NAME

psm - PS/2 mouse style pointing device driver

SYNOPSIS

options KBD_RESETDELAY=N
options KBD_MAXWAIT=N
options PSM_DEBUG=N
options KBDIO_DEBUG=N
device psm

In */boot/device.hints*:
hint.psm.0.at="atkbdc"
hint.psm.0.irq="12"

DESCRIPTION

The **psm** driver provides support for the PS/2 mouse style pointing device. Currently there can be only one **psm** device node in the system. As the PS/2 mouse port is located at the auxiliary port of the keyboard controller, the keyboard controller driver, **atkbdc**, must also be configured in the kernel. Note that there is currently no provision of changing the *irq* number.

Basic PS/2 style pointing device has two or three buttons. Some devices may have a roller or a wheel and/or additional buttons.

Device Resolution

The PS/2 style pointing device usually has several grades of resolution, that is, sensitivity of movement. They are typically 25, 50, 100 and 200 pulse per inch. Some devices may have finer resolution. The current resolution can be changed at runtime. The **psm** driver allows the user to initially set the resolution via the driver flag (see *DRIVER CONFIGURATION*) or change it later via the ioctl(2) command MOUSE_SETMODE (see *IOCTLS*).

Report Rate

Frequency, or report rate, at which the device sends movement and button state reports to the host system is also configurable. The PS/2 style pointing device typically supports 10, 20, 40, 60, 80, 100 and 200 reports per second. 60 or 100 appears to be the default value for many devices. Note that when there is no movement and no button has changed its state, the device will not send anything to the host system. The report rate can be changed via an ioctl call.

Operation Levels

The **psm** driver has three levels of operation. The current operation level can be set via an ioctl call.

At the level zero the basic support is provided; the device driver will report horizontal and vertical movement of the attached device and state of up to three buttons. The movement and status are encoded in a series of fixed-length data packets (see *Data Packet Format*). This is the default level of operation and the driver is initially at this level when opened by the user program.

The operation level one, the ‘extended’ level, supports a roller (or wheel), if any, and up to 11 buttons. The movement of the roller is reported as movement along the Z axis. 8 byte data packets are sent to the user program at this level.

At the operation level two, data from the pointing device is passed to the user program as is. Conversely, command from the user program is passed to the pointing device as is and the user program is responsible for status validation and error recovery. Modern PS/2 type pointing devices often use proprietary data format. Therefore, the user program is expected to have intimate knowledge about the format from a particular device when operating the driver at this level. This level is called ‘native’ level.

Data Packet Format

Data packets read from the **psm** driver are formatted differently at each operation level.

A data packet from the PS/2 mouse style pointing device is three bytes long at the operation level zero:

Byte 1

- bit 7 One indicates overflow in the vertical movement count.
- bit 6 One indicates overflow in the horizontal movement count.
- bit 5 Set if the vertical movement count is negative.
- bit 4 Set if the horizontal movement count is negative.
- bit 3 Always one.
- bit 2 Middle button status; set if pressed. For devices without the middle button, this bit is always zero.
- bit 1 Right button status; set if pressed.
- bit 0 Left button status; set if pressed.

Byte 2 Horizontal movement count in two’s complement; -256 through 255. Note that the sign bit is in the first byte.

Byte 3 Vertical movement count in two’s complement; -256 through 255. Note that the sign bit is in the first byte.

At the level one, a data packet is encoded in the standard format MOUSE_PROTO_SYSMOUSE as defined in mouse(4).

At the level two, native level, there is no standard on the size and format of the data packet.

Acceleration

The **psm** driver can somewhat 'accelerate' the movement of the pointing device. The faster you move the device, the further the pointer travels on the screen. The driver has an internal variable which governs the effect of the acceleration. Its value can be modified via the driver flag or via an ioctl call.

DRIVER CONFIGURATION

Kernel Configuration Options

There are following kernel configuration options to control the **psm** driver. They may be set in the kernel configuration file (see config(8)).

KBD_RESETDELAY=X, KBD_MAXWAIT=Y

The **psm** driver will attempt to reset the pointing device during the boot process. It sometimes takes a long while before the device will respond after reset. These options control how long the driver should wait before it eventually gives up waiting. The driver will wait *X* * *Y* msecs at most. If the driver seems unable to detect your pointing device, you may want to increase these values. The default values are 200 msec for *X* and 5 for *Y*.

PSM_DEBUG=N, KBDIO_DEBUG=N

Sets the debug level to *N*. The default debug level is zero. See *DIAGNOSTICS* for debug logging.

Driver Flags

The **psm** driver accepts the following driver flags. Set them in */boot/device.hints* (see *EXAMPLES* below).

bit 0..3 RESOLUTION

This flag specifies the resolution of the pointing device. It must be zero through four. The greater the value is, the finer resolution the device will select. Actual resolution selected by this field varies according to the model of the device. Typical resolutions are:

1 (low)	25 pulse per inch (ppi)
2 (medium low)	50 ppi
3 (medium high)	100 ppi
4 (high)	200 ppi

Leaving this flag zero will selects the default resolution for the device (whatever it is).

bit 4..7 ACCELERATION

This flag controls the amount of acceleration effect. The smaller the value of this flag is, more sensitive the movement becomes. The minimum value allowed, thus the value for the most

sensitive setting, is one. Setting this flag to zero will completely disables the acceleration effect.

bit 8 NOCHECKSYNC

The **psm** driver tries to detect the first byte of the data packet by checking the bit pattern of that byte. Although this method should work with most PS/2 pointing devices, it may interfere with some devices which are not so compatible with known devices. If you think your pointing device is not functioning as expected, and the kernel frequently prints the following message to the console,

psmintr: out of sync (xxxx != yyyy).

set this flag to disable synchronization check and see if it helps.

bit 9 NOIDPROBE

The **psm** driver will not try to identify the model of the pointing device and will not carry out model-specific initialization. The device should always act like a standard PS/2 mouse without such initialization. Extra features, such as wheels and additional buttons, will not be recognized by the **psm** driver.

bit 10 NORESET

When this flag is set, the **psm** driver will not reset the pointing device when initializing the device. If the FreeBSD kernel is started after another OS has run, the pointing device will inherit settings from the previous OS. However, because there is no way for the **psm** driver to know the settings, the device and the driver may not work correctly. The flag should never be necessary under normal circumstances.

bit 11 FORCETAP

Some pad devices report as if the fourth button is pressed when the user ‘taps’ the surface of the device (see *CAVEATS*). This flag will make the **psm** driver assume that the device behaves this way. Without the flag, the driver will assume this behavior for ALPS GlidePoint models only.

bit 12 IGNOREPORTERROR

This flag makes **psm** driver ignore certain error conditions when probing the PS/2 mouse port. It should never be necessary under normal circumstances.

bit 13 HOOKRESUME

The built-in PS/2 pointing device of some laptop computers is somehow not operable immediately after the system ‘resumes’ from the power saving mode, though it will eventually

become available. There are reports that stimulating the device by performing I/O will help waking up the device quickly. This flag will enable a piece of code in the **psm** driver to hook the ‘resume’ event and exercise some harmless I/O operations on the device.

bit 14 INITAFTERSUSPEND

This flag adds more drastic action for the above problem. It will cause the **psm** driver to reset and re-initialize the pointing device after the ‘resume’ event.

LOADER TUNABLES

Extended support for Synaptics touchpads can be enabled by setting *hw.psm.synaptics_support* to *1* at boot-time. This will enable **psm** to handle packets from guest devices (sticks) and extra buttons. Similarly, extended support for IBM/Lenovo TrackPoint and Elantech touchpads can be enabled by setting *hw.psm.trackpoint_support* or *hw.psm.elantech_support*, respectively, to *1* at boot-time.

Tap and drag gestures can be disabled by setting *hw.psm.tap_enabled* to *0* at boot-time. Currently, this is supported on Synaptics touchpads regardless of Extended support state and on Elantech touchpads with Extended support enabled. The behaviour may be changed after boot by setting the sysctl with the same name and by restarting moused(8) using */etc/rc.d/moused*.

Active multiplexing support can be disabled by setting *hw.psm.mux_disabled* to *1* at boot-time. This will prevent **psm** from enabling active multiplexing mode needed for some Synaptics touchpads.

IOCTLS

There are a few ioctl(2) commands for mouse drivers. These commands and related structures and constants are defined in <*sys/mouse.h*>. General description of the commands is given in mouse(4). This section explains the features specific to the **psm** driver.

MOUSE_GETLEVEL *int *level*
MOUSE_SETLEVEL *int *level*

These commands manipulate the operation level of the **psm** driver.

MOUSE_GETHWINFO *mousehw_t *hw*

Returns the hardware information of the attached device in the following structure.

```
typedef struct mousehw {
    int buttons;    /* number of buttons */
    int iftype;     /* I/F type */
    int type;       /* mouse/track ball/pad... */
    int model;      /* I/F dependent model ID */
    int hwid;       /* I/F dependent hardware ID */
```

```
} mousehw_t;
```

The buttons field holds the number of buttons on the device. The **psm** driver currently can detect the 3 button mouse from Logitech and report accordingly. The 3 button mouse from the other manufacturer may or may not be reported correctly. However, it will not affect the operation of the driver.

The iftype is always MOUSE_IF_PS2.

The type tells the device type: MOUSE_MOUSE, MOUSE_TRACKBALL, MOUSE_STICK, MOUSE_PAD, or MOUSE_UNKNOWN. The user should not heavily rely on this field, as the driver may not always, in fact it is very rarely able to, identify the device type.

The model is always MOUSE_MODEL_GENERIC at the operation level 0. It may be MOUSE_MODEL_GENERIC or one of MOUSE_MODEL_XXX constants at higher operation levels. Again the **psm** driver may or may not set an appropriate value in this field.

The hwid is the ID value returned by the device. Known IDs include:

0 Mouse (Microsoft, Logitech and many other manufacturers)
2 Microsoft Ballpoint mouse
3 Microsoft IntelliMouse

MOUSE_SYN_GETHWINFO *synapticshw_t *synhw*

Retrieves extra information associated with Synaptics Touchpad. Only available when a supported device has been detected.

```
typedef struct synapticshw {
   int infoMajor;     /* major hardware revision */
   int infoMinor;     /* minor hardware revision */
   int infoRot180;    /* touchpad is rotated */
   int infoPortrait;  /* touchpad is a portrait */
   int infoSensor;    /* sensor model */
   int infoHardware;          /* hardware model */
   int infoNewAbs; /* supports the newabs format */
   int capPen;                /* can detect a pen */
   int infoSimplC;  /* supports simple commands */
   int infoGeometry;          /* touchpad dimensions */
   int capExtended; /* supports extended packets */
   int capSleep;      /* can be suspended/resumed */
```

```
    int capFourButtons;         /* has four buttons */
    int capMultiFinger;         /* can detect multiple fingers */
    int capPalmDetect;          /* can detect a palm */
    int capPassthrough;         /* can passthrough guest packets */
    int capMiddle;    /* has a physical middle button */
    int nExtendedButtons; /* has N additional buttons */
    int nExtendedQueries; /* supports N extended queries */
} synapticshw_t;
```

See the *Synaptics TouchPad Interfacing Guide* for more information about the fields in this structure.

MOUSE_GETMODE *mousemode_t *mode*

The command gets the current operation parameters of the mouse driver.

```
typedef struct mousemode {
    int protocol;    /* MOUSE_PROTO_XXX */
    int rate;        /* report rate (per sec), -1 if unknown */
    int resolution;  /* MOUSE_RES_XXX, -1 if unknown */
    int accelfactor; /* acceleration factor */
    int level;       /* driver operation level */
    int packetsize;  /* the length of the data packet */
    unsigned char syncmask[2]; /* sync. bits */
} mousemode_t;
```

The protocol is MOUSE_PROTO_PS2 at the operation level zero and two. MOUSE_PROTO_SYSMOUSE at the operation level one.

The rate is the status report rate (reports/sec) at which the device will send movement report to the host computer. Typical supported values are 10, 20, 40, 60, 80, 100 and 200. Some mice may accept other arbitrary values too.

The resolution of the pointing device must be one of MOUSE_RES_XXX constants or a positive value. The greater the value is, the finer resolution the mouse will select. Actual resolution selected by the MOUSE_RES_XXX constant varies according to the model of mouse. Typical resolutions are:

MOUSE_RES_LOW	25 ppi
MOUSE_RES_MEDIUMLOW	50 ppi
MOUSE_RES_MEDIUMHIGH	100 ppi

MOUSE_RES_HIGH 200 ppi

The accelfactor field holds a value to control acceleration feature (see *Acceleration*). It must be zero or greater. If it is zero, acceleration is disabled.

The packetsize field specifies the length of the data packet. It depends on the operation level and the model of the pointing device.

level 0 3 bytes
level 1 8 bytes
level 2 Depends on the model of the device

The array syncmask holds a bit mask and pattern to detect the first byte of the data packet. syncmask[0] is the bit mask to be ANDed with a byte. If the result is equal to syncmask[1], the byte is likely to be the first byte of the data packet. Note that this detection method is not 100% reliable, thus, should be taken only as an advisory measure.

MOUSE_SETMODE *mousemode_t *mode*

The command changes the current operation parameters of the mouse driver as specified in *mode*. Only rate, resolution, level and accelfactor may be modifiable. Setting values in the other field does not generate error and has no effect.

If you do not want to change the current setting of a field, put -1 there. You may also put zero in resolution and rate, and the default value for the fields will be selected.

MOUSE_READDATA *mousedata_t *data*

MOUSE_READSTATE *mousedata_t *state*

These commands are not currently supported by the **psm** driver.

MOUSE_GETSTATUS *mousestatus_t *status*

The command returns the current state of buttons and movement counts as described in mouse(4).

FILES

/dev/psm0 ‘non-blocking’ device node
/dev/bpsm0 ‘blocking’ device node

EXAMPLES

In order to install the **psm** driver, you need to add

```
device atkbdc
device psm
```

to your kernel configuration file, and put the following lines to */boot/device.hints*.

```
hint.atkbdc.0.at="isa"
hint.atkbdc.0.port="0x060"
hint.psm.0.at="atkbdc"
hint.psm.0.irq="12"
```

If you add the following statement to */boot/device.hints*,

```
hint.psm.0.flags="0x2000"
```

you will add the optional code to stimulate the pointing device after the ‘resume’ event.

```
hint.psm.0.flags="0x24"
```

The above line will set the device resolution high (4) and the acceleration factor to 2.

DIAGNOSTICS

At debug level 0, little information is logged except for the following line during boot process:

```
psm0: device ID X
```

where *X* the device ID code returned by the found pointing device. See MOUSE_GETINFO for known IDs.

At debug level 1 more information will be logged while the driver probes the auxiliary port (mouse port). Messages are logged with the LOG_KERN facility at the LOG_DEBUG level (see syslogd(8)).

```
psm0: current command byte:xxxx
kbdio: TEST_AUX_PORT status:0000
kbdio: RESET_AUX return code:00fa
kbdio: RESET_AUX status:00aa
kbdio: RESET_AUX ID:0000
[...]
psm: status 00 02 64
psm0 irq 12 on isa
psm0: model AAAA, device ID X, N buttons
```

psm0: config:00000www, flags:0000uuuu, packet size:M
psm0: syncmask:xx, syncbits:yy

The first line shows the command byte value of the keyboard controller just before the auxiliary port is probed. It usually is 40, 45, 47 or 65, depending on how the motherboard BIOS initialized the keyboard controller upon power-up.

The second line shows the result of the keyboard controller's test on the auxiliary port interface, with zero indicating no error; note that some controllers report no error even if the port does not exist in the system, however.

The third through fifth lines show the reset status of the pointing device. The functioning device should return the sequence of FA AA <ID>. The ID code is described above.

The seventh line shows the current hardware settings. These bytes are formatted as follows:

Byte 1
- bit 7 Reserved.
- bit 6 0 - stream mode, 1 - remote mode. In the stream mode, the pointing device sends the device status whenever its state changes. In the remote mode, the host computer must request the status to be sent. The **psm** driver puts the device in the stream mode.
- bit 5 Set if the pointing device is currently enabled. Otherwise zero.
- bit 4 0 - 1:1 scaling, 1 - 2:1 scaling. 1:1 scaling is the default.
- bit 3 Reserved.
- bit 2 Left button status; set if pressed.
- bit 1 Middle button status; set if pressed.
- bit 0 Right button status; set if pressed.

Byte 2
- bit 7 Reserved.
- bit 6..0 Resolution code: zero through three. Actual resolution for the resolution code varies from one device to another.

Byte 3 The status report rate (reports/sec) at which the device will send movement report to the host computer.

Note that the pointing device will not be enabled until the **psm** driver is opened by the user program.

The rest of the lines show the device ID code, the number of detected buttons and internal variables.

At debug level 2, much more detailed information is logged.

SEE ALSO

ioctl(2), syslog(3), atkbdc(4), mouse(4), sysmouse(4), moused(8), syslogd(8)

Synaptics TouchPad Interfacing Guide, http://www.synaptics.com/.

AUTHORS

The **psm** driver is based on the work done by quite a number of people, including Eric Forsberg, Sandi Donno, Rick Macklem, Andrew Herbert, Charles Hannum, Shoji Yuen and Kazutaka Yokota to name the few.

This manual page was written by Kazutaka Yokota <*yokota@FreeBSD.org*>.

CAVEATS

Many pad devices behave as if the first (left) button were pressed if the user ‘taps’ the surface of the pad. In contrast, some pad products, e.g. some versions of ALPS GlidePoint and Interlink VersaPad, treat the tapping action as fourth button events.

It is reported that ALPS GlidePoint, Synaptics Touchpad, IBM/Lenovo TrackPoint, and Interlink VersaPad require *INITAFTERSUSPEND* flag in order to recover from suspended state. This flag is automatically set when one of these devices is detected by the **psm** driver.

Some PS/2 mouse models from MouseSystems require to be put in the high resolution mode to work properly. Use the driver flag to set resolution.

There is not a guaranteed way to re-synchronize with the first byte of the packet once we are out of synchronization with the data stream. However, if you are using the *XFree86* server and experiencing the problem, you may be able to make the X server synchronize with the mouse by switching away to a virtual terminal and getting back to the X server, unless the X server is accessing the mouse via moused(8). Clicking any button without moving the mouse may also work.

BUGS

Enabling the extended support for Synaptics touchpads has been reported to cause problems with responsivity on some (newer) models of Synaptics hardware, particularly those with guest devices.