} is the most recent finger position, and U_{prev} is the previous finger position, i.e., 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} ; hence, 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 $\frac{1}{4} U_{new} + \frac{3}{4} X_{prev}$ 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. 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" equation *(2)* above for U_{new} as a function of X_{new} and X_{prev}.

2.6.3. 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 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 in the industry, believe it or not, as *mickeys*.

Most mouse drivers include an "acceleration" feature which varies S_C based on the mouse speed in order 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 2.3.2 describes Absolute position units. For a standard model TM41*xxx*134 TouchPad module, the overall speed factor is about 240 mickeys per inch or 9 mickeys per millimeter at full speed. Figure 2-12 of section 2.4.3 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.

2.6.4. Sensing tapping gestures

For the purposes of this document, a *gesture* is any motion or action of the finger that the system interprets as something other than regular cursor motion. The TouchPad itself supports *tap* and *drag* gestures for simulating mouse button clicks using the finger; the Synaptics TouchPad drivers add several more gestures such as "Virtual Scrolling™."

To detect a tapping gesture, the TouchPad looks for Z to increase beyond the touch threshold and then to return to 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 2-17 illustrates tap and drag detection.

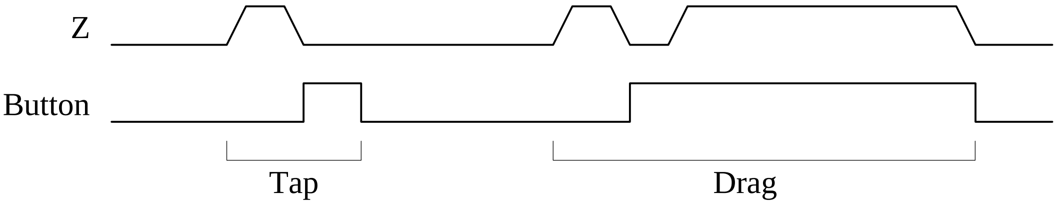


Figure 2-17. Tap and drag gestures

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 drag gesture. Higher Z indicates the presence of a finger on the pad; a high level on the Button signal represents a simulated left button press as reported by the TouchPad.

To move the cursor a long distance, the user may need to make several finger stokes. But because a drag gesture ends when the user lifts the finger, the entire drag must occur in a single stroke. Synaptics TouchPads offer a feature called *Edge Motion*™ to assist with long-distance drags: If, during a drag, 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.

2.6.5. TouchPad calibration

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

2.6.6. Host interface to 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.

In modern PC's, the pointing device talks to the computer using either the PS/2 protocol (see section 3) or the RS-232 serial protocol (see section 4). "Bus mice" were popular in earlier days but have since fallen out of use. Some early serial mice used different protocols, but today serial mice have mostly converged on the protocol described in section 4, particularly Figure 4-16. In the future, USB (the Universal Serial Bus) may replace the PS/2 and RS-232 protocols, but as of this writing USB has had little penetration in the field of pointing devices. Contact Synaptics for more information about USB TouchPads.

Figure 2-18 illustrates a typical path that motion data travels on its way from a PS/2 TouchPad to a Windows® 95 application:

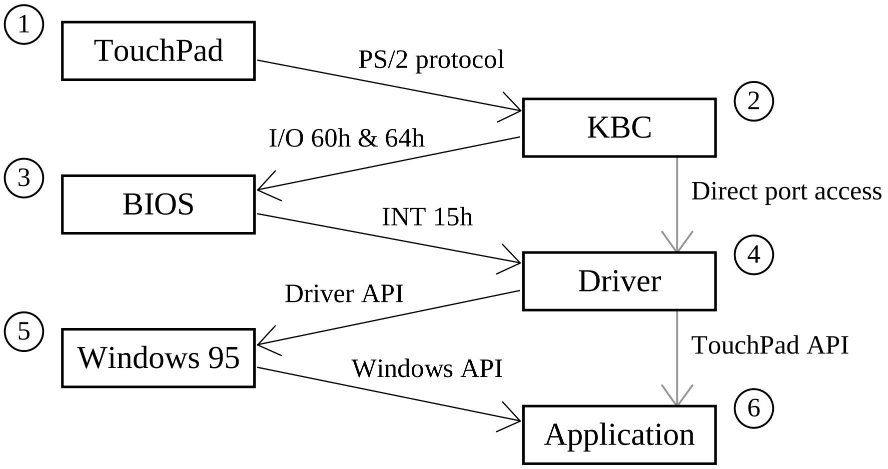


Figure 2-18. TouchPad to host data path (PS/2)

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 device as well as the keyboard. The KBC communicates with the TouchPad via CLK and DATA wires as described below in section 3.2. 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 device does not try to talk.

The KBC communicates to the main CPU 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 3.7.1 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 nowadays many drivers talk directly to the KBC instead of using the BIOS facilities. (Note the gray arrow in Figure 2-18 between the driver and the KBC.)

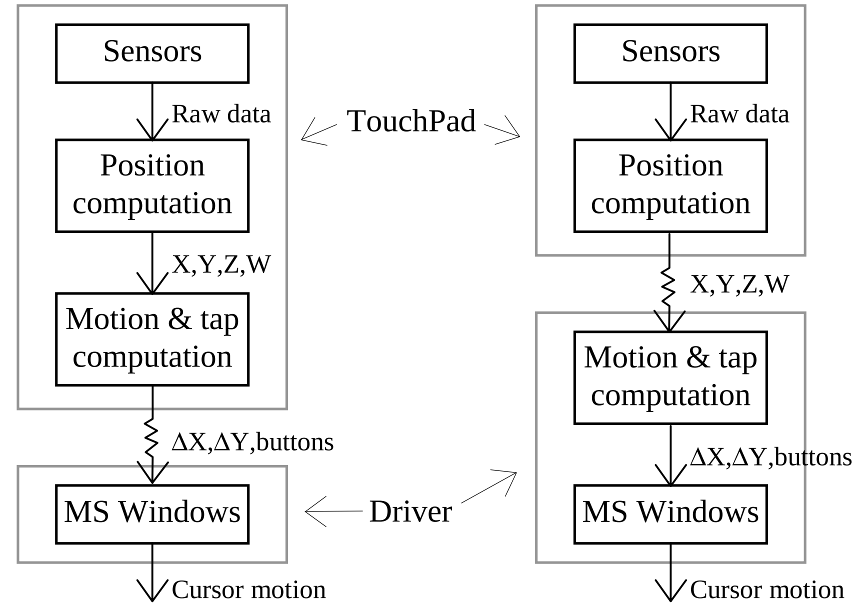
Frank van Gilluwe’s book, *The Undocumented PC*, is an excellent reference to the KBC and the BIOS from the point of view of host software.

For a Serial TouchPad, the path is similar except that a UART chip services the serial ports instead of the KBC. Also, serial pointing device drivers generally always use direct port access to the UART, since there are only rudimentary serial port facilities in the standard BIOS.

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 press 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).

Figure 2-19 compares the processing steps taken when the TouchPad is used with a standard mouse driver and with the Synaptics Windows 95 or NT TouchPad driver.



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

Figure 2-19. 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, W) position information into (ΔX, ΔY, 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 two advantages: First, the driver executes on a powerful CPU and so is able to use better algorithms with better pointing performance; second, 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 has its own API which applications can use 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 2-18. Section 6 of this *Guide* describes the Synaptics TouchPad driver API.

2.7. Synaptics TouchPad model numbers

Synaptics offers a wide variety of TouchPad products. These products are identified by the *model number*, an alphanumeric string that encodes the board size and type, host communication protocol, label color, and general feature set. This section provides a rough description of the model number scheme; contact Synaptics for more complete information on the available models.

A typical Synaptics TouchPad model number looks like this:

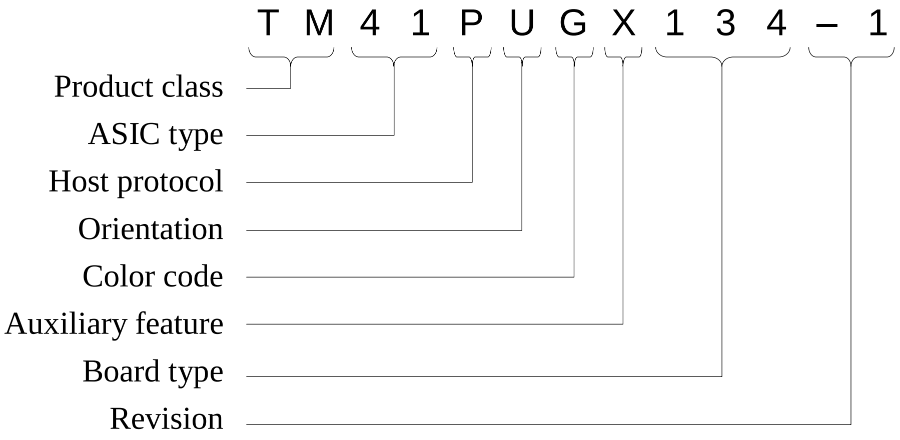


Figure 2-20. TouchPad model number

The primary product class is **TM** (TouchPad module, the basic capacitive finger-operated TouchPad).

The ASIC type **41** identifies newer pads employing the T1004 ASIC. The ASIC type **21** signifies an older pad using a T1002 ASIC with a separate microprocessor chip.

Possible host protocol letters include **P** (PS/2), **B** (PS/2-and-Serial combo), and **A** (ADB). Other less common letters are **S** (Serial-only) and **T** (TTL-level serial; see section 4.1.1).

The orientation letter is **U** (Up), **D** (Down), or **P** (Portrait). The "up" orientation is defined as the orientation in which the cable exits from the top edge of the board, i.e., away from the user, as shown in Figure 2-10(*a*).

The color code letter identifies the color of the mylar label that forms the touch-sensitive surface of the pad. For example, color **G** is the standard Synaptics Slate color; contact

Synaptics for information about additional colors. (Note that the color letter is the *only* part of the model number which cannot be read out using the host queries described in section 2.4.)

The auxiliary feature letter is an optional letter that follows the color code; it is present if the TouchPad has unusual or custom-ordered hardware or firmware features.

The board type identifies the physical size, shape, and thickness of the TouchPad sensor board. Type **134** signifies the standard 65 × 49 × 1.9mm board. Figure 2-8 lists some additional board type codes.

The revision suffix is a single number that encodes the overall product revision level. It plays a similar role to the major/minor version number of section 2.4.1; however, the revision suffix can change while the version number stays the same, for example to reflect a change in the manufacturing process. Also, the version number can change while the revision suffix stays the same, for example when a version of firmware is developed at Synaptics but never put into production.

For further information about model numbers, or to order samples of any TouchPad model, contact Synaptics at (408) 434-0110 or sales@synaptics.com.

3. PS/2 Protocol

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.

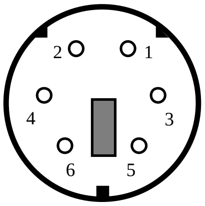
TouchPads integrated into notebook computers typically use the PS/2 protocol.

3.1. 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-collector" signals; they are normally held at a high (+5V) level by a 5–10K pull-up resistor on the host, but either the host or the TouchPad device can pull them 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 allowed to float high. The CLK and DATA lines should have a total capacitance of no more than 500pF to ensure that the 5–10K pull-up resistor is able to drive them to a high voltage level in a reasonable time.

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 3-1. PS/2 cable pinout

On the Synaptics Standard PS/2 TouchPad module TM41P*xx*134, the 8-pin FFC connector has the following pinout:

1	2	3	4	5	6	7	8
Power +5V	PS/2 DATA	PS/2 CLK	Right Switch	Left Switch	Ground 0V	N/C	N/C

Figure 3-2. PS/2 module connector pinout

The button switch inputs (pins 4 and 5) include pull-ups to +5V on the module, and should be grounded when the corresponding switch is closed (pressed).

Pins 7 and 8 of the 8-pin connector are reserved for future use; they should be left unconnected by the host.

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

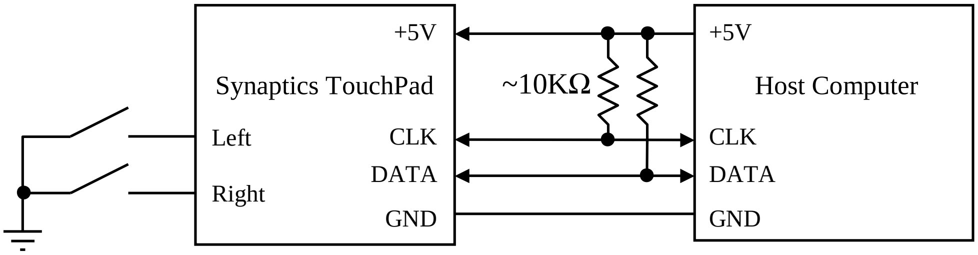


Figure 3-3. PS/2 system diagram

3.1.1. Connector pinouts

Figure 3-2 showed the pinout of the 8-pin connector for the TM41P*xx*134 board. Synaptics also offers a variety of other board types each with its own connector and pinout.

The “Ultra-Thin” TouchPad TM41P*xx*220 uses either a six- or a twelve-pin connector:

1	2	3	4	5	6
CLK	DATA	Left Sw	Right Sw	Ground	+5V

1	2	3	4	5	6	7	8	9	10	11	12
CLK		DATA		Left Switch		Right Switch		Ground		+5V	

Figure 3-4. PS/2 UltraThin module connector pinouts

The “Mini” TouchPad TM41P*xx*156 uses a six-pin connector:

1	2	3	4	5	6
Ground	CLK	DATA	+5V	Left Sw	Right Sw

Figure 3-5. PS/2 Mini module connector pinout

The “SubMini” TouchPad TM41P*xx*140 uses a twelve-pin connector:

1	2	3	4	5	6	7	8	9	10	11	12
Gnd	Left Switch		+5V		DATA		CLK		Right Switch		Gnd

Figure 3-6. PS/2 SubMini module connector pinout

3.2. 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 1’s. 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.

3.2.1. Output to host

The device can transmit a byte to the host whenever the bus is idle. The device cannot transmit if the bus is inhibited or in the request-to-send state.

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

If the bus is in the host request-to-send state, the device 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 device 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 device changes the state of the DATA wire during the CLK high period.

If the host inhibits the bus by holding CLK low for at least 100µs during a device transmission, the device will recognize this and abort the transmission. The device 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 device will resend the interrupted byte as soon as the inhibit is released. (An ACK ($FA) reply to a command or argument byte is simply 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.

The host may hold CLK low after the transmission, effectively extending clock 11, to inhibit the device 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 shown in Figure 3-7 takes about one millisecond, the host should inhibit the bus for less than one millisecond per byte on average in order to achieve the maximum packet rate.

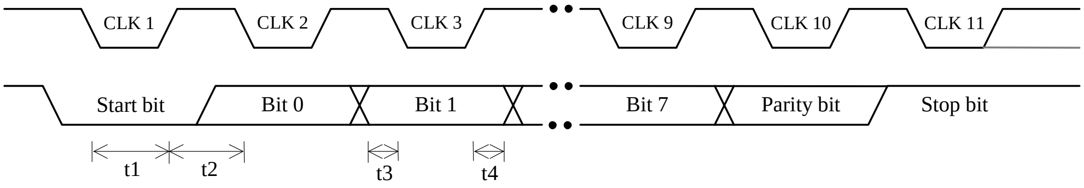


Figure 3-7. PS/2 output waveforms

In Figure 3-7, the CLK signal is low for 30–50µs (t1) and high for 30–50µs (t2) in each bit cell. DATA will be valid at least 5µs before the falling edge (t3) and at least 5µs after the rising edge (t4) of the clock. Device actions are shown in black; host actions are in gray.

3.2.2. Input from host

The host signals its intent to transmit a command or argument byte by holding CLK low for at least 100µs, then pulling DATA low and releasing CLK, thus putting the bus into the host request-to-send state. The device checks for this state at least every 10ms (t5). When the device detects a request-to-send, it pulses CLK low 11 times to receive a byte. The host is expected to change the DATA line while CLK is low; the device samples the DATA line while CLK is high. The host can abort the transmission midway through by holding CLK low for at least 100µs at any time before the eleventh CLK pulse.

After the tenth clock, the device 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 50µs (t12) to inhibit the device until the host is ready to receive the reply. If the device finds DATA low during the stop bit, a framing error has occurred; the device continues to clock until DATA goes high, then sends a Resend to the host as described in the next section.

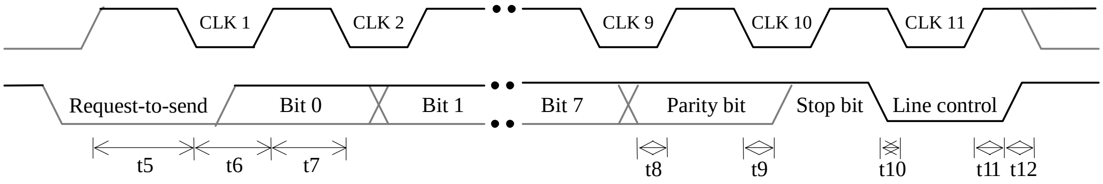


Figure 3-8. PS/2 input waveforms

In Figure 3-8, the CLK signal is low for 30–50µs (t6) and high for 30–50µs (t7) in each bit cell. DATA is sampled when CLK is high, and must be valid no later than 1µs after the rising edge of the clock (t8 ≥ –1µs, t9 ≥ 0µs). In the line control bit, DATA goes low at least 5µs before the falling edge (t10) and stays low at least 5µs after the rising edge (t11) of the clock. Device actions are shown in black; host actions are in gray.

3.2.3. Acknowledgement of commands

Each command or argument byte produces at least one response byte from the device. For every command or argument byte except the Resend ($FE) command, the response always begins with an "Acknowledge" or ACK ($FA) byte. 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 device responds within 25ms, 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 20ms. The Reset ($FF) command is an exception, where the $FA and $AA bytes are separated by up to 500ms 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 3.2.1.

If the device receives an erroneous input (an invalid command or argument byte, a byte with incorrect parity, or a framing error), the device sends a Resend ($FE) response to the host instead of an ACK. If the next input from the host is also invalid, the device sends an Error ($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 command, not just the argument byte.

On many PC's, the PS/2 port will also report a manufactured $FE response if the device does not send a response after a suitable timeout, or if the device 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.

Historical notes:

Parity errors and framing errors are detected properly by current Synaptics TouchPads (version 4.*x* and later), but some earlier TouchPads ignored parity and framing errors. Likewise, earlier TouchPads did no range checking on Set Resolution and Set Sample Rate argument bytes; modern 4.*x* TouchPads will reject out-of-range Resolution arguments but still do no range checking on Sample Rate arguments.

3.3. Power-on reset

At power-on, the PS/2 device performs a self-test and calibration, then transmits the completion code $AA and ID code $00. If the device 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 device until the calibration/self-test is complete.

Power-on self-test and calibration takes 300–1000ms. Self-test and calibration following a software Reset command takes 300–500ms. (In the standard Synaptics TouchPad device, the delays are nominally 750ms and 350ms, respectively.)

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 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 300–1000ms required for power-on calibration.

See section 2.6.5 for further comments on the TouchPad calibration process.

The reset state of the device is as follows:

- Reported sample rate is 100 samples per second (see page 34).
- Reported resolution is 4 counts per mm (see page 35).
- Scaling is 1:1.
- Stream mode is selected.
- Data reporting is disabled.
- Absolute mode is disabled.

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 pad 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 pad to spontaneously revert from Absolute to Relative mode. If the host notices the pad spontaneously reverting to the Relative mode packet format, it should reinitialize the pad in the same manner as at power-up.

Historical notes:

In older (version 3.*x* and earlier) TouchPads, the Reset command cleared the *Absolute* mode bit as described above, but the Set Defaults command did not affect any of the mode bits.

In very old 2.*x* TouchPads, the initial $AA, $00 announcement at power-on was omitted. (These pads transmitted $AA, $00 only in response to a Reset command.)

3.4. Command set

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

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

As elsewhere in this document, “$” signifies hexadecimal notation.

$FF	**Reset.** Perform a software reset and recalibration as described in section 3.3 above. Response is ACK ($FA), followed by $AA, $00 after a calibration delay of 300–500ms.
$FE	**Resend.** The host sends this command when it detects invalid output from the device. The device retransmits the last packet of data, for example, a three- or six-byte motion data packet, a one-byte response to the Read Device Type ($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; however, if the last output from the device was an ACK with no additional data bytes, Resend responds with an ACK. The device will send a Resend ($FE) to the host if it receives invalid input from the host; see section 3.2.3.
$F6	**Set Defaults.** Restore conditions to the initial power-up state. This resets the sample rate, resolution, scaling, and Stream mode to the same states as for the Reset ($FF) command, and disables the device. This command disables Absolute mode, but it leaves the rest of the TouchPad mode byte unaffected.
$F5	**Disable.** Disable Stream mode reporting of motion data packets. All other device operations continue as usual.
$F4	**Enable.** Begin 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 device 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 device 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 device. Thus, a complete Set Sample Rate = 10 command consists of $F3 from the host, $FA from the device, $0A from the host, and $FA from the device.

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 2.5.

$F2 **Read Device Type.** The response is an ACK ($FA) followed by a $00 device ID byte.

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

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

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

$EB **Read Data.** The device replies with an ACK ($FA) followed by a three- or six-byte motion data packet as described below in section 3.6. 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; section 2.5), there is little point in polling more often than that.

$EA **Set Stream Mode.** Switch 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.

$E9 **Status Request.** Response is an ACK ($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 3-9. Standard 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.

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 section 3.5 below.

$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 2.3.2, 2.4.2, and 3.6.1 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.
$E6	**Set Scaling 1:1.** Clears the PS/2 “scaling” bit, as described above.
other	If the device receives an invalid command byte, it replies with a Resend ($FE) byte. If it immediately receives a second invalid command, it replies with an Error ($FC) byte.

3.5. 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 3.4. In order 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 which 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.

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 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 3.4 (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, it is wise to precede the sequence with an “inert” command such as Disable or Set Scaling 1:1 just in case the most recent command sent to the device happened to be 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

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

3.5.1. Information queries

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 many 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:

$00 **Identify TouchPad.** See section 2.4.1. 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.

	Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Byte 1	infoMinor							
Byte 2	0	1	0	0	0	1	1	1
Byte 3	infoModelCode				infoMajor			

Figure 3-10. 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 ($00).

$01 **Read TouchPad Modes.** See section 2.4.5. The first two bytes of the response are the constants $3B and $47, respectively, and the third byte is the TouchPad mode byte.

	Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Byte 1	0	0	1	1	1	0	1	1
Byte 2	0	1	0	0	0	1	1	1
Byte 3	Mode byte (Figure 2-14)							

Figure 3-11. PS/2 Read Modes response

Historical note: On pre-4.*x* TouchPads, the first byte reported "mode byte 1" as described in section 7.1.1.

$02 **Read Capabilities.** See section 2.4.4. The first and third byte 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	Capability bits 15–8 (Figure 2-13)							
Byte 2	0	1	0	0	0	1	1	1
Byte 3	Capability bits 7–0 (Figure 2-13)							

Figure 3-12. PS/2 Read Capabilities response

Historical note: On pre-4.*x* TouchPads, this query returned the Edge Motion margin adjustment factors instead of capability bits. The adjustment factors were $55, $55 by default; starting at version 3.2, these bytes were no longer adjustable and were fixed at $55, $55.

$03 **Read Model ID.** See section 2.4.2. The three response bytes hold the 24-bit TouchPad model ID. Model ID bit 8 (the least significant bit of the second byte) is always 0 in this protocol.

	Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Byte 1	Model ID bits 23–16 (Figure 2-7)							
Byte 2	Model ID bits 15–8 (Figure 2-7)							0
Byte 3	Model ID bits 7–0 (Figure 2-7)							

Figure 3-13. PS/2 Read Model ID response

Historical note: TouchPads prior to version 3.2 did not support the Model ID query. Those pads returned $47 in the second byte of the response to query number $03. Also, non-Synaptics devices will return $03 in the second byte of this query. Hence, host software can check for the presence of model ID information by examining the least significant bit of the second byte: That bit will be 0 if model ID information is present, or 1 if the information is not present. See the *historical notes* at the end of section 2.4.2 for suitable default model ID information to use when this query is not supported.

$06 **Read Serial Number Prefix.** This query returns the first twelve bits (bits 35–24) of the TouchPad's unique serial number. See section 2.4.5; note that the pads currently produced by Synaptics do not yet include serial numbers. The bits shown as "—" in the figure are reserved and hold undefined data.

	Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Byte 1	Serial Number bits 31–24							
Byte 2	Serial Number bits 35–32				—	—	—	—
Byte 3	—	—	—	—	—	—	—	—

Figure 3-14. PS/2 Serial Number Prefix response

Historical note: This query is supported only in versions 4.*x* and later of Synaptics TouchPads. Also, 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 attempt the Serial Number Prefix query only if *infoMajor* ≥ 4, and it 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.

	Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Byte 1	Serial Number bits 23–16							
Byte 2	Serial Number bits 15–8							
Byte 3	Serial Number bits 7–0							

Figure 3-15. PS/2 Serial Number Suffix response

$08 **Read Resolutions.** See section 2.4.3. 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.

	Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Byte 1	infoXupmm							
Byte 2	1	—	—	—	—	—	—	—
Byte 3	infoYupmm							

Figure 3-16. PS/2 Read Resolutions response

Historical note: This query is supported only in some 4.*x* versions of the Synaptics TouchPad. The host should only issue this query if *infoMajor* ≥ 4; also, the result should only be considered valid if bit 7 of byte 2 of the response is 1 and the *infoXupmm* and *infoYupmm* bytes are both non-zero.

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

Historical notes:

In versions before 3.2, the Synaptics TouchPad recognized one additional special command sequence which was used by some mouse drivers to identify three-button mice. This sequence was supported to allow use of a standard mouse driver with a TouchPad that had been configured by other means to set the “3-button” mode bit. In practice, this feature was never used (and the “3-button” bit has been discontinued); thus, to avoid confusion, version 3.2 and later no longer recognize this command sequence. Therefore, non-Synaptics drivers now recognize the TouchPad as a generic two-button mouse.

3.5.2. Mode setting sequence

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 as described in section 2.5 and Figure 2-14.

For example, to set the mode byte to $C1 (Absolute mode, high packet rate, Wmode 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 ($FA) acknowledgments. Note that, as described at the beginning of section 3.4, it is important to ensure that the device is disabled ($F5) before sending this command sequence; to receive Absolute mode packets, follow this sequence with an Enable ($F4) command.

Historical notes:

Older Synaptics TouchPads supported up to four mode bytes; the sequences to set those bytes ended with Set Sample Rate commands with arguments other than $14. On the present (4.*x*) TouchPad, sequences of four Set Resolution commands followed by a Set Sample Rate with any argument other than $14 have an undefined effect on the TouchPad and should not be used.

Some older Synaptics TouchPads also supported a second way to read or write the mode byte using PS/2 command code $E1. See section 7.1.2.

3.6. Data reporting

The Synaptics TouchPad supports two formats 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 20ms (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 20ms between bytes it can assume the delay comes at a packet boundary.

3.6.1. Default packet format

In the default Relative format, 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 3-17. 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.

Y delta: This is the amount of motion ΔY that has occurred in the Y (vertical) direction. Upward motion is positive, downward is negative.

Note that 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 drag 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. Because standard Synaptics TouchPads only support two buttons, the “middle button” bit is always zero.

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 standard Synaptics pad; see section 2.6.3 for further details.

3.6.2. 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 pad, as well as the Z (pressure) value and various other measurements and status bits. Section 2.3 discusses the contents of the Absolute mode packet in great detail.

Modern PS/2 TouchPads support two different Absolute packet formats, depending on the setting of the *Wmode* bit of the TouchPad mode byte (section 2.5). Some very old TouchPads actually use a different packet format; see the *historical notes* below.

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 3.2.1 for further information.

The Absolute X/Y/Z packet format when *Wmode* = 0 is shown in Figure 3-18:

<table>
<tr><th></th><th><i>Bit 7</i></th><th><i>Bit 6</i></th><th><i>Bit 5</i></th><th><i>Bit 4</i></th><th><i>Bit 3</i></th><th><i>Bit 2</i></th><th><i>Bit 1</i></th><th><i>Bit 0</i></th></tr>
<tr><td><i>Byte 1</i></td><td>1</td><td>0</td><td>Finger</td><td><i>Reserved</i></td><td>0</td><td>Gesture</td><td>Right</td><td>Left</td></tr>
<tr><td><i>Byte 2</i></td><td colspan="4">Y position 11..8</td><td colspan="4">X position 11..8</td></tr>
<tr><td><i>Byte 3</i></td><td colspan="8">Z pressure 7..0</td></tr>
<tr><td><i>Byte 4</i></td><td>1</td><td>1</td><td>Y pos 12</td><td>X pos 12</td><td>0</td><td>Gesture</td><td>Right</td><td>Left</td></tr>
<tr><td><i>Byte 5</i></td><td colspan="8">X position 7..0</td></tr>
<tr><td><i>Byte 6</i></td><td colspan="8">Y position 7..0</td></tr>
</table>

Figure 3-18. 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 3-19:

<table>
<tr><th></th><th><i>Bit 7</i></th><th><i>Bit 6</i></th><th><i>Bit 5</i></th><th><i>Bit 4</i></th><th><i>Bit 3</i></th><th><i>Bit 2</i></th><th><i>Bit 1</i></th><th><i>Bit 0</i></th></tr>
<tr><td><i>Byte 1</i></td><td>1</td><td>0</td><td colspan="2">W value 3..2</td><td>0</td><td>W val 1</td><td>Right</td><td>Left</td></tr>
<tr><td><i>Byte 2</i></td><td colspan="4">Y position 11..8</td><td colspan="4">X position 11..8</td></tr>
<tr><td><i>Byte 3</i></td><td colspan="8">Z pressure 7..0</td></tr>
<tr><td><i>Byte 4</i></td><td>1</td><td>1</td><td>Y pos 12</td><td>X pos 12</td><td>0</td><td>W val 0</td><td>R/D</td><td>L/U</td></tr>
<tr><td><i>Byte 5</i></td><td colspan="8">X position 7..0</td></tr>
<tr><td><i>Byte 6</i></td><td colspan="8">Y position 7..0</td></tr>
</table>

Figure 3-19. 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 2.3.4 describes the various purposes and interpretations of the W value.

On normal pads, 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 2.4.4) 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 “smart” keyboard controllers) whenever the Up and Down buttons are not pressed.

Historical notes:

Some version 3.2 and earlier TouchPads used an older, incompatible Absolute packet format. The *infoNewAbs* bit in the Model ID query response shows which packet format is in use; *infoNewAbs* is 1 for pads which use the format shown in the above figures, and 0 for pads which use the old format shown in Figure 3-20:

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

Figure 3-20. Old-style PS/2 absolute motion packet (infoNewAbs = 0)

This old packet format, which lacked the duplicate copy of the gesture and switch bits in byte 4, was susceptible to interference from certain types of low-level host software that try to interpret incoming PS/2 data as mouse-style Relative packets before they reach the high-level driver software. The new packet format contains the same bits as the old format, but rearranged to minimize the impact of meddling by low-level software. (The “Second” bit of Figure 3-20, for reporting “secondary” gestures, corresponds to a feature that has now been discontinued.)

3.7. PS/2 implementations

The next two sections 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 3.7.2 shows the source code for such an implementation.

3.7.1. The 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 2.6.6 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 3.2.3, 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 in order for the device to be recognized.
- Some KBCs will only send the PS/2 commands listed in section 3.4 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 3.5) are disguised as sequences of valid section 3.4 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.
- 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 itself thinks no buttons are pressed. In order 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 3-18) 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.

3.7.2. 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 implemention 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. */
```

Figure 3-21. PS/2 support functions

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 25ms).

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

Figure 3-22. PS/2 `wait_CLOCK` function

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. */
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 }
```

Figure 3-23. Receiving PS/2 data from the TouchPad

The `PS2_get` function reads one byte from the TouchPad, as shown in Figure 3-7 on page 30. 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 20µs 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 50µs 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:

```
void PS2_send(byte value)
{
  byte i, ack, p = 1;
  set_CLOCK(0);               /* Begin inhibit, if necessary. */
  wait_us(100);               /* Inhibit for about 100us. */
  set_DATA(0);                /* Hold data pin low while still inhibited. */
  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. */
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 }
```

Figure 3-24. Sending PS/2 data to the TouchPad

See Figure 3-8 on page 30. The `PS2_send` function first inhibits the bus for at least 100µs; 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 10ms 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; note that 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 you should 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 3.5, 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  }
```

Figure 3-25. PS/2 `send_tp_arg` *function*

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 3.3, the device will send an $AA, $00 announcement 300–1000ms 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 300ms after power-up before talking to the device. Note that the host should be prepared to wait up to 1000ms 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 3-18.

```
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 3.5.1). */
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 3.5.2). */
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. */
```

Figure 3-26. PS/2 initialization sequence

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 pad.

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 30–50µs 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.

4. Serial Protocol

The Synaptics Serial TouchPad communicates with the host via a standard RS-232 interface or “COM port.” In the default (Relative) mode, the TouchPad uses a protocol fully compatible with the Microsoft serial mouse. The Serial TouchPad offers all the same features as the PS/2 TouchPad, though the commands and packet formats are different.

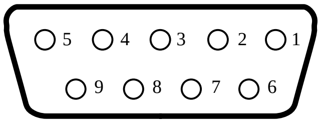
Although Synaptics has manufactured true Serial-only TouchPads in the past, all new Serial-capable pads are actually “Combo” devices which support both the Serial and the PS/2 protocols. In PS/2 mode, the Combo pad acts exactly the same as a PS/2-only pad as described in section 3 of this *Guide.* (The connector pinout is different, however; see below.)

4.1. Electrical interface

The RS-232C interface includes receive and transmit signal wires RxD and TxD plus a set of control signals. Of the control signals, the Serial TouchPad uses only RTS and DTR. RS-232 signals operate at high voltages and are logically inverted: A logic “0” corresponds to a voltage between +5V and +15V, and a logic “1” corresponds to a voltage between –5V and –15V relative to ground. (On a typical portable computer, the serial port operates at ±6V or ±12V; for concreteness, the following text will use ±12V.)

Serial pointing devices use the RS-232 control wires in an unconventional way in order to avoid the need for an external power supply. First, RTS and DTR, which are at –12V at boot time, are switched to +12V by the mouse driver. The device uses a voltage regulator to derive its +5V power supply from RTS and/or DTR. Second, RxD, which is at –12V as long as the host is not transmitting anything, is used as the negative power supply for generating negative voltages on the TxD wire. A side effect is that the device is unable to transmit reliably to the host when the host itself is transmitting. Fortunately, the host does not need to transmit to the device in normal operation.

The serial port on a portable computer typically uses a DB-9 connector as follows (female connector view):



1	DCD	*Not used*
2	TxD	Transmit to host
3	RxD	Receive from host
4	DTR	Power supply 5–15V
5	GND	Ground 0V
6	DSR	Plug-and-play signal
7	RTS	Power supply / Reset
8	CTS	*Not used*
9	RI	*Not used*

Figure 4-1. Serial connector pinout

On the Synaptics Standard Combo TouchPad module TM41B*xx*134, the 8-pin FFC connector has the following pinout:

1	2	3	4	5	6	7	8
DTR / DATA	TxD	RxD	Right Switch	Left Switch	Ground 0V	CTS / +5V	RTS / CLK

Figure 4-2. Combo module connector pinout

The button switch inputs (pins 4 and 5) include pull-ups to +5V on the module, and should be grounded when the corresponding switch is closed (pressed).

The Combo TouchPad module should be wired to a DB-9 female connector; the signals TxD, RxD, DTR, RTS, CTS, and GND should be wired between the two connectors as indicated in Figures 4-1 and 4-2. To support plug-and-play, DTR and DSR should be wired together on the connector. The DB-9 connector can be plugged directly into a computer's RS-232 port for Serial TouchPad operation. (Note that the CTS pin must be wired from pin 8 of the DB-9 connector to pin 7 of the module connector, even though the CTS signal is marked "not used" in Figure 4-1. The CTS wire will be unused when operating in Serial mode but it is needed to supply power in PS/2 mode.)

The following diagram shows the interconnections between the host Serial port and the Synaptics Combo TouchPad module:

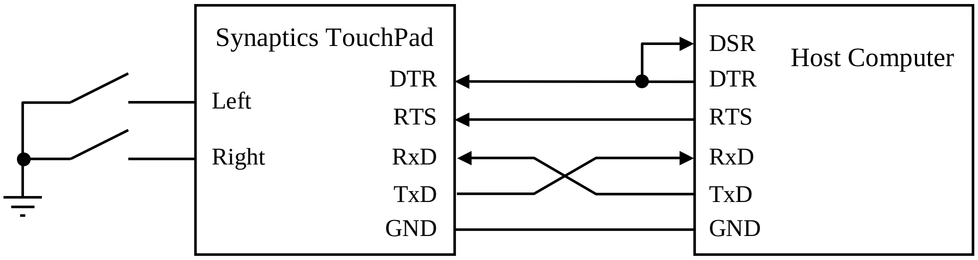


Figure 4-3. Serial system diagram

In order to use the Combo TouchPad with a PS/2 port, an adapter or converter cable is used between the TouchPad's Serial connector and the host's PS/2 port. The adapter has a male DB-9 connector on one end and a male DIN-6 connector on the other. The adapter may separate the DB-9 and DIN-6 connectors with a short length of cable, or the two connectors may be encased in a molded plastic block. The DB-9 and DIN-6 connectors are wired in the adapter as follows:

DIN-6 *(PS/2)*			DB-9 *(Serial)*	
1	PS/2 DATA	↔	4	DTR
2	N/C		—	—
3	Ground 0V	↔	5	Ground 0V
4	Power +5V	↔	8	CTS
5	PS/2 CLK	↔	7	RTS
6	N/C		—	—

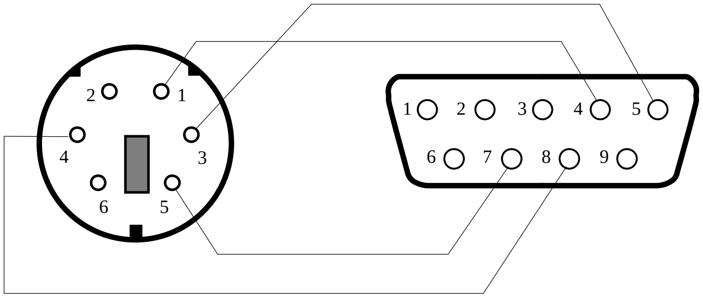


Figure 4-4. Serial-to-PS/2 adapter

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

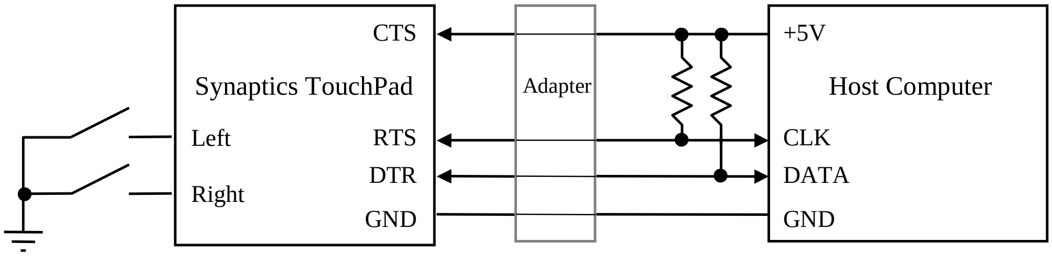


Figure 4-5. PS/2 system diagram with Combo module

Historical notes:

Some older Synaptics TouchPad models supported the Serial protocol *only*; they connected to the host RS-232 port by the same DB-9 connector shown in Figure 4-1, but the firmware programmed on the module did not recognize the PS/2 protocol; hence, there was no way to build a PS/2 adapter comparable to Figure 4-4 for those modules. The Serial-only module had the same connector pinout as shown in Figure 4-2, except that pin 7 was a no-connect instead of CTS (since CTS is wired on the Combo module only to support the PS/2 adapter).

Note that in this *Guide,* the phrase “Serial TouchPad” refers equally to a Serial-only pad or to a Combo pad operating in Serial mode. The phrase “Serial-only TouchPad” refers specifically to a non-Combo Serial pad.

4.1.1. TTL-level Serial TouchPad

Synaptics is also able to manufacture TouchPads which use the RS-232 serial protocol but with all signals operating at TTL voltage levels. TxD and RxD are non-inverted, so that logic "0" is 0V and logic "1" is +5V. Non-inverted signals are suitable for connecting directly to a UART without an intervening level-shifter chip. Either or both signals can be inverted as a firmware compile-time option. This "TTL-level Serial" pad is not a part of Synaptics' standard product line, but it is available by special arrangement.

The TTL-level Serial firmware can run on the same standard Synaptics TouchPad module used for PS/2 devices. On the standard TM41T*xx*134 module, the 8-pin FFC connector has the following pinout:

1	2	3	4	5	6	7	8
Power +5V	TxD	RxD	Right Switch	Left Switch	Ground 0V	N/C	N/C

Figure 4-6. TTL-level Serial standard module connector pinout

Similarly, on other module types, the PS/2 DATA pin becomes the RS-232 TxD pin, and the PS/2 CLK pin becomes the RS-232 RxD pin. The RxD pin should be tied to a logic "1" if it is not used.

Note that the pinout shown above does not involve RTS or DTR, and thus the TTL-level Serial TouchPad will not obey the "M" protocol that allows a standard serial mouse driver to recognize it. For full emulation of a serial mouse, you must arrange for the TouchPad to receive +5V power only when the serial port's RTS signal is a logic "0". One possible solution is shown here:

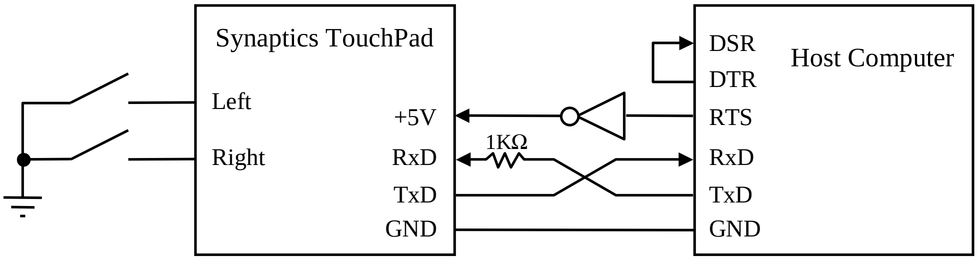


Figure 4-7. TTL-level Serial system diagram with RTS support

Since the Synaptics TouchPad draws only a few milliamps, a digital inverter easily serves as its power source. The resistor (about 1K) on the TouchPad's RxD line is to prevent excessive current draw in case the host's RTS and TxD outputs are both "1" at the same time. (If the host can guarantee not to let that happen, the resistor is unnecessary. If the host microprocessor has an output pin capable of sourcing enough milliamps to drive the TouchPad directly, it can dispense with the inverter. In fact, if the host does not care about using the RTS handshake to identify the pad as a "mouse," then the TouchPad's +5V pin can simply be connected to the regular +5V power supply.)

4.2. Byte transmission

Data transmission works the same in either direction. The signal wire (TxD or RxD) normally rests at a logic “1” level (negative voltage). To send a byte, the transmitter sends a “0” start bit, seven data bits (LSB first), and one or more “1” stop bits. (Since the wire rests at “1”, the stop bits can be thought of as an inter-byte delay measured in bit times.) The length of each bit cell is determined by the *baud rate,* in bits per second.

For the Synaptics TouchPad, the serial port is set to 1200 or 9600 baud, 7 data bits, and no parity. The default baud rate at power-up is 1200 baud. The device transmits with two stop bits and can receive with one or more stop bits (two stop bits are recommended).

In the following example, the bytes $25 and $43 (hexadecimal) are sent at 1200 baud with two stop bits. Each bit cell is 1/1200 second = 833µs.

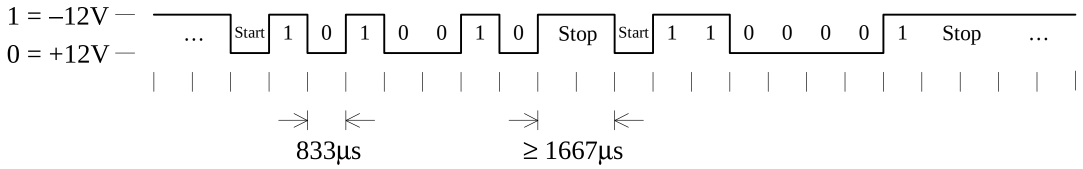


Figure 4-8. Serial TxD or RxD waveform

The receiver typically samples each bit in the middle of the bit cell, synchronized by the falling edge of the start bit. The transmitter’s and receiver’s baud rate generators must be precise enough to ensure that the receiver’s sampling point will not have drifted out of the bit cell by the time the stop bit arrives. Hence, the TouchPad’s transmit baud rate will be accurate to within ±2%, and the host should be comparably accurate. A *framing error* occurs if the receiver finds a zero when it expects a stop bit. When the TouchPad detects a framing error, it treats the received byte as invalid and cancels any command in progress.

It is also possible to send a *break* signal by holding the transmit wire at a logic “0” (a positive voltage) for at least 9 bit times. The Synaptics TouchPad treats a received break signal like an invalid command byte; the TouchPad never sends a break signal.

4.3. Power-on reset

To apply power to a Serial TouchPad, first set the DTR signal to a positive voltage and RTS to a negative voltage. Then wait at least 100ms without transmitting any bytes or break signals. Finally, bring RTS positive while holding DTR also positive. The TouchPad will send a $4D (ASCII “M”) announcement byte to identify itself as a standard serial pointing device. The announcement byte is sent at 1200 baud, and the start bit occurs within 30ms after the time at which RTS goes positive.

Note that the device may detect power-on either by drawing its power directly from RTS, or by drawing power from DTR and using RTS as a hardware or software reset signal. In any case, the host must hold DTR positive whenever it holds RTS positive, and the host must hold both DTR and RTS positive whenever it transmits command bytes to the device.

The host can reset the device at any time by pulsing RTS negative for at least 100ms. This procedure is known as the “RTS handshake.” When RTS goes positive again after the handshake, the device will send the announcement byte “M” and all TouchPad parameters will return to their power-up default states. In particular, the TouchPad mode byte (section 2.5) will reset to $00.

After the RTS handshake, the Combo TouchPad determines the type of protocol by looking at the RxD wire. In Serial mode, RxD will be held negative by the host (because the host is not transmitting to the TouchPad). In PS/2 mode, RxD will be unconnected and will float to 5V due to the design of the Combo circuit. Thus, to ensure that the device correctly identifies the protocol at power-on, it is imperative for Serial hosts to refrain from transmitting to the device until they receive the “M” announcement, and for PS/2 adapters to leave the RxD pin unconnected.

If the pad supports *Plug-and-Play*, then the “M” announcement is followed by a separator string, then by a Plug-and-Play ID string. In modern Serial TouchPads, the separator string consists of the bytes $40 $00 $00 $00. Mouse drivers unaware of Plug-and-Play will interpret this as a harmless zero-motion packet; without the separator, mouse drivers would interpret the “M” and the first two bytes of the Plug-and-Play ID as a motion packet when the device is hot-plugged.

The Plug-and-Play ID string itself is a sequence of characters whose ASCII character codes are offset by subtracting $20. (For example, the character “**(**“ is encoded as the byte $08.) The Plug-and-Play ID takes the form,

(*vv* **SYN0001\\MOUSE\PNP0F0C\TOUCHPAD** *cc* **)**

where *vv* represents two version number bytes, and *cc* represents a two-digit checksum. Note that the separator and Plug-and-Play ID string are subject to change by Synaptics. For further information, see the *Plug and Play External COM Device Specification,* available from Microsoft.

After sending the initial “M”, the device does a self-test and calibration taking 300–1000ms. However, the device begins watching RxD immediately after sending “M”. If the device receives host commands or other input during the initial calibration, the device may restart the calibration as soon as the commands are finished. Finger motion processing will not begin until the calibration is complete. See section 2.6.5 for more information about power-on calibration.

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 pad will reset itself as if after a power-on reset, and all TouchPad mode byte settings will be lost. In particular, note that a spurious reset will cause the pad to spontaneously revert from Absolute to Relative mode and from 9600 baud to 1200 baud. If the host notices the pad spontaneously reverting to 1200 baud Relative mode (usually characterized by a series of framing errors if the host UART was set to 9600 baud), then the host should reinitialize the pad in the same manner as at power-up.

Historical notes:

Some older Serial TouchPads transmitted a two-byte announcement consisting of a $4D ("M") and a $33 ("3"); this identified the pad to certain non-Synaptics drivers as a three-button mouse. Since the current TouchPad does not support or emulate a third button, the current TouchPad announces itself as an ordinary two-button mouse.

4.4. Command set

The host can send a command to the TouchPad at any time. Any motion packets or responses that were being transmitted by the TouchPad are cancelled and lost. Each command consists of a "%" character (ASCII code $25), a command letter, and optionally several argument characters. The device ignores invalid commands; in particular, between commands it ignores all characters except for "%".

4.4.1. Serial command timing

Commands, their arguments, and the TouchPad's replies are always transmitted at 1200 baud (even if data reporting is set to 9600 baud). The characters of a command with its arguments must be separated by no more than 50ms (t1); characters of a motion packet or reply will be separated by at most 25ms (t3, t5). The device responds to valid commands after no less than 25ms and no more than 50ms (t2). After processing a command and sending the response, the device waits at least 50ms before it resumes sending motion packets (t4). The time between two characters is measured from the beginning of the first stop bit to the beginning of the subsequent start bit.

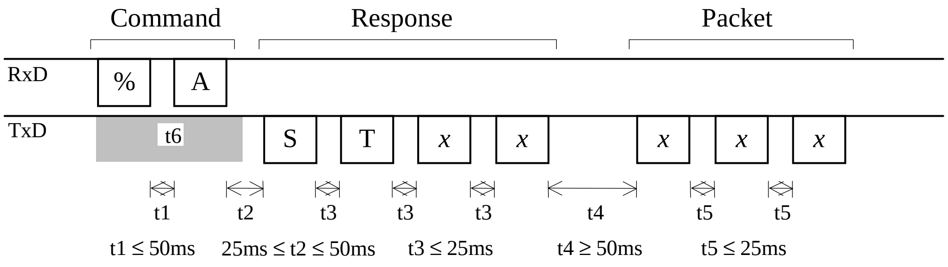


Figure 4-9. Serial command timing

Because the device uses RxD as a power supply for generating TxD, the device may send spurious data when the host transmits on RxD. Therefore, the host should ignore all input from the device while sending commands and preferably for 5–20ms thereafter (t6). (This is safe since the device does not reply for at least 25ms after a command.)

The host can set the baud rate for motion data packets to 1200 or 9600 baud. In 9600 baud mode, the host must switch its UART back to 1200 baud before sending a command. The host can take advantage of the 50ms post-command delay to reconfigure the UART cleanly. To send a command while in 9600 baud mode, the host can send an invalid byte such as NUL ($00) or a break signal; the device will ignore the byte as a command, but it will still abort any transmission in progress and wait 50ms before transmitting another packet. The host thus has 50ms of quiet time to switch the UART to 1200 baud and send the "%" character. After the command is finished, there will be another 50ms lull during which the host can set the UART back to the desired baud rate.

4.4.2. Identify TouchPad command

To identify a Synaptics TouchPad, send the command “%A”. The response will be four bytes identifying the TouchPad and reporting its version number.

	Bit 6	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Send 1	$25 “%”						
Send 2	$41 “A”						
Receive 1	$53 “S”						
Receive 2	$54 “T”						
Receive 3	infoModelCode				infoMajor		
Receive 4	infoMinor						

Figure 4-10. Serial Identify TouchPad response

Note that the *infoMajor* and *infoMinor* fields are each one bit shorter than their PS/2 counterparts (compare to Figure 3-10).

To identify an unknown serial device as a Synaptics Serial TouchPad, first perform the “RTS handshake” described in section 4.3 and check for an “M” response. Then, send the “%A” query and check for an “ST” response followed by two version bytes. If the device fails to respond to the RTS handshake, it is not a mouse-compatible pointing device. (In this case, the host should not try a “%A” query since the “%A” characters may have an unknown effect on a non-pointing serial device.) If the device responds to the RTS handshake but not to the “%A” query, then it is a non-Synaptics pointing device.

4.4.3. Read TouchPad Modes command

To read the current TouchPad modes and capabilities, send the command “%B”. The response will be eight hexadecimal digits encoding four bytes of data including the constant “3B”, the mode byte (Figure 2-14 of section 2.5), and the capability bits (Figure 2-13 of section 2.4.4). Each digit is a character from “0” to “9” ($30 to $39) or from “A” to “F” ($41 to $46).

	Bit 6	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Send 1	$25 “%”						
Send 2	$42 “B”						

Receive 1	$33 “3”						
Receive 2	$42 “B”						
Receive 3	Mode byte, bits 7–4 (hex digit)						
Receive 4	Mode byte, bits 3–0 (hex digit)						
Receive 5	Capability bits 15–12 (hex digit)						
Receive 6	Capability bits 11–8 (hex digit)						
Receive 7	Capability bits 7–4 (hex digit)						
Receive 8	Capability bits 3–0 (hex digit)						

Figure 4-11. Serial Read TouchPad Modes response

Historical notes:

On pre-4.*x* TouchPads, the first two digits reported “mode byte 1” as described in section 7.1.1, and the last four digits reported the Edge Motion margin adjustment factors instead of the capability bits. The margin adjustment factors were “5555” by default; starting at version 3.2, these bytes were no longer adjustable and were fixed at “5555”.

4.4.4. Set TouchPad Modes command

To change the TouchPad mode byte, send the command “%C” followed by eight hexadecimal digits encoding the mode byte as shown below. Each digit is a character from “0” to “9” ($30 to $39) or from “A” to “F” ($41 to $46). Only upper-case letters are accepted. If “%C” is followed by less than eight digits, or if any of the digits are invalid, the entire command is rejected and the mode byte is not changed.

	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Send 1	$25 “%”						
Send 2	$43 “C”						
Send 3	$33 “3”						
Send 4	$42 “B”						
Send 5	Mode byte, bits 7..4 (hex digit)						
Send 6	Mode byte, bits 3..0 (hex digit)						
Send 7	$35 “5”						
Send 8	$35 “5”						
Send 9	$35 “5”						
Send 10	$35 “5”						

Figure 4-12. Serial Set TouchPad Modes command

See Figure 2-16 of section 2.5 for a list of suitable values for the mode byte on a Serial TouchPad.

The first two, and last four, argument digits are ignored by current TouchPads. For compatibility with older and future Synaptics TouchPads, host software may either use the values shown in Figure 4-12 (“3B” and “5555”, respectively), or echo the same values reported by a recent “%B” command in the corresponding digit positions.

Historical notes:

On pre-4.*x* TouchPads, the first two argument digits set “mode byte 1” as described in section 2.5, and the last four digits set the Edge Motion margin adjustment factors.

4.4.5. Read Model ID command

To read the model ID as described in Figure 2-7 of section 2.4.2, send the command “%D”. The response will be six hexadecimal digits encoding the three model ID bytes. Each digit is a character from “0” to “9” ($30 to $39) or from “A” to “F” ($41 to $46).

	Bit 6	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Send 1	$25 “%”						
Send 2	$44 “D”						
Receive 1	Model ID, bits 23–20 (hex digit)						
Receive 2	Model ID, bits 19–16 (hex digit)						
Receive 3	Model ID, bits 15–12 (hex digit)						
Receive 4	Model ID, bits 11–8 (hex digit)						
Receive 5	Model ID, bits 7–4 (hex digit)						
Receive 6	Model ID, bits 3–0 (hex digit)						

Figure 4-13. Serial Read Model ID response

Historical notes:

In Synaptics TouchPads prior to version 3.2, the model ID bytes were not implemented; the “%D” command had no effect and produced no response from the TouchPad. To determine whether a TouchPad supports the model ID query, send a “%D” and then wait to see if there is a reply within 50ms consisting of six valid hexadecimal digits. (Alternatively, simply check the version number and omit the model ID query for pads older than 3.2.)

4.4.6. Read Serial Number command

To read the serial number as described in section 2.4.5, send the commands “%G” and “%H”. The response to each command will be six hexadecimal digits encoding three bytes of data.

	Bit 6	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Send 1	$25 “%”						
Send 2	$44 “G”						
Receive 1	Serial Number, bits 31–28 (hex digit)						
Receive 2	Serial Number, bits 27–24 (hex digit)						
Receive 3	Serial Number, bits 35–32 (hex digit)						
Receive 4	*Reserved* (hex digit)						
Receive 5	*Reserved* (hex digit)						
Receive 6	*Reserved* (hex digit)						

Send 1	$25 “%”
Send 2	$44 “H”

Receive 1	Serial Number, bits 23–20 (hex digit)
Receive 2	Serial Number, bits 19–16 (hex digit)
Receive 3	Serial Number, bits 15–12 (hex digit)
Receive 4	Serial Number, bits 11–8 (hex digit)
Receive 5	Serial Number, bits 7–4 (hex digit)
Receive 6	Serial Number, bits 3–0 (hex digit)

Figure 4-14. Serial Read Serial Number responses

If the “%G” command returns “000” for bits 35–24 of the serial number, or if “%G” returns no response, then the device is unserialized and the result, if any, of the “%H” command is undefined. The values of the final three response digits of the “%G” command are undefined in any case.

Historical notes:

In Synaptics TouchPads prior to version 4.*x*, the serial number was not implemented; the “%G” and “%H” commands had no effect and produced no response from the TouchPad. All 4.*x* and later TouchPads support the “%G” query, though as of this writing Synaptics had not begun serializing TouchPads and so the “%G” response was always “000” in the first three digits.

4.4.7. Read Resolutions command

To read the coordinate resolutions as described in section 2.4.3, send the command “%I”. The response will be six hexadecimal digits encoding the X and Y resolutions in Absolute units per millimeter, plus a middle byte with undefined data.

	Bit 6	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Send 1	$25 “%”						
Send 2	$44 “I”						

	Bit 6	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Receive 1	*infoXupmm*, bits 7–4 (hex digit)						
Receive 2	*infoXupmm*, bits 3–0 (hex digit)						
Receive 3	*Reserved* (hex digit)						
Receive 4	*Reserved* (hex digit)						
Receive 5	*infoYupmm*, bits 7–4 (hex digit)						
Receive 6	*infoYupmm*, bits 3–0 (hex digit)						

Figure 4-15. Serial Read Resolutions response

Historical notes:

In Synaptics TouchPads prior to version 4.5, the resolution bytes were not implemented; the “%I” command had no effect and produced no response from the TouchPad. To determine whether a TouchPad supports the resolution query, send a “%I” and then wait to see if there is a reply within 50ms consisting of six valid hexadecimal digits. (Alternatively, simply check the version number and omit the resolution query for pads older than 4.5.)

4.5. Data reporting

The Synaptics TouchPad supports two formats for motion data packets. The default Relative format is compatible with Microsoft and Logitech serial mice. The Absolute format gives additional information that may be of use to TouchPad-cognizant applications.

4.5.1. Default packet format

In the default Relative format, each motion report consists of three bytes. Reports are sent whenever finger motion and/or button state changes occur.

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

Figure 4-16. Serial relative motion packet

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

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

X delta: The two fields in bytes 1 and 2 combine to form an eight-bit signed, two’s-complement integer ΔX representing the amount of horizontal motion since the last data packet. Rightward motion is positive, leftward is negative.

Y delta: The two fields in bytes 1 and 3 combine to form an eight-bit signed, two’s-complement integer ΔY representing the amount of vertical motion since the last data packet. Downward motion is positive, upward is negative.

The X and Y deltas have a resolution of about 240 DPI on a standard Synaptics pad; see section 2.6.3 for further details.

Historical notes:

Some older TouchPads were capable of simulating a “middle” mouse button using advanced gestures. The Relative packet would grow to four bytes to report middle button activity. Because Synaptics TouchPads no longer support advanced gestures in Relative mode, the Relative packet is now exclusively a three-byte packet.

4.5.2. Absolute packet format

When Absolute mode is enabled, each motion report consists of six, seven, or eight bytes. These bytes encode the absolute X, Y location of the finger on the sensor pad, as well as the Z (pressure) value and various other measurements and status bits. Section 2.3 discusses the contents of the Absolute mode packet in great detail.

The Absolute packet size is controlled by the *PackSize* and *Wmode* bits of the TouchPad mode byte (section 2.5).

PackSize	*Wmode*	*Packet size*
0	0	Six bytes (bytes 7 & 8 omitted)
0	1	*Reserved*
1	0	Seven bytes (byte 8 omitted)
1	1	Eight bytes

Figure 4-17. Serial absolute packet sizes

The full eight-byte Absolute packet is arranged as follows. Note that with *Wmode* = 0, the W value is not present, and with *PackSize* = 0, the least significant bits of X and Y, the least significant two bits of Z, and the Down and Up bits are also not present.

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

Figure 4-18. Serial absolute motion packet

The “Middle” bit reports the state of the physical middle mouse button; since all current Synaptics pads support just two buttons, this bit is always zero.

The “Down” and “Up” bits report the states of the extra two buttons on “MultiSwitch” pads with the *capFourButtons* capability bit set. On pads for which *capFourButtons* is clear, these bits are reserved.

Bits marked *reserved* may be reported as 0 or 1 by the TouchPad, and should be ignored by the host.

The host should always set the pad to 9600 baud mode (the *Baud* bit of the mode byte should be 1) when operating the pad in Absolute mode. (At 1200 baud, the pad would transmit only 20 packets per second and the reported data might not be very accurate.)

5. ADB Protocol

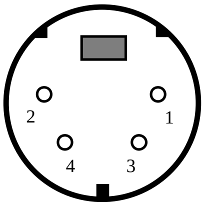
The Apple Desktop Bus protocol is used by Apple Macintosh pointing devices and other peripherals. The ADB bus allows for bidirectional communication between a host computer and up to 16 peripheral devices on a single shared bus wire.

The Synaptics ADB TouchPad identifies itself as a mouse device on the bus, and in its default mode is fully compatible with Apple mice. The ADB TouchPad offers the same set of gestures and advanced features as other Synaptics TouchPads.

5.1. Electrical interface

The ADB bus includes one signal wire in addition to +5V power and ground. The signal wire is a bidirectional “open-collector” signal, normally held at a high (+5V) level by a 470 Ω pull-up resistor on the host. Either the host, the TouchPad device, or any other device on the bus can pull the wire low at any time. However, in ADB, all transactions are initiated by the host.

An external ADB port uses a mini-DIN-4 connector with the following pinout (male connector view):



1	ADB Signal
2	Power-on
3	Power +5V
4	Ground 0V

Figure 5-1. ADB cable pinout

The “power-on” pin is not used by pointing devices, and should be left unconnected.

The Synaptics ADB firmware can run on the same standard Synaptics TouchPad module used for PS/2 devices. On the standard TM41A*xx*134 module, the 8-pin FFC connector has the following pinout:

1	2	3	4	5	6	7	8
Power +5V	N/C	ADB Signal	N/C	Button Switch	Ground 0V	N/C	N/C

Figure 5-2. ADB standard module connector pinout

Similarly, on other module types, the PS/2 CLK pin becomes the ADB Signal pin, the Left Switch pin becomes the sole button switch pin, and the PS/2 DATA and Right Switch pins are not connected by the host.

The button switch input includes a pull-up to +5V on the module, and should be grounded when the button switch is closed (pressed).

The following diagram shows the interconnections between the host and the Synaptics ADB TouchPad:

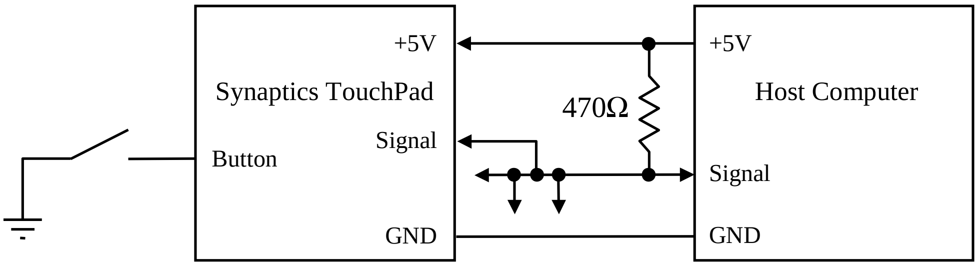


Figure 5-3. ADB system diagram

5.2. Byte transmission

The ADB protocol is amply described in various Apple publications, such as the *Guide to the Macintosh Family Hardware*, second edition (Addison Wesley, 1990).

However, please note the following errors and ambiguities in the *Guide to the Macintosh Family Hardware* ("*GMFH*"):

- The stop bit is a "0" bit for both commands and data; in figure 8-13 of the *GMFH*, the data stop bit is erroneously shown as a "1". The low time is 70µs for both stop bits. While the *GMFH* refers to the stop bit as a normal bit with a low portion and an (invisible) high portion, it is most useful to think of the stop bit as consisting solely of the 70µs low portion.
- In a Service Request, the low portion of the command byte's stop bit extends to be 300µs ± 30% long. Service Requests occur only on the command stop bit, not on the data stop bit. A device never asserts a Service Request on a command addressed to that device.
- The stop-to-start time following the command is measured from the rising edge in the stop bit, not from the (invisible) end of the stop bit's high portion as shown in figure 8-13 of the *GMFH*. In the case of a Service Request, the stop-to-start time is measured from the observed rising edge at the end of the Service Request.
- To aid collision detection, the stop-to-start time on a "Talk 3" command is randomized with a random number generator. This is not necessary for other "Talk" commands. Also, the device reports a random number in the "address" field of the "Talk 3" response, rather than its true address.
- In a "Listen" command, the effect of sending fewer argument bytes than the device expects is undefined. Extraneous "Listen" argument bytes are ignored. The effect of sending a "reserved" command byte is undefined.

The Synaptics ADB TouchPad follows all of these conventions.

5.3. Power-on reset

The Synaptics TouchPad is able to respond to ADB bus traffic within 200ms of power-up. The device's ADB port is fully functional after this point for as long as power is applied.

At power-up, the TouchPad mode byte is set to $00. The initial ADB address is 3, and the initial handler ID is $01. Power-up settings are restored whenever a Global Reset signal or SendReset command occurs on the ADB bus.

5.4. Command set

The Synaptics TouchPad emulates a standard ADB mouse. It supports the Cursor Device Manager interface as well as the older 100- and 200-dpi interfaces. It also supports extensions that allow the host to access the Synaptics TouchPad's special features.

All ADB devices have four logical registers up to eight bytes in length. The Synaptics TouchPad supports the usual ADB commands for accessing these registers:

- The "Talk" command reports the current contents of any of the four ADB registers.
- The "Listen" command stores a new value into one of the four registers.
- The "Flush" command clears any pending motion packet, but has no other effect on the state of the device.
- The "SendReset" command and the "Global Reset" signal reset the device to its power-up state.

The Synaptics TouchPad treats invalid command codes the same as "Flush" commands.

In the Synaptics TouchPad, the four ADB registers are assigned as follows:

ADB Register	*Length*	*Contents*
0	0–5	Current finger motion packet
1	8	CDM identification
2	8	Synaptics TouchPad mode bytes
3	2	ADB identification

Figure 5-4. ADB Registers

These registers are described in the sections below.

5.4.1. ADB Register 0

This register contains the current finger motion or position data. It has the special property that it is empty ("Talk 0" does not respond) if there is no motion to report. The host must poll Register 0 periodically to collect the motion data. The device asserts a "Service Request" whenever Register 0 becomes full but the host is talking to a different ADB device. The device continues to assert Service Requests until the host sends a

"Talk 0" to read the new motion data. After a "Talk 0", Register 0 becomes empty until further motion occurs.

The Synaptics TouchPad supports three operating modes, each with its own format for Register 0:

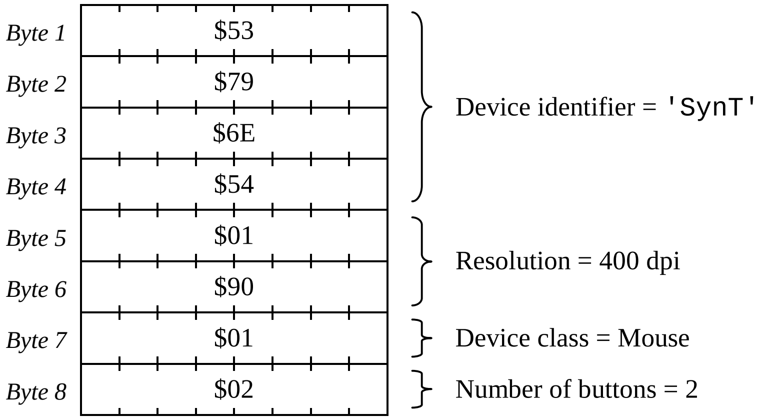
- **Non-CDM Relative mode.** This is the default mode at power-up; it emulates an original 100- or 200-dpi ADB mouse.
- **CDM Relative mode.** This mode was introduced by Apple to allow for uniform handling of a wider range of pointing devices.
- **Absolute mode.** In this mode, the Synaptics TouchPad reports the finger's absolute position and pressure on the pad.

See sections 5.5.1–5.5.3 below for further details about Register 0.

5.4.2. ADB Register 1

Register 1 contains eight bytes of read-only identification data as described in the CDM specification. (CDM is described in the Apple technical report, *ADB—The Untold Story: Space Aliens Ate My Mouse*, January 1994.)

In the Synaptics TouchPad, Register 1 contains the following data:



Byte 1	$53	Device identifier = `'SynT'`
Byte 2	$79	
Byte 3	$6E	
Byte 4	$54	
Byte 5	$01	Resolution = 400 dpi
Byte 6	$90	
Byte 7	$01	Device class = Mouse
Byte 8	$02	Number of buttons = 2

Figure 5-5. ADB Register 1: CDM identification

While Register 1 is present in all modes, it is mainly applicable to CDM Relative mode. For example, the resolution is really 400 dpi only in CDM Relative mode.

The number of buttons is reported as 2 to reflect the fact that the physical switch and gestures are reported as two distinct logical buttons. Unless told otherwise, the CDM driver software will merge these two buttons into a single button signal as seen by application programs.

The recommended way to identify a Synaptics ADB TouchPad is to search for a device with the following properties:

1. The device's original ADB address was 3 (though collision detection may since have moved the device to a different address);

2. The handler ID reported by a “Talk 3” command is $01, $02, or $04; and
3. The device responds to a “Talk 1” command with eight bytes, where the first four bytes are “SynT” as shown in Figure 5-5.

Host software should check each of these items before assuming Register 2 has the format shown below.

5.4.3. ADB Register 2

This register holds data specific to the Synaptics TouchPad. It contains the TouchPad version number, the mode byte, and the capability bits.

	Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Command	*Address*				$E = “Talk 2”			
Byte 1	infoModelCode				infoMajor			
Byte 2	infoMinor							
Byte 3	$3B							
Byte 4	Absolute	—	—	—	—	DisGest	—	Wmode
Byte 5	Capability bits 15..8							
Byte 6	Capability bits 7..0							
Byte 7	Model ID bits 23..16							
Byte 8	Model ID bits 7..0							

Figure 5-6. ADB Register 2: Talk response

Bytes 1 and 2 report the TouchPad version information; see section 2.4.1.

Byte 3 always reports the constant $3B.

Byte 4 is the mode byte, as described in section 2.5. Note that the *Rate* bit is not supported since the ADB bus always operates in a polled mode with no fixed report rate.

Bytes 5 and 6 are the extended capability bits, as described in section 2.4.4.

Bytes 7 and 8 are a subset of the model ID bits described in section 2.4.2. *Historical note:* In TouchPads prior to version 4.*x*, bytes 7 and 8 report the model ID only if the *Wmode* bit of byte 4 is 1. If the *Wmode* bit is 0, these bytes return undefined data. Note that the *Wmode* bit may be used to obtain model information even on older pads where “W mode” and the “W” value themselves are not supported; in those cases, the *Wmode* bit will have no effect on the register 0 packet format. On very old pads which do not support model ID information, setting *Wmode* will have no effect on Register 2, and bytes 7 and 8 will report as $00, $00.

To change the current TouchPad modes, execute a “Listen 2” command with the eight-byte argument shown below:

	Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Command	*Address*				$A = "Listen 2"			
Byte 1	$FF							
Byte 2	$FF							
Byte 3	$3B							
Byte 4	Absolute	—	—	—	—	DisGest	—	Wmode
Byte 5	$55							
Byte 6	$55							
Byte 7	$00							
Byte 8	$00							

Figure 5-7. Setting ADB mode byte

Bits shown as "—" in the diagram should be set to zero for compatibility with future Synaptics TouchPad models.

The TouchPad will accept the Listen 2 command only if the first two arguments bytes are $FF and the last two argument bytes are $00. Note that a Listen command that simply echoes the version numbers in the first two bytes will be ignored; this helps to minimize the risk of unexpected interaction with software that writes to Register 2 thinking it is configuring a different type of pointing device.

The third, fifth, and sixth Listen argument bytes may either be the constants $3B, $55, $55 as shown in Figure 5-7, or they can echo the values most recently read in the corresponding byte positions by a Talk 2 command. Hence, the recommended way to modify ADB Register 2 is as follows:

1. Execute a Talk 2 command to read the 8-byte contents of Register 2 into a buffer.
2. Modify bytes 1 and 2 to $FF, $FF.
3. Modify bytes 7 and 8 to $00, $00.
4. Modify byte 4 to the desired mode byte value.
5. Execute a Listen 2 command to write the modified buffer back into Register 2.

To read the model ID information, it is necessary to perform this transaction once to set the *Wmode* bit, then to perform a second Talk 2 command to read the model ID data in bytes 7 and 8.

Historical notes:

In older TouchPads, bytes 3–6 of Register 2 reported up to four mode bytes and margin adjustment parameters. See section 7.1.1 for more information.

5.4.4. ADB Register 3

The layout and meaning of Register 3 is fixed for all ADB devices.

	Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Byte 1	0	1	Enable	0	Device address			
Byte 2	Device handler ID							

Figure 5-8. ADB Register 3: ADB identification

The Enable bit enables Service Requests from the device; it is 1 (enabled) by default. The default device address is 3. The default handler ID is $01 (100 dpi mouse), and can be changed to $02 (200 dpi mouse) or $04 (CDM device) by the host. The special handler ID codes $00 and $FD–$FF are also supported; for code $FD, the “activator” is the physical button switch; for code $FF, the self-test command has no effect.

See the ADB specification for further details.

5.5. Data reporting

The TouchPad reports information about finger motion and button state changes in response to a “Talk 0” command on the ADB bus. The format of the “Talk 0” response depends on the current TouchPad operating mode.

5.5.1. Default packet format

When the “handler ID” of ADB Register 3 is $01 or $02, the TouchPad responds with a two-byte motion packet fully compatible with older Apple mouse products.

	Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Byte 1	Button	Y delta						
Byte 2	1	X delta						

Figure 5-9. Non-CDM ADB motion packet

Button:	0 = Button is currently pressed, 1 = released. This bit combines physical switch and gesture-based button information.
Y delta:	This is the amount of motion ΔY that has occurred in the Y (vertical) direction since the last motion data packet. This field is a 7-bit signed, two’s-complement integer. Downward motion is positive, upward is negative.
X delta:	This is the amount of motion ΔX that has occurred in the X (horizontal) direction. Rightward motion is positive, leftward is negative.

If the handler ID is $01, the ΔX and ΔY resolution is approximately 100 units per inch. If the handler ID is $02, the ΔX and ΔY resolution is approximately 200 units per inch.

5.5.2. CDM Relative mode packet format

When the “handler ID” of ADB Register 3 is $04, the TouchPad responds with between two and five bytes of motion data compatible with Apple’s CDM specification.

	Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Byte 1	Switch	Y delta 6..0						
Byte 2	Gesture	X delta 6..0						
Byte 3	1	Y delta 9..7			1	X delta 9..7		
Byte 4	1	Y delta 12..10			1	X delta 12..10		
Byte 5	1	Y delta 15..13			1	X delta 15..13		

Figure 5-10. CDM ADB motion packet

Switch: 0 = Physical button switch is currently pressed, 1 = released.

Gesture: 0 = Gesture is in progress, 1 = no gesture. This bit represents the state of the “virtual” mouse button controlled by tap-and-drag gestures.

Y delta: This is the amount of motion ΔY that has occurred in the Y (vertical) direction since the last motion data packet. This field is a 7-, 10-, 13-, or 16-bit signed, two’s-complement integer. Downward motion is positive, upward is negative.

X delta: This is the amount of motion ΔX that has occurred in the X (horizontal) direction. Rightward motion is positive, leftward is negative.

The ΔX and ΔY resolution, in units per inch, is available from ADB Register 1. In current devices, the resolution is 400 units per inch.

CDM devices can choose to report anywhere from two to five bytes in each “Talk 0” response. The most significant reported bit (bit 6, 9, 12, or 15) becomes the sign bit of a two’s-complement integer of the corresponding size. The device may omit one or more of the final bytes if both deltas are small enough to be represented correctly in the remaining bytes. The Synaptics TouchPad uses the shortest packet (two bytes) whenever possible.

5.5.3. Absolute packet format

When Absolute mode is enabled, the ADB TouchPad always reports five bytes in Register 0 regardless of the current handler ID. These bytes encode the absolute X, Y location of the finger on the sensor pad, as well as the Z (pressure) value and various other measurements and status bits. Section 2.3 discusses the contents of the Absolute mode packet in great detail.

	Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Byte 1	Switch	Y position 8..2						
Byte 2	Gesture	X position 8..2						
Byte 3	1	Y position 11..9			1	X position 11..9		
Byte 4	1	Y position 14..12			1	X position 14..12		
Byte 5	1	Z pressure 7..5			1	Z pressure 4..2		

Figure 5-11. ADB Absolute motion packet

Note that the low two bits of each of X, Y, and Z are missing from the ADB Absolute packet format, as is the Finger bit.

The Switch and Gesture bits are actually controlled by the handler ID even in Absolute mode: If the handler ID is $04 (CDM mode), then Switch and Gesture are reported as distinct bits as shown. If the handler ID is $01 or $02, the bit shown as Switch reports a combination of switch and gesture information, and the bit shown as Gesture always reports as 1.

The X and Y resolution in Absolute mode is the same as described in section 2.4.3, even though Register 1 will continue to report a resolution of 400 DPI. (The information reported in Register 1 applies only to CDM Relative mode, not to Absolute mode.)

The ADB TouchPad will report Absolute mode packets continuously when the finger is down (i.e., when Z exceeds a suitable internal threshold value). When the finger is up, the TouchPad reports a single packet with X = Y = Z = 0, then reports no further packets except for button state changes.

Note: “W” mode is not yet implemented on the ADB version of the Synaptics TouchPad. Contact Synaptics for more information about this mode.

6. Driver API

Sections 3 and 4 of this *Guide* have described the PS/2 and Serial protocols that most IBM PC-compatible computers use 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 2-19 of section 2.6.6, the Synaptics Windows 95 and NT 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.

This section of the *Synaptics TouchPad Interfacing Guide* will present an introduction to the TouchPad API. For complete documentation, download the API developer's kit from the Synaptics Web site, http://www.synaptics.com.

6.1. API basics

The TouchPad API is a set of C++ classes that can be used in any C++ program. (The API will soon support access from other languages; contact Synaptics for availability.)

The *CTouchPad* object represents a connection to a particular TouchPad device. A computer may have more than one Synaptics TouchPad attached (say, an internal PS/2 TouchPad plus an external Serial TouchPad). By creating several *CTouchPad* objects, an application can receive separate and independent position data from each pad. In fact, an application can create a *CTouchPad* object for any mouse-compatible pointing device under control of the Synaptics driver, though if the device is not a Synaptics TouchPad, the *CTouchPad* object will only deliver relative motion and button information.

The *CTouchPadPacket* object represents the state of the TouchPad at a particular instant in time. The *CTouchPadPacket* constructor takes a pointer to a *CTouchPad* object as an argument; it fills in the *CTouchPadPacket* object with data reflecting the current state of the indicated TouchPad. By repeatedly constructing *CTouchPadPacket* objects, an application can sample X, Y, Z, and other TouchPad data as the finger moves around on the pad.

More typically, an application would establish a *feed* from the TouchPad API that causes the API to notify the application whenever anything “interesting” happens. For example, the application is notified whenever a new packet of finger data arrives from the pad, and whenever a TouchPad is plugged in, unplugged, or reconfigured using the Control Panel.

The driver API supports several kinds of feeds:

1. In a normal feed, the driver sends a Windows message (with the message number of your choice) every time a packet arrives from the pad. The application’s main event loop can handle these messages just like any other Windows message. Typically, it would respond by constructing a *CTouchPadPacket* object and acting on the packet data in some appropriate way. If the application falls behind processing the messages, Windows will queue the messages up and play them out when the application is ready. Each message includes a packet sequence number, which the *CTouchPadPacket* constructor uses to find which recent packet to retrieve.
2. In an *interlocked* feed, the driver sends Windows messages to your application one at a time. Your application must acknowledge the receipt of one message before the driver will send the next. This scheme ensures that the Windows message queue will not overflow with TouchPad messages. The application can use the packet sequence numbers to read all the packets that have come in since the last TouchPad message. Or, the application can read the single latest packet in response to each message; this will ensure that the application can not drop behind in packet processing, but at the expense of possibly dropping packets when the system is busy.
3. It is also possible to dispense with Windows messages altogether and have the API call a function in your program directly each time a packet arrives. The API will create a special Win32 thread in your application for handling the TouchPad; your function will be called in the context of that thread, and it will be called immediately even if your application’s main thread is busy.
4. A *notification* feed sends you a Windows message whenever the configuration of the TouchPad changes in a significant way, for example, when the user changes a TouchPad-related setting on the Control Panel.

A feed can either “spy” on packets on their way to their regular processing by Windows, or it can take over the TouchPad and prevent Windows from moving the cursor in response to finger actions. For example, the “virtual scroll bar” feature of the Synaptics driver spies on the finger at all times; if it sees the finger come down in the scroll zone of the pad, it takes over the TouchPad for the duration of the scrolling gesture.

The API also supports *reverse feeds*, where the application tells the driver to simulate a mouse packet as if it had arrived from the TouchPad. For example, the virtual scroll bars feature sometimes uses reverse feeds to move the cursor onto the scroll bar, press the simulated button, and drag the scroll box up and down.

6.2. Information available from the API

Here are some of the many functions that the *CTouchPad* object offers to describe the type, capabilities, or current configuration of the TouchPad.

IsTouchPad	Is this device a Synaptics TouchPad?
GetFWRev	The TouchPad/firmware version numbers (section 2.4.1).
GetDriverVersion	The driver version number.
GetPort	The protocol and attachment port (e.g., COM1).
GetSensorType	The *infoSensor* code (section 2.4.2).
GetGeometry	The *infoGeometry* code (section 2.4.2).
GetCapabilities	The extended capability bits (section 2.4.4).
GetXLoSensor	The absolute coordinate limits (section 2.3.2).
GetXLoRim	The bezel coordinate limits (section 2.3.2).
GetXLoBorder	The user's current edge margins (section 2.3.2).
GetXLoWideBorder	Like edge margins, but with an extra-wide edge zone.
GetXDPI	The resolution, in absolute units per inch.
IsMultiFingerCapable	Is this TouchPad able to sense multiple fingers?
IsPenCapable	Is this TouchPad able to sense pens as well as fingers?
IsTapEnabled	Are taps enabled in control panel?
IsDragEnabled	Are drag gestures enabled in control panel? (etc…)

Similarly, the *CTouchPadPacket* object has a number of functions that describe the state of the TouchPad.

IsValid	Does this object contain valid data from a packet?
GetSeq	The packet's sequence number.
GetTime	The time at which the packet arrived.
GetX, GetY	The current finger position.
GetZ	The current pressure.
GetW	The current finger width.
GetCurrentFingers	The number of fingers on the pad.
GetDX, GetDY	The amount of finger motion.
GetDXB, GetDYB	Finger motion with ballistics (acceleration) applied.
GetXRaw, …	Data from TouchPad with no driver filtering.
IsLeftSw	Is the physical left button switch pressed?
IsPrimSw	Is the "primary" button switch pressed?
IsProx	Is the finger on or near the TouchPad surface?
IsTouch	Is the finger touching the TouchPad surface?
IsFinger	Is the finger "present" according to the driver's algorithms?
IsStylus	Is this a pen stroke instead of a finger stroke?
IsTap	Is a tap gesture in progress?
IsDrag	Is a drag gesture in progress?
IsMoving	Is the finger moving at a significant speed?

See the API documentation for a complete listing with full descriptions of the various functions.

6.3. Sample program

Here is a sample piece of C++ code that uses the TouchPad API. This is not a complete program, just the part of a larger program that uses the API to access the TouchPad. The program first enables a feed on the TouchPad, and then calls the MyProcessPacket() function with the X, Y, and Z coordinates from each packet.

```
#include "TouchPad.h"

CTouchPad* pTouchPad;

void MyInitTouchPad()
{
  pTouchPad = new CTouchPad;    // Create the CTouchPad object.
  if (!pTouchPad->IsTouchPad()) {
    MessageBox("No Synaptics TouchPad detected. Exiting.\n");
    PostQuitMessage(0);
    delete pTouchPad;
    return;
  }
  pTouchPad->SetupFeed(hMyWnd, iMyMessage);    // Set up messaging.
  pTouchPad->StartFeed();    // Start the flow of TouchPad packets.
}

// If using MFC and the MS Development environment, set up a Windows
// message handler:
BEGIN_MESSAGE_MAP(MyClass, CWnd)
  ON_MESSAGE(iMyMessage, OnSynTpFeed)
END_MESSAGE_MAP()

// Otherwise, in a non-MFC windows program, your window procedure will
// be called as follows:
//  LRESULT CALLBACK MyWndProc(hMyWnd, iMyMessage, wParam, lParam)
// and will then probably call OnSynTpFeed(wParam, lParam) or some such handler.

LRESULT OnSynTpFeed(UINT wParam, LONG lParam)
{
  ASSERT(lParam);

  // Create a CTouchPadPacket object with data for this packet.
  // wParam holds a packet sequence number used for validation and lParam contains
  // a pointer to the CTouchPad object that sent the message.
  CTouchPadPacket pkt((CTouchPad*) lParam, wParam);

  if (pkt.IsValid()) {
    // It is important to check the IsFinger bit, because x and y are not valid
    // unless a finger is on the pad (and thus IsFinger is TRUE).
    MyProcessPacket(pkt.GetX(), pkt.GetY(), pkt.GetZ(), pkt.IsFinger());
  }
}
```

7. Appendices

7.1. Historical TouchPad features

This appendix describes some features of older Synaptics TouchPads which have been discontinued in modern (version 4.*x*) pads. The present appendix may be interesting if you are writing software that is sure to be used *only* with older TouchPads.

7.1.1. Old-style mode bytes

TouchPads prior to version 3.2 supported a total of four mode bytes. Starting with version 3.2, the last two mode bytes were discontinued, leaving just two host-settable mode bytes. Certain later 3.*x* pads, and all 4.*x* pads, further reduced the configurability to a single mode byte. When the *infoSimpleCmd* bit (section 2.4.2) is 1, the TouchPad pad supports just one mode byte as described in section 2.5. When *infoSimpleCmd* is 0, the TouchPad supports additional mode bytes as described below.

In the PS/2 protocol, query $01 read the first two mode bytes and query $02 read the second two mode bytes. In modern pads, query $02 reads the extended capability bits. To set a mode byte, the host would transmit four Set Resolution commands followed by a Set Sample Rate with an argument of 10, 20, 40, or 60 (decimal) to set mode byte 1, 2, 3, or 4, respectively. In modern pads, mode byte 2 is the only mode byte; attempts to set the other mode bytes are ignored.

In the Serial and ADB protocols, the host could read or set all four mode bytes in a single command. The commands documented in sections 4.4 and 5.4.3 to read and set the modern mode byte generalize to access the four mode bytes in the obvious way.

The original four TouchPad mode bytes were arranged as follows:

	Bit 7	*Bit 6*	*Bit 5*	*Bit 4*	*Bit 3*	*Bit 2*	*Bit 1*	*Bit 0*
Byte 1	Corner	Z-threshold			Tap Mode		Edge Motion	
Byte 2	Absolute	Rate	—	—	Baud	3-Button	Middle	Hop
Byte 3	Right edge margin				Left edge margin			
Byte 4	Top edge margin				Bottom edge margin			

Figure 7-12. PS/2 TouchPad mode bytes (obsolete)

The four mode bytes defaulted to $3B, $00, $55, and $55, respectively, at power-up.

The *Corner* bit enabled the corner-tap feature; a tap or drag gesture initiated in the upper-right corner of the pad simulated a right-button click instead of a left-button click. In modern TouchPads, tap and drag gestures always simulate a left click in Relative mode.

The *Z-threshold* bits allowed the host to adjust the Z threshold used for detecting the presence of the finger. Values from 1 to 7 varied from “light” to “heavy” touch; 0 selected an especially light touch. In modern TouchPads, the Z threshold is fixed at approximately 30 units.

The *Tap Mode* bits were 00 to disable all tap gestures, 01 to enable taps but not drags, 10 to enable taps and drags, and 11 for taps and "locking" drags. With locking drags, a drag gesture would continue to hold the virtual button down until the user executed a second tap to end the locking drag. Modern TouchPads have a single *DisGest* bit in a different location; *DisGest* = 0 corresponds to *Tap Mode* = 10, and *DisGest* = 1 corresponds to *Tap Mode* = 00.

The *Edge Motion* bits were 00 to disable Edge Motion, 01 for Edge Motion at all times, or 11 for Edge Motion only during drag gestures. In modern TouchPads this selection is effectively stuck at 11.

The *3-Button* bit enabled a mode in which tap gestures were reported as left clicks, and the left and right physical switches reported as middle and right mouse buttons, respectively.

The *Middle* bit caused corner-tap or hop gestures to simulate middle button clicks instead of right button clicks.

The *Hop* bit enabled a gesture which caused taps to simulate right button clicks when the user tapped far to the left or right of the previous finger location. This feature was never very successful, and it was discontinued in version 3.2 of the TouchPad.

The *edge margin* bit fields allowed the host to control the positions of the edge margins, or equivalently, the sizes of the edge zones which triggered the Edge Motion feature. For each edge, a value 0 produced a narrow margin, and a value of 15 produced a wide margin. Starting with version 3.2 of the TouchPad, adjustable margins were discontinued.

7.1.2. Fast PS/2 mode byte access

TouchPads with the *infoSimpleCmd* bit set to 0 supported a set of non-standard PS/2 command codes that provided easier access to the four mode bytes. These commands used byte codes $E0 through $E3. To avoid confusion with other PS/2 devices, commands $E0 through $E3 were recognized only if the special Identify TouchPad sequence (four Set Resolution 0 commands and a Status command) had been sent to the device since power-up.

Each special command would "access" one mode byte. If the most recent Set Scaling command was Set Scaling 1:1 ($E6), the command would read a mode byte. The response was an ACK ($FA), followed by a data byte. If the most recent Set Scaling command was Set Scaling 2:1 ($E7), the command was followed by one argument byte; it would set a mode byte according to the argument.

Commands $E0, $E1, $E2, and $E3 accessed mode bytes 1, 2, 3, and 4, respectively. Commands $E2 and $E3 were discontinued starting with version 3.2.

While the $E0–$E3 commands were faster and simpler than the command sequences described in section 3.5.2, they were of limited utility since the standard PC BIOS does not allow non-standard command codes to be sent to a PS/2 device. Thus, the $E0–$E3 commands were discontinued altogether in later devices (including all 4.*x* devices).

7.2. 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 the *Guide* where a word or concept is discussed.

$	In this *Guide*, the dollar sign signifies hexadecimal (base-16) notation: $7FF = 0x7FF = 7FFh = 2047 decimal.
—	In bit-field diagrams, see *reserved.*
Absolute Mode	A mode in which the TouchPad reports the *absolute position* of the finger in each *packet*. (§2.3)
Absolute Position	The position of the finger on the pad surface measured absolutely with respect to a coordinate system with the point (0,0) in the lower-left corner. See also *relative motion.* (§2.3.2, Figure 2-4.)
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. (§2.3.2)
Acceleration	Pointing devices typically offer a feature called "acceleration" or "ballistics" which increases the speed factor at higher speeds in order to help the user move the cursor a long distance with a small motion. (§2.6.3)
ACK	A response byte with value $FA used to acknowledge each host command or argument byte in the PS/2 protocol. (§3.2.3)
ACPI	The Advanced Configuration and Power Management Interface, a standard promoted by Intel, Microsoft, and Toshiba.
ADB	Apple Desktop Bus™. The interface used by all but the earliest Apple Macintosh® computers to connect to low-speed peripherals like mice and keyboards. (§5)
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 (§3.3). In *Serial* devices, the announcement is $4D ("M") possibly followed by a *plug-and-play* ID string (§4.3).
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. (§6)
Application Software	Software that interacts directly with the user, generally that was

	deliberately run by the user to accomplish some task. Application software generally interacts with the TouchPad via the TouchPad *API.* (§6)
ASIC	Application specific integrated circuit. A chip specially designed for a particular task. TouchPads include the T1002 or T1004 capacitive sensing ASIC designed by Synaptics.
Bezel	A physical covering that surrounds the TouchPad sensor. 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 a recommended bezel shape and position for each TouchPad model. 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 Edge Motion™ 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 2-3 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 pad is used with small fingers. (§2.3.2)
Button	See *virtual button* and *physical button*.
Calibration	A process taking about one-half of a second in which the TouchPad prepares its capacitive sensors for operation. (§2.6.5)
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…*”. (§2.4.4)
Capacitance	The electrical phenomenon which most TouchPads use to sense the presence of fingers. (§2.6.1)
Click	Clicking a *button* involves pressing (activating) the button for a short time, then releasing the button, generally not involving cursor motion while the button is pressed.
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. (§3.1)
Combo	Short for “combination,” in pointing devices this refers to a device that can use either the *PS/2* or the *Serial* protocol depending on which type of port it is connected to. (§4.1)

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. In *PS/2*, the host has a male connector and the pointing device is female. In *RS-232*, the host is female and the pointing device is male. (§3.1, §4.1, §5.1)
Coordinates	A pair of numbers identifying an *absolute position* on the surface of the TouchPad. See *X coordinate* and *Y coordinate.* (§2.3.2)
CTS	The “Clear To Send” pin in the RS-232 protocol. The Synaptics “Combo” TouchPad actually uses CTS for an entirely different purpose, as a PS/2 power supply pin. (§4.1)
Cursor	A symbol displayed on a computer screen which can be moved around the screen using a *pointing device*. The cursor is often shaped like an arrow. The user can operate a control or select a piece of text by moving the cursor onto the control or text and clicking the “mouse” button. (§2.6.3)
DATA	The PS/2 signal wire that carries data bits between the *device* and the *host*. (§3.1)
DB-9	The nine-pin connector used for *RS-232* ports. Some ports actually use a DB-25 connector, whose twenty-five pins contain all the signals on a DB-9 port plus many others which are not relevant to TouchPads. (§4.1, Figure 4-1)
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. (§2.6.3)
Device	In the discussion of TouchPad *protocols*, “device” refers to the TouchPad or other pointing device, as distinct from the *host*.
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. (§3.1, Figure 3-1)
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 2-10(*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. (§2.4.3)
Drag	The act of pressing a *mouse button* and holding the button down while moving the pointing device.
Drag Gesture	A *gesture* involving a *tap* on the pad followed immediately by the return of the finger to the pad surface. Viewed another way, the gesture feels like a double-tap in which the second tap is extended into a full motion *stroke;* this leads to the nickname *tap-and-a-half* for the drag gesture. This gesture activates the *virtual button* for as long as the finger remains on the pad. The virtual button is released when the finger lifts from the pad at the end of the motion stroke. (§2.6.4, Figure 2-17)
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. (§2.6.6)
DSR	The "Data Set Ready" pin in the RS-232 protocol. Serial pointing devices actually use DSR for an entirely different purpose, as a *plug-and-play* identification pin. (§4.1)
DTR	The "Data Terminal Ready" pin in the RS-232 protocol. Serial pointing devices actually use DTR for an entirely different purpose, as a power supply pin. (§4.1)
Edge Margins	The coordinate limits that identify the dividing lines beween the *edge zone* and the interior of the pad. (§2.3.2)
Edge Motion™	A feature which assists with long-distance cursor motions, especially during *drag gestures.* If the finger moves into the *edge zone* of the pad, the pad begins to generate continuous *relative motion* in the direction corresponding to the edge. (§2.6.4)
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 pad, and the edge zone is the part of the pad surface not in the interior.) Moving the finger into the edge zone triggers *Edge Motion™*. (§2.3.2, §2.6.4)
Electrical Noise	See *noise.*
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 play havoc with sensitive electronic devices such as TouchPads. To minimize the risk of ESD disruption, designers should carefully follow Synaptics' recommendations on grounding and *bezel* design.
Feed	An active connection to the TouchPad API which delivers information to the application about finger and button actions on the TouchPad. (§6.1)
Female Connector	See *connector.*
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. (§2.3.2)
Finger	In most Synaptics TouchPads, the "finger" must be a grounded, conductive object roughly the size of a human fingertip. Certain TouchPad models can also sense *pens;* except where pens are explicitly treated differently, the word "finger" in this *Guide* refers to whichever object is activating the TouchPad sensor, whether a pen or a true finger. (§2.6.1, §2.3.4)
Firmware	The software that operates the microcontroller built-in to the TouchPad module. See also *version number.* (§2.6)
Framing Error	The PS/2 and RS-232 protocols both send 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. (§3.2.2, §4.2)
Gesture	A finger action which the host interprets as a special command instead of as a simple cursor motion. See *tap, drag,* and *scrolling gesture.*
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. (§2.6.6)
Host Software	Any software running on the host that interacts with the TouchPad; for example, *drivers, applications,* and the operating system. (§2.6.6)
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 a Serial device generally works. However, please note that hot-plugging is *not* recommended for PS/2, Serial, or ADB 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. (§4.3)
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. (§3.2)
KBC	Keyboard Controller. The part of the *host*, usually a subsidiary microprocessor, which manages the host side of the *PS/2* interface. (§2.6.6, §3.7.1)
Lifting	Lifting the finger means taking the finger far enough away from the surface of the pad so that the pad no longer registers the finger's presence (i.e., until the *Z value* falls below the *touch threshold*). (§2.6.1)
Line Control Bit	The final bit of a PS/2 transmission from *host* to *device*. (§3.2.2)
Male Connector	See *connector.*
Margin	See *edge margins.*
Mickey	One unit of mouse motion as reported by the pointing device to the host. (§2.6.3)
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. (§2.5)
Model ID	A 24-bit response to a certain *query* which describes the size and shape of the TouchPad. (§2.4.2)
Model Number	An alphanumeric code, such as "TM41PUA134", which describes a kind of TouchPad *module* as ordered from Synaptics. (§2.7)
Module	The standard 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. The system integrator typically installs the module in a *bezel* and adds two *button switches* to form a complete working TouchPad. (§2.7)
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*.
MultiSwitch	A type of Synaptics TouchPad which supports four *physical buttons* instead of just the usual two. (§2.4.4, §3.6.2)
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 (§2.2); filtering is another (§2.3.2).
Open-Collector	Also “open-drain.” 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 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. (§3.1)
Packet	One transmission from the pointing device to the host describing the user’s actions. The device sends many packets per second in order to form the illusion of continuous cursor motion. (§2.1, §2.3)
Packet Rate	The rate at which the TouchPad potentially sends packets to the host. In PS/2 Remote mode and in ADB, the packet rate is actually the rate at which new packets become available for polling. Note that in Relative mode, the packet rate is merely the maximum possible number of packets transmitted per second, but 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.* (§2.2)
Palm Check™	A feature of Synaptics drivers which tries to use *palm detection* and other clues to suppress the effects of accidental TouchPad activation.
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. (§2.3.4, §2.4.4)
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 ‘1’s 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 but not all multi-bit errors. (§3.2)
Pen	Any object except a *finger* which is used to point on a TouchPad. Only TouchPad models with the *capPen* capability are able to

	sense pens. The pen may be an actual (non-inking) writing stylus, or it may be a fingernail or any other rigid, non-conductive object that is convenient for pointing. (§2.3.4)
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.” On an Apple Macintosh, the pointing device typically has just one button. The phrase “physical button” refers to an actual button, not a *virtual button* activated by *tap gestures*. (§2.3.1)
Plug and Play	A standard which allows PC-compatible computers to identify the hardware connected to them. PS/2 pointing devices do not need any special attention to comply with Plug-and-Play; Serial pointing devices need a special connection on the DSR pin, plus a special Plug-and-Play ID string. (§4.1, §4.3)
Pointing Device	A *mouse,* track ball, *touch pad,* or other device which is used to move the *cursor* on the computer screen.
Portrait	A portrait-oriented pad is one which is rotated 90° from the standard orientation, so that the vertical axis is longer than the horizontal. The standard orientation, where the horizontal axis is longer, is sometimes called “landscape.” (Figure 2-11(*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. (§2.3.3, §2.6.1)
Protocol	A defined method for communicating between a *device* and the *host.* See *PS/2, RS-232,* and *ADB.*
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. (§3)
PS/2 Sample Rate	A parameter of the PS/2 mouse protocol; not fully implemented on Synaptics TouchPads. See *packet rate.* (§3.4)
PS/2 Resolution	A parameter of the PS/2 mouse protocol; not fully implemented on Synaptics TouchPads. See *resolution.* (§3.4)
Query	A command from the host which is a request for information about the properties or current state of the TouchPad. (§2.4)
Relative Mode	A mode in which the TouchPad reports the *relative motion* of the finger in each *packet*. (§2.1)

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). (§2.6.3)
Request To Send	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. See also *RTS.* (§3.2)
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 in any way; if the host writes to a reserved bit, it should write the default value shown in this *Guide* (or zero if not shown), or alternatively it can read the reserved bit and then write back the same value.
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 (§3.3); in Serial, the RTS wire (§4.3); in ADB, the Global Reset signal or SendReset command (§5.3).
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.* (§2.4.3)
Response	One or more bytes sent by the *device* to the *host* in response to a host *command*.
RS-232	The protocol used by serial "COM" ports for communicating with modems, printers, and other devices. *Serial* pointing devices use the RS-232 port and protocol in an unconventional way to send pointing information to the host. (§4)
RTS	The "Request To Send" pin in the RS-232 protocol. Serial pointing devices actually use RTS for an entirely different purpose, as a power supply and software reset pin. (§4.3)
RTS Handshake	The process of bringing DTR positive, then RTS first negative and then positive, to identify a pointing device on a serial port. (§4.3)
RxD	The "Receive Data" pin in the RS-232 protocol. The RxD pin on the TouchPad connects to TxD on the host's serial connector. Note that, in this document, RxD is named from the TouchPad's point of view: It carries data received from the host. (§4.1)
Scrolling Gesture	A *gesture* which causes the current window to scroll. The Synaptics TouchPad does not decode scrolling gestures itself, but the Synaptics driver software does offer a variety of scrolling gestures.

Sensor	The top surface of the TouchPad module, which is able to sense the presence and position of a finger.
Serial Device	A "Serial" pointing device interfaces to the host using the *RS-232* protocol. Named because the RS-232 protocol sends bits serially, one at a time, over a wire. (Note that, even though the *PS/2* protocol is also a bit-serial protocol, PS/2 pointing devices are never called "Serial" pointing devices.) Not to be confused with *serial numbers.* (§4)
Serial Number	A unique number assigned to each individual TouchPad. A TouchPad with an assigned serial number is said to be *serialized.* Not to be confused with *Serial devices.* (§2.4.5)
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). (§2.5)
Stop Bit	One or more bits of a known value at the end of each transmitted byte. The PS/2 and RS-232 protocols use the stop bit to detect *framing errors;* in RS-232, the receiver can also use guaranteed transition between a stop bit and a subsequent byte's start bit to resynchronize itself to the transmitter. (§3.2.2, §4.2)
Stroke	A finger stroke is one complete finger action involving placing the finger on the pad, possibly moving the finger around for some amount of time, then lifting the finger away from the pad. (§2.6.3)
Stylus	See *pen.*
Switch	See *physical button.*
Tap Gesture	A *gesture* involving touching the pad briefly and then immediately lifting the finger, with little or no finger motion while the finger is on the pad. The TouchPad decodes the tap gesture to simulate a brief *click* of the *virtual button.* See also *drag gesture.* (§2.6.4)
Tap-and-a-half	See *drag gesture.*
TBD	"To be determined." Contact Synaptics for further information.
Touch Pad	A pointing device which translates motions of a finger on a sensor surface into cursor motions.
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. (§2.6.1)
Touch Threshold	A value to which the *Z value* is compared to decide whether or not there is a finger on the pad. (§2.6.3, §7.1.1)

TTL	An older technology for digital circuits which operated with a 5V power supply. While TTL has largely been supplanted by CMOS, its 5V power supply standard has survived. Nowadays, digital signals operating at 5V are often called “TTL-level signals.” (§4.1.1)
TxD	The “Transmit Data” pin in the RS-232 protocol. The TxD pin on the TouchPad connects to RxD on the host’s serial connector. Note that, in this document, TxD is named from the TouchPad’s point of view: It carries data transmitted to the host. (§4.1)
UART	(Also **USART** or **SCI**.) Universal Asynchronous Receiver / Transmitter. A chip or circuit which implements the RS-232 byte-level protocol. On a modern PC, this is typically a 16550 device or equivalent. (§4.2)
Undefined	Same as *reserved.*
Up Orientation	The standard “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 2-10(*a*))
USB	Universal Serial Bus. A protocol for connecting low- and medium-speed devices to personal computers. (§2.6.6)
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 “4.5”, “4” is the *major version* and “5” is the *minor version.* (§2.4.1)
Virtual Button	A simulated mouse button which is activated by *tap* and *drag gestures* instead of by a physical switch. The system typically treats the physical and virtual buttons the same so that tap gestures can serve as a convenient alternative to button clicks. (§2.6.4)
Virtual Scrolling™	See *scrolling gesture.*
Windows®	Any of the Microsoft Windows® family of operating systems, such as Windows® 95 or Windows NT®.
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, the number of fingers, and whether the contact is by a *pen* or a true *finger.* (§2.3.4)
X Coordinate	The *coordinate* identifying the horizontal position of the finger; low values are near the left edge of the pad, and high values are near the right edge. (§2.3.2)

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 pad (closest to the user), and high values are near the top edge. (§2.3.2)
ΔY Value	The *delta* identifying the amount of vertical finger motion.
Z Value	A numeric value reported by the TouchPad which measures the *pressure* of the finger contact. (§2.3.3)

7.3. References: Synaptics literature

Synaptics TouchPad Single-Chip Standard Module TM41P-134: Product Specification,
Synaptics TouchPad Single-Chip Ultra-Thin Module TM41P-220: Product Specification,
Synaptics TouchPad Single-Chip Mini-Module TM41P-134: Product Specification,
Synaptics TouchPad Single-Chip SubMini-Module TM41P-140: Product Specification,
Synaptics, Inc., 1997.
The mechanical and electrical specifications for various TouchPad modules.

Synaptics TouchPad Driver API, Synaptics, Inc., 1995.
Information on the programming interfaces provided by the TouchPad driver software.

Synaptics Web page: http://www.synaptics.com
A variety of on-line literature, specifications, software, and corporate information.

On-line help for Synaptics Windows® 95 & NT drivers.
The driver help file has user-friendly descriptions of many TouchPad features.

(Contact Synaptics for TouchPad Application Notes, etc.)

7.4. References: Other literature

The Undocumented PC, Frank van Gilluwe, Addison-Wesley, 1994.
Discusses the BIOS mouse functions (chapter 13) and keyboard controller (chapter 8).

Plug and Play External COM Device Specification, Microsoft Inc., 1995.
Documents the Plug-and-Play specification, esp. for Serial pointing devices.

Guide to the Macintosh™ Family Hardware, 2nd ed., Apple Inc., Addison-Wesley, 1990.
Chapter 8 is an excellent description of the ADB protocol.

ADB—The Untold Story: Space Aliens Ate My Mouse, Apple technical report, 1994.
Further notes and examples on ADB; coverage of new CDM standard.

Inside Macintosh: Devices, Apple Inc., Addison-Wesley, 1994.
Chapter 5 describes the Macintosh system functions for operating the ADB port.