Philips webcams (pwc driver)

This file contains some additional information for the Philips and OEM webcams. E-mail: webcam@smcc.demon.nl Last updated: 2004-01-19 Site: http://www.smcc.demon.nl/webcam/

As of this moment, the following cameras are supported:

- Philips PCA645
- Philips PCA646
- Philips PCVC675
- Philips PCVC680
- Philips PCVC690
- Philips PCVC720/40
- Philips PCVC730
- Philips PCVC740
- Philips PCVC750
- Askey VC010
- Creative Labs Webcam 5
- Creative Labs Webcam Pro Ex
- Logitech QuickCam 3000 Pro
- Logitech QuickCam 4000 Pro
- Logitech QuickCam Notebook Pro
- Logitech QuickCam Zoom
- Logitech QuickCam Orbit
- Logitech QuickCam Sphere
- Samsung MPC-C10
- Samsung MPC-C30
- Sotec Afina Eye
- AME CU-001
- Visionite VCS-UM100
- Visionite VCS-UC300

The main webpage for the Philips driver is at the address above. It contains a lot of extra information, a FAQ, and the binary plugin 'PWCX'. This plugin contains decompression routines that allow you to use higher image sizes and framerates; in addition the webcam uses less bandwidth on the USB bus (handy if you want to run more than 1 camera simultaneously). These routines fall under a NDA, and may therefore not be distributed as source; however, its use is completely optional.

You can build this code either into your kernel, or as a module. I recommend the latter, since it makes troubleshooting a lot easier. The built-in microphone is supported through the USB Audio class.

When you load the module you can set some default settings for the camera; some programs depend on a particular image-size or -format and don't know how to set it properly in the driver. The options are:

size

Can be one of 'sqcif', 'qsif', 'qcif', 'sif', 'cif' or 'vga', for an image size of resp. 128x96, 160x120, 176x144, 320x240, 352x288 and 640x480 (of course, only for those cameras that support these resolutions).

fps

Specifies the desired framerate. Is an integer in the range of 4-30.

fbufs

This parameter specifies the number of internal buffers to use for storing frames from the cam. This will help if the process that reads images from the cam is a bit slow or momentarily busy. However, on slow machines it only introduces lag, so choose carefully. The default is 3, which is reasonable. You can set it between 2 and 5.

mbufs

This is an integer between 1 and 10. It will tell the module the number of buffers to reserve for mmap(), VIDIOCCGMBUF, VIDIOCMCAPTURE and friends. The default is 2, which is adequate for most applications (double buffering).

Should you experience a lot of 'Dumping frame...' messages during grabbing with a tool that uses mmap(), you might want to increase if. However, it doesn't really buffer images, it just gives you a bit more slack when your program is behind. But you need a multi-threaded or forked program to really take advantage of these buffers.

The absolute maximum is 10, but don't set it too high! Every buffer takes up 460 KB of RAM, so unless you have a lot of memory setting this to something more than 4 is an absolute waste. This memory is only allocated during open(), so nothing is wasted when the camera is not in use.

power_save

When power_save is enabled (set to 1), the module will try to shut down the cam on close() and re-activate on open(). This will save power and turn off the LED. Not all cameras support this though (the 645 and 646 don't have power saving at all), and some models don't work either (they will shut down, but never wake up). Consider this experimental. By default this option is disabled.

compression (only useful with the plugin)

With this option you can control the compression factor that the camera uses to squeeze the image through the USB bus. You can set the parameter between 0 and 3:

```
0 = prefer uncompressed images; if the requested mode is not available
    in an uncompressed format, the driver will silently switch to low
    compression.
1 = low compression.
2 = medium compression.
3 = high compression.
```

High compression takes less bandwidth of course, but it could also introduce some unwanted artefacts. The default is 2, medium compression. See the FAQ on the website for an overview of which modes require compression.

The compression parameter does not apply to the 645 and 646 cameras and OEM models derived from those (only a few). Most cams honour this parameter.

leds

This settings takes 2 integers, that define the on/off time for the LED (in milliseconds). One of the interesting things that you can do with this is let the LED blink while the camera is in use. This:

```
leds=500,500
```

will blink the LED once every second. But with:

```
leds=0,0
```

the LED never goes on, making it suitable for silent surveillance.

By default the camera's LED is on solid while in use, and turned off when the camera is not used anymore.

This parameter works only with the ToUCam range of cameras (720, 730, 740, 750) and OEMs. For other cameras this command is silently ignored, and the LED cannot be controlled.

Finally: this parameters does not take effect UNTIL the first time you open the camera device. Until then, the LED remains on.

dev_hint

A long standing problem with USB devices is their dynamic nature: you never know what device a camera gets assigned; it depends on module load order, the hub configuration, the order in which devices are plugged in, and the phase of the moon (i.e. it can be random). With this option you can give the driver a hint as to what video device node (/dev/videoX) it should use with a specific camera. This is also handy if you have two cameras of the same model.

A camera is specified by its type (the number from the camera model, like PCA645, PCVC750VC, etc) and optionally the serial number (visible in /sys/kernel/debug/usb/devices). A hint consists of a string with the following format:

```
[type[.serialnumber]:]node
```

The square brackets mean that both the type and the serialnumber are optional, but a serialnumber cannot be specified without a type (which would be rather pointless). The serialnumber is separated from the type by a '.'; the node number by a ':'.

This somewhat cryptic syntax is best explained by a few examples:

```
dev_hint=3,5              The first detected cam gets assigned
                          /dev/video3, the second /dev/video5. Any
                          other cameras will get the first free
                          available slot (see below).

dev_hint=645:1,680:2      The PCA645 camera will get /dev/video1,
                          and a PCVC680 /dev/video2.

dev_hint=645.0123:3,645.4567:0     The PCA645 camera with serialnumber
                                   0123 goes to /dev/video3, the same
                                   camera model with the 4567 serial
                                   gets /dev/video0.

dev_hint=750:1,4,5,6       The PCVC750 camera will get /dev/video1, the
                           next 3 Philips cams will use /dev/video4
                           through /dev/video6.
```

Some points worth knowing:

- Serialnumbers are case sensitive and must be written full, including leading zeroes (it's treated as a string).
- If a device node is already occupied, registration will fail and the webcam is not available.
- You can have up to 64 video devices; be sure to make enough device nodes in /dev if you want to spread the numbers. After /dev/video9 comes /dev/video10 (not /dev/videoA).
- If a camera does not match any dev_hint, it will simply get assigned the first available device node, just as it used to be.

trace

In order to better detect problems, it is now possible to turn on a 'trace' of some of the calls the module makes; it logs all items in your kernel log at debug level.

The trace variable is a bitmask; each bit represents a certain feature. If you want to trace something, look up the bit value(s) in the table below, add the values together and supply that to the trace variable.

Value (dec)	**Value (hex)**	**Description**	**Default**
1	0x1	Module initialization; this will log messages while loading and unloading the module	On
2	0x2	probe() and disconnect() traces	On
4	0x4	Trace open() and close() calls	Off
8	0x8	read(), mmap() and associated ioctl() calls	Off
16	0x10	Memory allocation of buffers, etc.	Off
32	0x20	Showing underflow, overflow and Dumping frame messages	On
64	0x40	Show viewport and image sizes	Off
128	0x80	PWCX debugging	Off

For example, to trace the open() & read() functions, sum 8 + 4 = 12, so you would supply trace=12 during insmod or modprobe. If you want to turn the initialization and probing tracing off, set trace=0. The default value for trace is 35 (0x23).

Example:

```
# modprobe pwc size=cif fps=15 power_save=1
```

The fbufs, mbufs and trace parameters are global and apply to all connected cameras. Each camera has its own set of buffers.

size and fps only specify defaults when you open() the device; this is to accommodate some tools that don't set the size. You can change these settings after open() with the Video4Linux ioctl() calls. The default of defaults is QCIF size at 10 fps.

The compression parameter is semiglobal; it sets the initial compression preference for all camera's, but this parameter can be set per camera with the VIDIOCPWCSCQUAL ioctl() call.

All parameters are optional.