README for Linux device driver for the IBM "C-It" USB video camera

INTRODUCTION:

This driver does not use all features known to exist in
the IBM camera. However most of needed features work well.

This driver was developed using logs of observed USB traffic
which was produced by standard Windows driver (c-it98.sys).
I did not have data sheets from Xirlink.

Video formats:
      128x96 [model 1]
      176x144
      320x240 [model 2]
      352x240 [model 2]
      352x288
Frame rate: 3 - 30 frames per second (FPS)
External interface: USB
Internal interface: Video For Linux (V4L)
Supported controls:
- by V4L: Contrast, Brightness, Color, Hue
- by driver options: frame rate, lighting conditions, video format,
                     default picture settings, sharpness.

SUPPORTED CAMERAS:

Xirlink "C-It" camera, also known as "IBM PC Camera".
The device uses proprietary ASIC (and compression method);
it is manufactured by Xirlink. See http://www.xirlink.com/
(renamed to http://www.veo.com), http://www.ibmpccamera.com,
or http://www.c-itnow.com/ for details and pictures.

This very chipset ("X Chip", as marked at the factory)
is used in several other cameras, and they are supported
as well:

- IBM NetCamera
- Veo Stingray

The Linux driver was developed with camera with following
model number (or FCC ID): KSX-XVP510. This camera has three
interfaces, each with one endpoint (control, iso, iso). This
type of cameras is referred to as "model 1". These cameras are
no longer manufactured.

Xirlink now manufactures new cameras which are somewhat different.
In particular, following models [FCC ID] belong to that category:

XVP300 [KSX-X9903]
XVP600 [KSX-X9902]
XVP610 [KSX-X9902]

(see http://www.xirlink.com/ibmpccamera/ for updates, they refer
to these new cameras by Windows driver dated 12-27-99, v3005 BETA)
These cameras have two interfaces, one endpoint in each (iso, bulk).

Such type of cameras is referred to as "model 2". They are supported
(with exception of 352x288 native mode).

Some IBM NetCameras (Model 4) are made to generate only compressed
video streams. This is great for performance, but unfortunately
nobody knows how to decompress the stream :-(Therefore, these
cameras are *unsupported* and if you try to use one of those, all
you get is random colored horizontal streaks, not the image!
If you have one of those cameras, you probably should return it
to the store and get something that is supported.

Tell me more about all that "model" business

I just invented model numbers to uniquely identify flavors of the
hardware/firmware that were sold. It was very confusing to use
brand names or some other internal numbering schemes. So I found
by experimentation that all Xirlink chipsets fall into four big
classes, and I called them "models". Each model is programmed in
its own way, and each model sends back the video in its own way.

Quirks of Model 2 cameras:

Model 2 does not have hardware contrast control. Corresponding V4L
control is implemented in software, which is not very nice to your
CPU, but at least it works.

This driver provides 352x288 mode by switching the camera into
quasi-352x288 RGB mode (800 Kbits per frame) essentially limiting
this mode to 10 frames per second or less, in ideal conditions on
the bus (USB is shared, after all). The frame rate
has to be programmed very conservatively. Additional concern is that
frame rate depends on brightness setting; therefore the picture can
be good at one brightness and broken at another! I did not want to fix
the frame rate at slowest setting, but I had to move it pretty much down
the scale (so that framerate option barely matters). I also noticed that
camera after first powering up produces frames slightly faster than during
consecutive uses. All this means that if you use 352x288 (which is
default), be warned - you may encounter broken picture on first connect;
try to adjust brightness - brighter image is slower, so USB will be able
to send all data. However if you regularly use Model 2 cameras you may
prefer 176x144 which makes perfectly good I420, with no scaling and
lesser demands on USB (300 Kbits per second, or 26 frames per second).

Another strange effect of 352x288 mode is the fine vertical grid visible
on some colored surfaces. I am sure it is caused by me not understanding
what the camera is trying to say. Blame trade secrets for that.

The camera that I had also has a hardware quirk: if disconnected,
it needs few minutes to "relax" before it can be plugged in again
(poorly designed USB processor reset circuit?)

[Veo Stingray with Product ID 0x800C is also Model 2, but I haven't
observed this particular flaw in it.]

Model 2 camera can be programmed for very high sensitivity (even starlight
may be enough), this makes it convenient for tinkering with. The driver
code has enough comments to help a programmer to tweak the camera
as s/he feels necessary.

WHAT YOU NEED:

- A supported IBM PC (C-it) camera (model 1 or 2)

- A Linux box with USB support (2.3/2.4; 2.2 w/backport may work)

- A Video4Linux compatible frame grabber program such as xawtv.

HOW TO COMPILE THE DRIVER:

You need to compile the driver only if you are a developer
or if you want to make changes to the code. Most distributions
precompile all modules, so you can go directly to the next
section "HOW TO USE THE DRIVER".

The ibmcam driver uses usbvideo helper library (module),
so if you are studying the ibmcam code you will be led there.

The driver itself consists of only one file in usb/ directory:
ibmcam.c. This file is included into the Linux kernel build
process if you configure the kernel for CONFIG_USB_IBMCAM.
Run "make xconfig" and in USB section you will find the IBM
camera driver. Select it, save the configuration and recompile.

HOW TO USE THE DRIVER:

I recommend to compile driver as a module. This gives you an
easier access to its configuration. The camera has many more
settings than V4L can operate, so some settings are done using
module options.

To begin with, on most modern Linux distributions the driver
will be automatically loaded whenever you plug the supported
camera in. Therefore, you don't need to do anything. However
if you want to experiment with some module parameters then
you can load and unload the driver manually, with camera
plugged in or unplugged.

Typically module is installed with command 'modprobe', like this:

modprobe ibmcam framerate=1

Alternatively you can use 'insmod' in similar fashion:

insmod /lib/modules/2.x.y/usb/ibmcam.o framerate=1

Module can be inserted with camera connected or disconnected.

The driver can have options, though some defaults are provided.

Driver options: (* indicates that option is model-dependent)

Name	Type	Range [default]	Example
debug	Integer	0-9 [0]	debug=1
flags	Integer	0-0xFF [0]	flags=0x0d
framerate	Integer	0-6 [2]	framerate=1
hue_correction	Integer	0-255 [128]	hue_correction=115
init_brightness	Integer	0-255 [128]	init_brightness=100
init_contrast	Integer	0-255 [192]	init_contrast=200
init_color	Integer	0-255 [128]	init_color=130
init_hue	Integer	0-255 [128]	init_hue=115
lighting	Integer	0-2* [1]	lighting=2
sharpness	Integer	0-6* [4]	sharpness=3
size	Integer	0-2* [2]	size=1

Options for Model 2 only:

Name	Type	Range [default]	Example
init_model2_rg	Integer	0..255 [0x70]	init_model2_rg=128
init_model2_rg2	Integer	0..255 [0x2f]	init_model2_rg2=50
init_model2_sat	Integer	0..255 [0x34]	init_model2_sat=65
init_model2_yb	Integer	0..255 [0xa0]	init_model2_yb=200

debug — You don't need this option unless you are a developer. If you are a developer then you will see in the code what values do what. 0=off.

flags — This is a bit mask, and you can combine any number of bits to produce what you want. Usually you don't want any of extra features this option provides:

FLAGS_RETRY_VIDIOCSYNC	1	This bit allows to retry failed VIDIOCSYNC ioctls without failing. Will work with xawtv, will not with xrealproducer. Default is not set.
FLAGS_MONOCHROME	2	Activates monochrome (b/w) mode.
FLAGS_DISPLAY_HINTS	4	Shows colored pixels which have magic meaning to developers.
FLAGS_OVERLAY_STATS	8	Shows tiny numbers on screen, useful only for debugging.
FLAGS_FORCE_TESTPATTERN	16	Shows blue screen with numbers.
FLAGS_SEPARATE_FRAMES	32	Shows each frame separately, as it was received from the camera. Default (not set) is to mix the preceding frame in to compensate for occasional loss of Isoc data on high frame rates.
FLAGS_CLEAN_FRAMES	64	Forces "cleanup" of each frame prior to use; relevant only if FLAGS_SEPARATE_FRAMES is set. Default is not to clean frames, this is a little faster but may produce flicker if frame rate is too high and Isoc data gets lost.

FLAGS_NO_DECODING	128	This flag turns the video stream decoder off, and dumps the raw Isoc data from the camera into the reading process. Useful to developers, but not to users.

framerate — This setting controls frame rate of the camera. This is an approximate setting (in terms of "worst" ... "best") because camera changes frame rate depending on amount of light available. Setting 0 is slowest, 6 is fastest. Beware - fast settings are very demanding and may not work well with all video sizes. Be conservative.

hue_correction — This highly optional setting allows to adjust the hue of the image in a way slightly different from what usual "hue" control does. Both controls affect YUV colorspace: regular "hue" control adjusts only U component, and this "hue_correction" option similarly adjusts only V component. However usually it is enough to tweak only U or V to compensate for colored light or color temperature; this option simply allows more complicated correction when and if it is necessary.

init_brightness, init_contrast, init_color, init_hue — These settings specify _initial_ values which will be used to set up the camera. If your V4L application has its own controls to adjust the picture then these controls will be used too. These options allow you to preconfigure the camera when it gets connected, before any V4L application connects to it. Good for webcams.

init_model2_rg, init_model2_rg2, init_model2_sat, init_model2_yb — These initial settings alter color balance of the camera on hardware level. All four settings may be used to tune the camera to specific lighting conditions. These settings only apply to Model 2 cameras.

lighting — This option selects one of three hardware-defined photosensitivity settings of the camera. 0=bright light, 1=Medium (default), 2=Low light. This setting affects frame rate: the dimmer the lighting the lower the frame rate (because longer exposition time is needed). The Model 2 cameras allow values more than 2 for this option, thus enabling extremely high sensitivity at cost of frame rate, color saturation and imaging sensor noise.

sharpness — This option controls smoothing (noise reduction) made by camera. Setting 0 is most smooth, setting 6 is most sharp. Be aware that CMOS sensor used in the camera is pretty noisy, so if you choose 6 you will be greeted with "snowy" image. Default is 4. Model 2 cameras do not support this feature.

size — This setting chooses one of several image sizes that are supported by this driver. Cameras may support more, but it's difficult to reverse-engineer all formats. Following video sizes are supported:

size=0	128x96	(Model 1 only)
size=1	160x120	
size=2	176x144	
size=3	320x240	(Model 2 only)
size=4	352x240	(Model 2 only)
size=5	352x288	
size=6	640x480	(Model 3 only)

The 352x288 is the native size of the Model 1 sensor array, so it's the best resolution the camera can yield. The best resolution of Model 2 is 176x144, and larger images are produced by stretching the bitmap. Model 3 has sensor with 640x480 grid, and it works too, but the frame rate will be exceptionally low (1-2 FPS); it may be still OK for some applications, like security. Choose the image size you need. The smaller image can support faster frame rate. Default is 352x288.

For more information and the Troubleshooting FAQ visit this URL:

http://www.linux-usb.org/ibmcam/

WHAT NEEDS TO BE DONE:

- The button on the camera is not used. I don't know how to get to it. I know now how to read button on Model 2, but what to do with it?

- Camera reports its status back to the driver; however I don't know what returned data means. If camera fails at some initialization stage then something should be done, and I don't do that because I don't even know that some command failed. This is mostly Model 1 concern because Model 2 uses different commands which do not return status (and seem to complete successfully every time).

- Some flavors of Model 4 NetCameras produce only compressed video streams, and I don't know how to decode them.

CREDITS:

The code is based in no small part on the CPiA driver by Johannes Erdfelt, Randy Dunlap, and others. Big thanks to them for their pioneering work on that and the USB stack.

I also thank John Lightsey for his donation of the Veo Stingray camera.