Video For Linux 2

Sample Program Documentation

By: Aquiles Yáñez Cañas

About Video For Linux 2

Video For Linux 2, is the second version of the Video for Linux API. This New API is more advanced, complex, and it has resolved many design leaks of the former version. Video For Linux 2 (V4L2) is a totally different API, with respect to v4l, that means it uses different drivers and have different functions invocations (ioctl) and headers files.

The programs described in this page, allow the capture and show of video using the v4l2 API. This API is included by default in the 2.6 kernel series. In the 2.4 kernels series, this API is not used natively by default, and thus, these programs don't work, but the installation of the v4l2 API on kernels 2.4 is possible, but a little bit cumbersome.

Every device works with V4l2, only if has a kernel module (driver), for this API (v4l2). For backward compatibity, v4l2 has a v4l compatibility module, allowing the execution of legacies video for Linux I programs (sometimes with limitations).

Programs Description

All programs were packaged in only one package (.tar.gz)

capturer_mmap	Program who captures video using memory mapping and throws the data to standard output
capturer_read	Program who captures video using the read function and throws the data to the standard output
viewer	Program who reads data from standard input and shows that data on a window in order to display a video. This program auto detects the depth of the desktop (16 or 24) and makes internal conversion of the data in order to display it correctly.

These programs were made by Aquiles Yáñez C.(yanez<at>elo<dot>utfsm<dot>cl) under the design guidance of Agustín González

Versions changelog and URL's for download

This documentation is based on the last version of the programs (v0.4.1).

Jan 2010 - **Version - 0.4.1**

- Bug fixed on the viewer, missing pointer initialization, causing segfault sometimes.
- New version policy with 3 numbers is used. Useful for bug fix releases, like this one.

Nov 2009 - **Version - 0.4**

- This version translates the source code and comments to English.
- The viewer is converted to c from c++ (now all the source code is c).

Oct 2006 - **Version - 0.3**

- Program was divided in 3 new programs, two capturers and one viewer.
- This version was documented in the old spanish website: http://alumnos.elo.utfsm.cl/~yanez/atmlab/ejemplov4l2/

Feb 2005 - **Version - 0.2**

- Removal of global variables.
- Source code comments added.
- The versions 0.1 and 0.2 are explained in the old site of these demo programs: http://alumnos.elo.utfsm.cl/~yanez/atmlab/etapav4l2/

Jan 2005 - **Version - 0.1**

- Initial version, only one program does everything.

- The versions 0.1 and 0.2 are explained in the old site of these demo programs: http://alumnos.elo.utfsm.cl/~yanez/atmlab/etapav4l2/

Phillips ToUcam PRO II y QuickCam Pro 4000 Driver Configuration

The driver used by these cameras is called *PWC*. In some kernels of 2.6 series, this driver implemented the v4l2 API, like the 2.6.8 (used by Debian Sarge), Instead in the newer kernels, like 2.6.15, for some reason the maintainers of the kernel module decided to implement the former API v4l, by this reason, in these kernel the programs doesn't work).

To load the module pwc, is necessary to manually configure their parameters of capture. For example to capture video in 640x480 with 30 fps, must be loaded as follows:

```
# modprobe -r pwc
# modprobe -v pwc fps=15 compression=3 mbufs=4 fbufs=4 size=vga
```

However, if you want to capture in 320x240 with 30 pfs, must be loaded as follows:

```
# modprobe -r pwc
# modprobe -v pwc fps=30 compression=3 mbufs=4 fbufs=4 size=cif
```

Due to limitations with the USB1.1 technology, with these cameras, it doesn't possible to capture video with 640x480@30fps. But their Windows driver allow that, but this is possible only with tricks like interpolations. A table with the definitions of sizes is shown below:

sQCIF	**128x96**
QSIF	**160x120**
QCIF	**176x144**
SIF	**320x 240**
CIF	**352x288**
VGA	**640x48**

Capture Cards Driver Configuration

The capture cards depending on the chip that they have, could work with different drivers like: bttv or saa7130. These two drivers in their documentation includes a list of their supported card by manufacturer and model. For a basic operation of capture cards (like video capture from composite video input), only is necessary to have loaded the right driver. For the family bt8x8 (bt878 and bt848) the bttv module is used, and that module below uses others modules like bt878. And for the family saa713x (saa7130 and saa7134) the module saa7134 is used.

In addition, if you want to get working the TV or the FM tuner, is necessary the right configuration of the tuner type when the driver is loaded.

And in addition if you want get working all the capabilities of the card (like audio and the previous listed) is necessary to configure the right card model when the driver is loaded, Remember that both drivers in their documentation include a complete list of supported cards model

In some cards the model and the tuner type is automatically detected (due to additional hardware like EEPROM), and the driver is always loaded with the right configuration.

The documentation of v4l2 (including the card and tuner list) is included in the kernel sources, and could be installed by the following way in Debian Sarge:

```
# apt-get install kernel-source-2.6.8
```

The most important files are:

```
CARDLIST.bttv
CARDLIST.saa7134
CARDLIST.tuner
```

The documentation is located in the directory: `/usr/src/kernel-source-2.6.8/Documentation/video4linux/`

If the right tuner and card type is selected from the previous files, you need to load the kernels modules. For example for the card TV Master+FM, you must run:

```
# modprobe bttv tuner=17 card=70 radio=1
```

radio=1 means that FM support will be enabled

To remove the kernel module, run:

```
# modprobe -r bt878
# modprobe -r bttv
```

Build of the programs

First, the xlibs development files must be installed, by the following way in Debian Sarge:

```
# apt-get install xlibs-dev
```

And the build of the programs, looks like:

```
# tar -zxvf V4l2_samples-0.4.tar.gz
# cd V4l2_samples-0.4
# make
```

Finally to delete executables and (.o) files, run:

```
make clean
```

Programs syntax

capturer_mmap

To get usage info from command line, run:

```
# ./capturer_mmap -h
```

To get info about the configuration options of a certain device, run:

```
# ./capturer_mmap [-d Capture Device]
```

Capture Device: Video For Linux II device, eg: /dev/video

To capture video, and throw the data to standard output, run:

```
# ./capturer_mmap [-D Capture Device] [-i Input] [-s Standard] [-w Window Size] [-p Pixel Format]
```

Capture Device: Video For Linux II device, eg: /dev/video

Input: Input channel of the capture device, if you want to know the available values for a certain device, just use the program with the -d switch

Standard: A number, who represents the TV norm used (eg: NTSC). if you want to know the values for a certain device, use the -d switch

Window Size: Could be: 320*240 or 640*480

Pixel Format: A number, who represents the format used to save the images in the memory buffer, the values could be:

```
0 YUV420
1 RGB565
2 RGB32
```

capturer_read

To get usage info from command line, run:

```
# ./capturador_read -h
```

Capture Device: Video For Linux II device, eg: /dev/video

To capture video, and throw the data to standard output, run:

```
# ./capturer_mmap [-D Capture Device] [-i Input] [-s Standard] [-w Window Size] [-p Pixel Format]
```

Capture Device: Video For Linux II device, eg: /dev/video

To capture video, and throw the data to standard output, run:

```
# ./capturer_read [-D Capture Device] [-i Input] [-s Standard] [-w Window Size] [-p Pixel Format]
```

Capture Device: Video For Linux II device, eg: /dev/video

Input: Input channel of the capture device, if you want to know the available values for a certain device, just use the program with the -d switch

Standard: A number, who represents the TV norm used (eg: NTSC). if you want to know the values for a certain device, use the -d switch

Window Size: Could be: 320*240 or 640*480

Pixel Format: A number, who represents the format used to save the images in the memory buffer. the values could be:

```
0 YUV420
1 RGB565
2 RGB32
```

viewer

To get info of program usage, run:

```
# ./viewer -h
```

To display video, receiving data from the standard input, run:

```
#./viewer [-w Window Size] [-p Pixel Format]
```

Window Size: Could be: 320*240 or 640*480

Pixel Format: A number, who represents the format used in the memory buffer

```
0 YUV420
1 RGB565
2 RGB32
```

How To Use The Sample Programs

The programs were designed to work only in a concatenated way. In one side the video stream data can be generated by the capturer_mmap or by the capturer_read, and in the other side the stream must be readed and displayed by the viewer program. Both programs work together concatenated directly or with other program in the middle, like a video encoder or decoder. Moreover both programs can run in different machines and can be concatenated with a program who transmits (capturer side) and receives (viewer side) the stream over the network.

Examples:

To capture video with a resolution of 640x480, with a webcam who uses pwc driver (Phillips, Logitech), just run:

```
# modprobe -r pwc
# modprobe -v pwc fps=15 compression=3 mbufs=4 fbufs=4 size=vga
# ./capturer_mmap -D /dev/video -w 640*480 -p 0 | ./viewer -w 640*480 -p 0
```

To capture video with a resolution of 320x240, with a webcam who use pwc driver (Phillips, Logitech), just run:

```
# modprobe -r pwc
# modprobe -v pwc fps=30 compression=3 mbufs=4 fbufs=4 size=sif
# ./capturer_mmap -D /dev/video -w 320*240 -p 0 | ./viewer -w 320*240 -p 0
```

To capture video with a resolution of 320x240, with a capture card who uses bttv driver (i.e. TV Master+FM), just run:

```
# modprobe -r bt878
# modprobe -r bttv
# modprobe modprobe bttv tuner=17 card=70 radio=1
# ./capturer_mmap -D /dev/video0 -w 320*240 -p 0 | ./viewer -w 320*240 -p 0
```

Screen shoot for this example:

Video For Linux II Utility Programs

XAWTV

This program allows to watch videos captured by devices who use Video For Linux and Video For Linux II. To install this program in Debian GNU/Linux run:

```
# apt-get install xawtv
```

Another very helpfully feature of this program is the info displayed with the flag hwscan. This info let us know the info of every device, and the most important, which version of the API every device support (Video For Linux I or II).

```
# xawtv -hwscan
```

Example:

```
$ xawtv -hwscan
This is xawtv-3.94, running on Linux/i686 (2.6.8-1-686-smp)
looking for available devices
Xlib: extension "XVideo" missing on display ":0.0".
/dev/video0: OK [ -device/dev/video0 ]
type : v4l2
name : Philips 740 webcam
flags: capture
```

The parameter type of every device, indicates the version of the API (V4l or v4l2). And finally to watch a video from a certain device, in this example /dev/video0, just run:

```
# xawtv -device /dev/video0
```

Video For Linux II - Basic Programs Structure

This section shows the basic structure of programs who use the Video For Linux II API

Next the Annex F-2 , of the "Memoria de titulación" of the students Aquiles Yáñez (also author of the programs and this documentation) and Mauricio Venegas, made on the second semester of 2005, under the guide of Agustín González, with the subject "High Quality Video Transmission Over IP Networks"

F.2. Programming Basics

In most cases, a Video for Linux II application may execute the next steps:

- Open the device.
- Properties Negotiation (video input, video standard, and more)
- Pixel Format Negotiation
- Input/Output Method
- Main Loop
- Close the device

The V4L2 applications must be programmed using the *ioctl* system function. *ioctl* is used with the following syntax:

```
 ioctl(<device_descriptor>, <operation_code>, <pointer_to_structure>)
```

The next table, shows the available operation codes with their corresponding structures types.

op. code	**I/O**	**structure**
VIDIOC_QUERYCAP	IOR	struct v4l2_capability
VIDIOC_RESERVED	IO	1
VIDIOC_ENUM_FMT	IOWR	struct v4l2_fmtdesc
VIDIOC_G_FMT	IOWR	struct v4l2_format
VIDIOC_S_FMT	IOWR	struct v4l2_format
VIDIOC_G_COMP	IOR	struct v4l2_compression
VIDIOC_S_COMP	IOW	struct v4l2_compression
VIDIOC_REQBUFS	IOWR	struct v4l2_requestbuffers
VIDIOC_QUERYBUF	IOWR	struct v4l2_buffer
VIDIOC_G_FBUF	IOR	struct v4l2_framebuffer
VIDIOC_S_FBUF	IOW	struct v4l2_framebuffer
VIDIOC_OVERLAY	IOWR	int
VIDIOC_QBUF	IOWR	struct v4l2_buffer
VIDIOC_DQBUF	IOWR	struct v4l2_buffer
VIDIOC_STREAMON	IOW	int
VIDIOC_STREAMOFF	IOW	int
VIDIOC_G_PARM	IOWR	struct v4l2_streamparm
VIDIOC_S_PARM	IOW	struct v4l2_streamparm
VIDIOC_G_STD	IOR	v4l2_std_id
VIDIOC_S_STD	IOW	v4l2_std_id
VIDIOC_ENUMSTD	IOWR	struct v4l2_standard
VIDIOC_ENUMINPUT	IOWR	struct v4l2_input
VIDIOC_G_CTRL	IOWR	struct v4l2_control
VIDIOC_S_CTRL	IOW	struct v4l2_control
VIDIOC_G_TUNER	IOWR	struct v4l2_tuner
VIDIOC_S_TUNER	IOW	struct v4l2_tuner
VIDIOC_G_AUDIO	IOWR	struct v4l2_audio
VIDIOC_S_AUDIO	IOW	struct v4l2_audio
VIDIOC_QUERYCTRL	IOWR	struct v4l2_queryctrl
VIDIOC_QUERYMENU	IOWR	struct v4l2_querymenu
VIDIOC_G_INPUT	IOR	int
VIDIOC_S_INPUT	IOWR	int
VIDIOC_G_OUTPUT	IOR	int
VIDIOC_S_OUTPUT	IOWR	int
VIDIOC_ENUMOUTPUT	IOWR	struct v4l2_output
VIDIOC_G_AUDOUT	IOWR	struct v4l2_audioout
VIDIOC_S_AUDOUT	IOW	struct v4l2_audioout
VIDIOC_G_MODULATOR	IOWR	struct v4l2_modulator
VIDIOC_S_MODULATOR	IOW	struct v4l2_modulator
VIDIOC_G_FREQUENCY	IOWR	struct v4l2_frequency
VIDIOC_S_FREQUENCY	IOW	struct v4l2_frequency
VIDIOC_CROPCAP	IOR	struct v4l2_cropcap
VIDIOC_G_CROP	IOWR	struct v4l2_crop
VIDIOC_S_CROP	IOW	struct v4l2_crop
VIDIOC_G_JPEGCOMP	IOR	struct v4l2_jpegcompression
VIDIOC_S_JPEGCOMP	IOW	struct v4l2_jpegcompression
VIDIOC_QUERYSTD	IOR	v4l2_std_id
VIDIOC_TRY_FMT	IOWR	struct v4l2_format

Table F.2.1: ioctl parameters for Video For Linux II

The following section, explain in more detail the steps in a V4l2 program.

F.2.1. Open the device

The V4L2 devices, are character devices of the kernel and can be opened with the *open* function, using a file descriptor, like every other character device.

The table F.2.2, shows the Video For Linux II available devices.

Device Name		**Minor Number**	**Description**
From	**To**		
/dev/video0	/dev/video63	0-63	Video Capturer Devices
/dev/radio0	/dev/radio63	64-127	AM/FM Radio Devices
/dev/vtx0	/dev/vtx31	192-223	Teletext Devices
/dev/vbi0	/dev/vbi15	224-239	VBI Devices

Table F.2.2. description of Video For Linux II devices

In V4l2 most device can be opened several times, but only one application, can change the configuration of the device.

F.2.2. Properties Negotiation

We talk about negotiation instead of configuration, because, first the application asks for the possibles values of some property, next chooses one of the possible values, next configures that value, and finally checks the right configuration of the value. The most important properties to set on a Video For Linux II device, are:

- Video Input.
- Video Output.
- Norm (only for input devices).
- Modulator (only for output devices).
- Input Channel.
- Window Size.

F.2.3. Pixel Format Negotiation.

Pixel format negotiation is also a part of the properties negotiation, but is on a separated section, just to be explained more in detail. The pixel format is how every pixel is stored in memory, and the application need to know this format to allow the properly interpretation of that pixel. There are two "families" of pixel formats RGB and YUV.

YUV format are useful in the input of video codecs, in fact YUV formats have a basic kind of compression, because they reduce the crominance components, because these are less perceptible to the human eye. On the other side RGB format keep all the crominance information, and these are useful sometimes for video processing or displaying on a window.

In the most cases the devices capture natively in YUV formats, and in these programs, the video is converted to RGB formats, for displaying in the viewer, this takes some time of cpu.

F.2.4. Input/Output Method Negotiation

The V4L2 API, has tree different methods for input/output, these are:

F.2.4.1 Using READ/WRITE functions

The devices supports this method, when the flag V4L2_CAP_READWRITE is set on the capabilities member of the structure V4l2_capability. The use of this function could be slower than the other methods, because all the data must be copied.

F.2.4.2. Using Memory mapping

This is the fastest method, read and write functions are not needed, instead of those, the mmap() function is used. This functions returns a pointer to the start of a valid memory area, this memory is used by the application to read the data. A device support this method when the flag V4L2_CAP_STREAMING of the capabilities field in the v4l2_capability struct is set.

F.2.4.3. Using User Pointer

A device supports this method if the field V4L2_CAP_STREAMING in the member capabilities of the struct V4l2_capability returned by the VIDIOC_QUERYCAP ioctl is set. If the particular user pointer method (not only memory mapping) is supported must be determined by calling the VIDIOC_REQBUFS ioctl.

This method combines the advantage of the both previous methods. In the user pointer method buffers are allocated by the application and can be shared memory (mmap) or virtual. Only pointers to data are exchanged, these pointers and meta-information are passed in struct v4l2_buffer. The driver must be switched into user pointer I/O mode by calling the VIDIOC_REQBUFS with the desired buffer type. This allow the use of DMA.

F.2.5. Main loop

If user pointer or memory mapping is used, that are streaming oriented methods, the first step is the start of the transmission of data. In addition these methods have buffers with queues, in that way, V4L2 enqueue data, and the application dequeue the data, with a FIFO criteria. In the case of output devices, the process is the same but inverse. In addition the buffer must be previously configured with the properties of the frames, to know the size of one frame, for allocation of memory. And at the end of the loop, the data transmission must be stopped and the buffers freed.

In the case of use of read/write functions in the main loop, the application only have one memory buffer, that one allows the store of one frame. On a capture application in every loop the read function must be called to capture one frame.

For the 3 methods is valid the use of the select() function, to wait for events, avoiding the use of CPU when is waiting.

F.2.6. Close the device.

For the use of the 3 methods of I/O, the close function is used to close the device.