libdc1394 v2.x :: FAQ

Home
News
Download
Links
IIDC specs
Credits
Sponsors

FAQs
libdc1394 1.x
libdc1394 2.x

API v1
Migration

API v2
Intro
Types
Structures
System
Trigger
Features
Video
Capture
Format_7
Conversions
Utilities
Errors
AVT

Random photo

This FAQ is provided on an "as is" basis. There is no warranty whatsoever, either express or implied, regarding the FAQ, including warranties with respect to its merchantability or fitness for any particular purpose.

Recent ChangeLog

- 2008-11-16: removed the reference to the "drop_frames" parameter.
- 2007-09-26: one faq added.
- 2006-11-27: five or six entries added.
- 2006-10-12: Added 2 FAQs: one about the meaning of a video1394 device and another one about the use of the ISO channel / bandwidth cleanup function. Updated the URL for the IIDC specs.
- 2006-10-11: FAQ imported from Johann's website and adapted for v2.x. WORK IN PROGRESS... This documents needs to be proofread.

Questions

Answers

What is libdc1394?

libdc1394 is a library that is intended to provide a high level programming interface for application developers who wish to control IEEE 1394 based cameras that conform to the 1394-based Digital Camera Specification of the 1394 Trade Association (1394-TA). The specifications can be found on this website. The library development is an open-source project (homepage sourceforge.net/projects/libdc1394) licensed under the GNU Lesser General Public License (LGPL) www.gnu.org/copyleft/lesser.html. It was originally designed for the Linux platform but is now also compatible with OSX.

What is IEEE 1394?

IEEE 1394 is the name of a serial bus standard designed to be a versatile, high-speed and inexpensive method of interconnecting a variety of consumer electronics devices and personal computers. Apple Computer calls exactly the same thing Firewire® (developer.apple.com/firewire) and Sony Corporation (www.sony.net) calls it iLink®. The IEEE 1394 standard is mostly about the electrical specifications of the drivers, receivers and cables. Different camera manufacturers implement different higher-level protocols for controlling their cameras, see "How do I know if my camera is supported?".

Can I use it for my DV camcorder?

No, libdc1394 is not for camcorders. It is meant for uncompressed video cameras which are mostly used for scientific, industrial, microscopy, webcam and photography applications.

How do I know if my camera is supported?

libdc1394 supports cameras that are compliant with the IIDC 1394-Based Digital Camera Specifications (www.1394ta.org/Technology/Specifications/specifications.htm) also known as the DCAM spec. This is a CSR (control and status register) layer on top of IEEE 1394 and is used by many camera manufacturers. Check your camera user's manual or consult the excellent IEEE 1393 Digital Camera List (damien.douxchamps.net/ieee1394/cameras/).

Can I run more than one camera simultaneously?

Yes, as many 62 per interface card, subject to available IEEE 1394 bandwidth. The number 62 comes from 64 (the theoretical maximum) minus the interface card (which is also a device) and minus the reserved device 63 (which is a virtual broadcast device).

Can I use any IEEE 1394 host card?

The best set-up is for your host card to be OHCI (open host controller interface) compliant. The Adaptec AIC-5800 PCI-IEEE1394 and Texas Instruments PCILynx chips are also supported. You can get drivers (ohci1394, aic5800, pcilynx) from the IEEE 1394 for Linux homepage www.linux1394.org if they are not already in your distribution.

Can I install more than one IEEE 1394 host card?

Yes, as many as you have space for. Each card typically becomes a separate character device in /dev/video1394, numbered by its port number (for example /dev/video1394/0 for the first card). Host cards and device names are automatically dealt with internally so you normally don't have to care about that. Should you have strange device names, you may need to set the video1394 device name with dc1394_capture_set_dma_device_filename() before you call the capture setting function dc1394_capture_setup_dma().

Will libdc1394 work with my version of the kernel?

Kernel version 2.4.21 or later is strongly recommended. IEEE 1394 support in earlier kernel versions was incomplete and buggy. You are unlikely to get help from the mailing lists if you have an older kernel. Kernels 2.6.11 (included) to 2.6.16 (excluded) have a bug with timestamps. If you use timestamps you should avoid kernels in that range.

Where can I find more documentation?

- The README file and the source headers supplied with libdc1394, particularly the main include file dc1394/control.h. Seriously worth looking at first to get an idea of the defined enumerations, data structures, and library functions.
- There is a howto at www.tldp.org/HOWTO/libdc1394-HOWTO/.
- Support for Point Grey Research cameras at www.ptgrey.com/support/kb/ including some more generally useful information on using libdc1394.
- Technical papers at The Imaging Source www.1394imaging.com/en/resources/whitepapers/.
- The libdc1394 developers' list lists.sourceforge.net/lists/listinfo/libdc1394-devel.
- The archives of the IEEE users' mailing list lists.sourceforge.net/lists/listinfo/linux1394-user.
- The IIDC specifications can be found on this very site.

Where can I get the latest libdc1394?

From the libdc1394 project homepage sourceforge.net/projects/libdc1394/.

What else do I need apart from libdc1394?

For Linux, a number of kernel modules are required to run on your computer. (They can be of course compiled in the kernel instead of being run as modules but we just call them modules here.) These

modules are:

- ieee1394*: core of the IEEE 1394 subsystem
- raw1394*: higher-level module for bus access
- video1394: fast DMA frame transfer module
- a low-level host-card driver such as ohci1394 (see "Can I use any IEEE 1394 host card?" above)

* means that the module is probably included in your distribution. The technical documentation on the IEEE 1394 for Linux homepage www.linux1394.org/doc/overview.php has more detail. Libraw1394 must also be installed but without it libdc1394 will not compile anyway.

(TODO: comment on OSX requirements)

Is there example code to get me started?

- The basic example programs in the libdc1394 distribution.
- Coriander (damien.douxchamps.net/ieee1394/coriander).
  \
- Camwire (kauri.auck.irl.cri.nz/~johanns/camwire).
- Some examples tested on Point Grey Research cameras (http://www.ptgrey.com/support/kb/data/grabdma.tgz).

What is the difference between raw1394 and video1394 capture?

video1394 capture uses Direct Memory Access (DMA) to pipe camera data directly from the interface card to the computer memory, without copy or processor load (the data does not go through the processor at all.) This allows the maximal framerate to be reached in an efficient way. Video1394 is, in general, the capture mode that you should use.

libraw1394 uses libraw1394 to transfer images from the camera to a user-accessible buffer. This is not efficient because it requires copying data between internal and accessible buffers. This seriously reduces the maximum framerate and increases the processor load. In fact, due to other technical reasons the maximum framerate using libraw1394 is 50% of the framerate used by the camera: if the camera sends images at 15fps you will *never* reach more than 7.5fps [This may need confirmation]. For these reasons raw1394 access is only suitable for applications with limited requirements. It is however simpler to setup and does not rely on an extra module. As a rule of thumb, if you don't know exactly why you would use raw1394 capture then don't use raw1394 capture.

How does the DMA ring buffer work?

Once the camera is set up with dc1394_capture_setup() a contiguous block of memory-mapped frame buffers is waiting to be filled. It is organized as a ring buffer with each buffer internally set to a QUEUED state. Filling of the first buffer can start as soon as you start frame transmission with dc1394_video_set_transmission(,DC1394_ON). Each buffer is set to the READY state as soon as it is filled. Frame transmission continues until you call dc1394_video_set_transmission(,DC1394_OFF). If all of the buffers are filled during the capture then the capture stops (IOW you loose recent frames) until you stop transmission or make space by calling some capture functions.

The first time you call a DMA capture function dc1394_capture_dequeue() it returns a pointer to the first frame buffer structure (dc1394frame_t). Whether the function waits for a frame or returns immediately depends on whether the capturing is blocking or polling (see below). Each subsequent time you call a capture function it returns a pointer to the next frame in the DMA ring buffer (unless a polling call fails, see below). The accessed frame buffer is internally set to a FREE state.

After a successful capture function call, the capture_buffer pointer and the frame buffer it points to are available to you for reading and writing. It will not be overwritten with a newer frame while it is allocated to you (FREE), even if the ring buffer overflows. Once you have finished with it you should release it as soon as possible with a call to dc1394_capture_enqueue(). This resets the frame buffer to the QUEUED state so that it can again receive a new frame. You can dequeue several buffers at the same time, for example to look at their difference (change detection) but you have to release them otherwise there will be no buffers left for the DMA capture to write to. It is strongly advised to enqueue buffers in the same order as they have been dequeued. Tip: if you don't want to loose frames you should check that, in a loop, the number of buffers dequeued and enqueued are the same at each pass. If you cannot guarantee that you may run into problems. At last, enqueueing and dequeueing should take place in the same thread.

What is polling (non-blocking) capturing?

Polling applies only to the the dequeueing of DMA capture buffers via dc1394_capture_dequeue_dma(); it is an argument of that function. If you set the dc1394video_policy_t argument of this function to DC1394_VIDEO1394_POLL the function will *not* wait for frames if the DMA ring buffer is empty. It will return immediately either with the first frame waiting in the DMA ring buffer or with a NULL pointer if no frame is present.

What is non-polling (blocking) capturing?

Polling applies only to the the dequeueing of DMA capture buffers via dc1394_capture_dequeue_dma(); it is an argument of that function. If you set the dc1394video_policy_t argument of this function to DC1394_VIDEO1394_WAIT the function *will wait* for frames if the DMA ring buffer is empty. If a frame is not yet available (because the camera is waiting for an external trigger or if the frame rate is slow or if the latest frame is still being transmitted from the camera to the computer) then the blocking capture functions wait (block) until a frame is ready. There is no easy way to unblock this function other than stopping capture.

When should I use polling capture?

Polling (non-blocking) capture is useful if you need to check whether a frame is available in the DMA ring buffer but you don't want to wait if there is none. A typical example is flushing (capturing and discarding) frames from the DMA ring buffer until it is empty.

When should I use blocking capture?

Blocking (non-polling) capture is the one to use for most applications. It frees your program from worrying about whether a frame is available and automatically synchronizes your processing with the availability of frames. Your computer is not slowed down while a capture functions blocks on a frame. Other processes can continue while the capture process sleeps.

Why is DMA capture always one or more frames behind?

It depends what you mean by "behind". Somewhere in your initialization sequence you would call dc1394_video_set_transmission(). This causes the camera to start spewing out images until you tell it to stop with the same function. The capture functions can't start or stop the camera. The expected behaviour is for the first frame to be current and the next frame to be one frame time later, no matter when you actually call the capture function. The word "capture" in the names of the capture functions is misleading. The frame would have been captured by the camera at some earlier time. All the "capture" functions do is to request a pointer to the next waiting frame in the DMA ring buffer.

What happens when the DMA ring buffer overflows?

If the camera is running (ISO transmission is on) and you request a frame after a snooze, the DMA buffer is probably overflowing and throwing away the latest frames arriving from the camera. The frames you get from the capture functions will always be in chronological order but they may be old and/or irregularly spaced in time. Buffered (READY) frames in the DMA ring buffer can never be overwritten by newer frames.

How can I make absolutely sure I am getting the latest frame?

- Flush the DMA buffer and dequeue one fresh frame or
- Stop the camera with dc1394_stop_iso_transmission(), flush the DMA ring buffer (see below); use the one-shot function (see below). [is this OK for v2.x?]

If you need the very latest available frame every time you call a capture function, you must be sure to allocate a large enough number of DMA buffers and never allow them to fill up completely. See "What happens when the DMA ring buffer overflows?".

How do I capture just one fresh frame from the camera?

Use your camera's one-shot function: Stop the camera with dc1394_video_set_transmission(); flush unwanted frames from the DMA ring buffer if necessary; call dc1394_video_set_one_shot() and a non-polling capture method every time you need a new frame. Things to watch out for: The one-shot register is ignored if ISO transmission is on. The frame will take at least one frame period (as set in the camera) to be transmitted to your computer. Remember to always enqueue the DMA buffers that you have dequeued.

Don't I need to unset one-shot after using it?

No. The one-shot register in the camera self-clears after the camera has transmitted the frame to the computer. The only purpose of dc1394_video_set_one_shot(,DC1394_OFF) is to ensure that the camera does not transmit a spurious frame after it is stopped with dc1394_video_set_iso_transmission().

How do I know if my camera supports the one-shot function?

1. Read your camera's user's manual; or
2. Read the member dc1394camera_t->one_shot_capable; a value >0 means it is one shot capable

---

How do I set the frame rate?

For classic, non-scalable video modes the frame rate is set with dc1394_video_set_framerate(). Its frame_rate argument is an index to the actual frame rate: use the DC1394_FRAMERATE_xxxx enumeration in dc1394_control.h.

For scalable video modes (also known as Format_7 modes), the frame rate is set (believe it or not) by setting the IEEE 1394 packet size. dc1394_video_set_framerate() will *not* work. The packet size is one of the arguments ("bytes_per_packet") supplied to the Format_7 set-up functions like dc1394_format7_set_roi() or dc1394_format7_set_byte_per_packet(). See "How can I work out the packet size for a wanted frame rate?".

One consequence of this design is that it is not possible to change the frame rate in Format 7 without releasing the camera (with dc1394_capture_stop()) and setting it up again (dc1394_capture_setup[_dma]()). This a limitation of libdc1394, not the IIDC DCAM specification, and is quite computationally expensive especially when using DMA transfer.

In any Format, make sure that your integration time (shutter speed) is shorter (faster) than your frame period, otherwise frames will not be ready for transmission fast enough or your camera may misbehave.

Unfortunately, the term "frame rate" in IEEE 1394 transmission is misleading. It should really be called "transfer rate", because it is mostly determined by the IEEE 1394 packet size, not the rate at which frames arrive in the computer. Since exactly one packet is transmitted per camera per bus cycle, the transfer rate is determined by the number of bytes in each packet.

The transfer rate is meaningful even if the camera is using an external trigger or one-shot mode: it then determines the time it takes to transmit each frame. In this case you should make sure that the "frame rate" is faster than the fastest expected external trigger frequency or one-shot requests, otherwise the camera may produce frames faster than they are transmitted.

At last, a more accurate framerate can be obtained for both scalable and non-scalable video modes by using the frame rate feature that is specified in IIDC v1.31.

Why is my measured frame rate slower than the one I set?

Your camera is probably sending frames slower than you expect. A common cause is a shutter speed (integration time) slower than the reciprocal of the frame rate.

If the camera is set up to produce frames faster than the transfer rate (see "How do I set the frame rate?"), then the frame rate will be equal to the transfer rate. If the camera produces frames slower than the transfer rate, the frame rate will be determined by the camera, not the transfer rate.

At last, remember that raw1394 capture is significantly slower (50%).

How can I work out the packet size for a wanted frame rate?

This applies only to Format_7 (scalable image size). For classic non-scalable video modes the frame rate is set by dc1394_video_set_framerate(): see "How do I set the frame rate?".

It is a two-step calculation: first work out how many packets are transmitted per frame period, and then work out how big each packet has to be to contain the frame:

1. num_packets = (int) (1.0/(bus_period*frame_rate) + 0.5);
2. denominator = num_packets*8; packet_size = (width*height*depth + denominator - 1)/denominator;

where bus_period is listed in the table below.

The IIDC DCAM specification imposes some limitations on the allowed values of some of the variables:

- 0 < num_packets <= 4095
- packet_size = unit_bytes*n
- packet_size <= max_bytes

where n is a positive integer, and unit_bytes (also called min_bytes) and max_bytes are obtained from dc1394_query_format7_packet_para().

The function dc1394_format7_get_total_bytes() returns the needed frame size (width*height*depth)

observing these limitations. You may still prefer to work it out yourself because it has been suggested that different manufacturers interpret the meaning of the underlying camera register differently.

The IEEE 1394 bus cycle rate and period depends on the bus speed:

speed	period	rate
100 Mb/s	500 us	2000 /s
200 Mb/s	250 us	4000 /s
400 Mb/s	125 us	8000 /s
800 Mb/s	62.5 us	16000 /s

Note that "speed" does not equal "bandwidth". If you are interested in working out the bandwidth available for frame transmission from this table, keep in mind that 80% of the cycle period is reserved for isochronous transmission (image frames) and 20% for asynchronous transmission (camera commands).

In Format_7 the total number of packets needed per frame (num_packets) is calculated by the camera (for current settings of frame size and pixel depth) and can be read with dc1394_format7_get_packet_per_frame(). This is less useful than it sounds because to use it you first have to set up the camera with a dc1394_format7_set_roi() function which needs the packet size as an argument.

How can I work out the frame rate from the packet size?

This is a two-step calculation: first work out how many packets are required to transmit a frame, and then work out how long the transmission takes:

1. denominator = 8*packet_size; num_packets = width*height*depth + denominator - 1)/denominator;
2. frame_rate = 1.0/(bus_period*num_packets);

where bus_period is listed in the table in "How can I work out the packet size for a wanted frame rate?".

The IIDC DCAM specification imposes some limitations on the allowed values of some of the variables: see "How can I work out the packet size for a wanted frame rate?".

How do I minimize frame latency?

It depends what you mean. Some systems only need accurate synchronization of past events, in which case it does not matter how long it takes to capture an image as long as you have an accurate timestamp for it. In other systems you need to process the image as quickly as possible in which case you probably don't care much about the accuracy of the timestamp but want a quick answer.

For this discussion we can take latency to mean the time between the trigger instant and the instant the filltime is written into the dc1394frame_t structure at the end of transmission. Latency is dominated by:

1. shutter integration time
2. frame transmission time
3. computer scheduler (system tick) latency.

Delays within the camera are typically not significant compared to those. If you buffer many frames when using DMA transfer and your image processing falls behind the frame rate, that of course introduces another significant delay.

You can use (1) the shortest practical integration time.

You can get (2) the frame transmission time as low as possible by setting the number of packets per frame to a minimum (by setting the camera's frame rate as fast as possible: see "How do I set the frame rate?"). The fastest possible case has a single camera per IEEE 1394 host, using all the available bandwidth. Of course your computer may not appreciate frames repeatedly shoved down its throat at that fast rate, in which case you can either use an external trigger or one-shot to control the trigger instants, or manually empty the ring buffer before each capture to get only the freshest frame when you are ready to process it.

And (3), typical computer system ticks have a 10 millisecond period. Your frame capture process or thread may be denied processing time for several milliseconds at a time (sometimes much longer when other processes are busy) and there is not much you can do about it. You can use a real-time operating system to get control of the system tick, or you can mess with the configuration of your current OS. The Linux kernel can be configured to use various process prioritization strategies, for example.

Can I measure the actual frame latency?

It is possible to measure latency very accurately if your computer has a spare serial port, if you use DMA transfer, and your camera has an external trigger input. The latency typically comprises:

1. Trigger set-up time from signal edge until integration starts.
2. Integration time as set by dc1394_feature_set_value(,DC1394_FEATURE_SHUTTER,)
3. Sensor line transfer time
4. Transmission set-up time
5. Transmission time (number of packets times bus period: see "How can I work out the packet size for a wanted frame rate?")
6. Time it takes for the capturing process to wake up after for example blocking on waiting for a frame to be captured

The first five delays can be known from the camera's user's manual or by measurement or by calculation. The last delay (6) is harder to pin down although you could get an idea by collecting statistics of its average and maximum.

First check if your camera's external trigger input is compatible with the signal levels from your computer's serial port. It very likely is (most are opto-isolated) but you would want to minimize any risk to the camera. Connect the serial port data-terminal-ready (DTR) line (pin 4 on D-sub-9 connector, pin 20 on DB-25 connector) and the signal ground line (pin 5 on D-sub-9, pin 7 on DB-25) to your camera's external trigger input. Set the camera to use the external trigger, edge triggered, rising edge:

```
dc1394_external_trigger_set_mode(,DC1394_TRIGGER_MODE_0);
dc1394_external_trigger_set_power(,DC1394_ON);
dc1394_external_trigger_set_polarity(,DC1394_TRIGGER_ACTIVE_HIGH);
```

Set up the camera to capture 100 images at a frame rate faster than 10 fps and display their filltimes (from the dc1394frame_t structure). Compile and run the following little program to generate 100 trigger pulses on the serial port's DTR line and display their timestamps:

```
#include <stdio.h>

#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <time.h>
#include <sys/time.h>

/* Default trigger period */
```

```
#define T 0.1
/* Helper function */
static void setTrig(const int fd, const int level) {
    int sts;
    ioctl(fd,TIOCMGET,&sts);
    if (level) sts |= TIOCM_DTR;
    else       sts &= ~TIOCM_DTR;
    ioctl(fd,TIOCMSET,&sts);
}
int main(int argc, char *argv[]) {
    int fd,i;
    double T2=T/2,t1,t2;
    struct timespec halfT;
    struct timeval start,finish;
    fd = open("/dev/ttyS0",O_RDWR); /* serial port */
    /* Trigger half-period */
    if (argc>1) T2 = atof(argv[1])/2; /* optionally from cmd line */
    halfT.tv_sec = T2;
    halfT.tv_nsec = 1e9*(T2-(double)halfT.tv_sec);
    /* Generate 100 triggers */
    setTrig(fd,0);
    nanosleep(&halfT,0);
    for (i=0; i<100; ++i) {
        gettimeofday(&start,0);
        setTrig(fd,1); /* trigger */
        gettimeofday(&finish,0);
        t1 = start.tv_sec+1e-6*start.tv_usec;
        t2 = finish.tv_sec+1e-6*finish.tv_usec;
        printf("%.6f %.6f\n",(t1+t2)/2,(t2-t1)/2);
        fflush(stdout);
        nanosleep(&halfT,0);
        setTrig(fd,0); /* second half */
        nanosleep(&halfT,0);
    }
    close(fd);
    return 0;
}
```

Now you can compare the trigger timestamps to the corresponding filltimes. If you repeat this experiment with different image heights (number of lines) you should be able to work out parameters like sensor line transfer time (slope) and transmission set-up time (offset). Here are some typical numbers:

Parameter	Basler A602f	Basler A301fc	Note
Trigger set-up time	17 us	15 us	from user's manual
Integration time step	20 us	20 us	from user's manual
Sensor line transfer time	15.44 us per line	0 s per line	measured
Transmission set-up time	-20 us	695 us	measured
Transmission time	125 us times num_packets	125 us times num_packets	for 400 Mb/s

Some of these delays are not well documented in the manufacturer's manuals, or are inaccurately reported. It may be better to measure them yourself for your specific camera.

Can I find out exactly when a frame was acquired?

On return from a successful dc1394_capture_dequeue() function, the timestamp member of the dc1394frame_t structure contains the time when the frame's DMA buffer was filled, that is, when frame transmission was completed. On most systems it should be accurate to a few tens of microseconds. Its value is in microseconds since the Epoch.

If you are interested in the time the camera actually acquired the frame (the timestamp of the trigger or start of integration), you have to work backwards from the filltime by subtracting the delays mentioned in "Can I measure the actual frame latency?". Interfaces like Camwire can be configured to provide the time of the trigger signal as a frame timestamp.

Even if you have well-measured latency components, you do not have any control over the timing of the IEEE 1394 bus cycle. The camera has to wait for its isochronous time slot before the frame packets can hop on the bus. The upshot is that the frame timestamp has an inherent uncertainty of at least plus or minus half the IEEE 1394 bus period. Bus periods are listed in the table in "How can I work out the packet size for a wanted frame rate?".

How do I flush the DMA ring buffer?

Stop the camera ISO transmission (optional); wait (sleep) at least one frame period (optional, to be sure you have one frame); repeatedly call dc1394_capture_dequeue(,DC1394_VIDEO1394_POLL) until it returns NULL

Remember also to call dc1394_enqueue_buffer() after every successful call to a DMA capture function.

How do I ensure that I don't lose any frames?

Allocate a large DMA ring buffer (50 buffers) when you call the dc1394_capture_setup_dma() function, and make sure you call dequeue function faster than the camera frame rate. Note that it is not always possible to allocate 50 buffers if your images have a large resolution. Try to keep the total buffer size under a reasonable size. Kernel problems may occur over 32MB or 64MB.

How can I find out if a frame was dropped from not calling dc1394_capture_dequeue() frequently enough?

After each DMA capture function call, you can check the value returned in the frames_behind member of the dc1394frame_t structure. If frames_behind is equal to the ring buffer size minus one then you *may* have dropped frames.

If you have reason to believe that frames are transmitted regularly (for example on an external trigger signal) then you could also check the filltime member returned in dc1394frame_t for any irregularities in the time series.

Why do I get the error: "VIDEO1394_IOC_LISTEN_CHANNEL failed"?

You might get a message like "(capture.c) VIDEO1394_IOC_LISTEN_CHANNEL ioctl failed!" after your program was interrupted for any reason and did not close down video1394 properly. The reason is that video1394 is still listening on that channel and won't re-initialize.

The fix is to call ioctl(open(device,O_RDONLY),VIDEO1394_IOC_UNLISTEN_CHANNEL,&channel) where "device" is a char pointer your video1394 character device driver (by default "/dev/video1394/0" if your host card is port 0) and "channel" is an int variable initialized to the IEEE

1394 ISO channel previously allocated to your camera (usually numbered from 0 on each host. See "What is the ISO channel variable for?"). You could also try to do the same thing with dc1394_capture_stop() but then you will need the previous contents of the dc1394camera_t structure, which may no longer be available.

What is the ISO channel variable for?

As firewire is a shared bus, each camera must place a unique identifier in each packet so that the software can identify which camera a frame came from. The unique identifier used here is known as the ISO channel, and is just an integer. As well as writing the ISO channel number to the camera, you must also tell the video1394 layer which channel(s) it should listen for packets on.

The ISO channel allocation is automatic but you can also override it manually with dc1394_video_specify_iso_channel(). Should you choose to do it manually you should call the previous function before setting up the capture. You must first choose a unique channel number for the camera you are retrieving images from. If you are looping around an array of cameras, your loop variable "i" or "i+1" could be a suitable unique ISO channel identifier, or you could use the raw1394 (nodeid_t) node number as the channel number, which allows re-use of channel numbers on separate ports (host adaptors).

Is it safe to multi-thread with libdc1394?

Yes. (But see the "Can I detect another process using my camera?" too.)

Can I detect another process using my camera?

The IIDC DCAM specification (and hence libdc1394) does not provide a way of finding out if a camera is already in use by another process. When another process tries to use my camera (usually by calling one of the libdc1394 capture set-up functions), the camera is typically rendered useless for both processes. However, you may be able to find out if *any* camera is in use by checking if /dev/raw1394 or /dev/video1394 has been opened, with `/usr/sbin/lsof | grep 1394'.

What do the video1394 device files stand for?

Contrary to many beginners' idea, the video1394 device filenames (like /dev/video1394/0, /dev/video1394/1, etc...) *do not* correspond to a video channel to the camera. In other words, they are *not* related to ISO channels or to a specific camera on the bus. They are referring to IEEE1394 interface cards. Yes, that means the physical card (PCI or other) that you have inserted in your computer. For instance, if you have a laptop with an integrated 1394 port it will be assigned to /dev/video1394/0. This device filename will be used for all cameras attached to that port. Imagine that you want another port on your laptop. You insert a PCMCIA card and you now have two 1394 interfaces, hence two busses, hence /dev/video1394/0 and /dev/video1394/1 will refer to your integrated and PCMCIA cards. *Not respectively*: the device number depends on the order of the devices on the PCI bus (or other buses like PCIe, PCIx,...). If the PCMCIA port is probed before the integrated firewire of your laptop the device filenames will change (PCMCIA:0, integrated:1 while the integrated port had 0 when there was no PCMCIA card).

OK, as you can see things can be a bit complicated. But this is version 2 of the library so obviously there must be something prepared to help you, right? Well, yes there is something and it's rather simple: video1394 device filenames are handled automatically by libdc1394. That's it! You don't even have to think about it. However, some distributions may have strange device filenames, or you may have tweaked these yourself. In that case libdc1394 won't find those exotic device names and you have to specify them manually using dc1394_capture_set_dma_device_filename() *before setting up the capture*. This must only be performed once before the first capture setup. Video1394 device

filenames that are automatically supported are:

- /dev/video1394/x, where x is the interface number (this is the most common tree)
- /dev/video1394-x, where x is the interface number
- /dev/video1394, but only for the first interface (interface number 0). This is a legacy name.

Should I cleanup ISO channel and bandwidth? If so, when?

If your program always calls dc1394_capture_stop() on all capturing cameras before exiting, then you don't have to call this function. But if your program fails (segfault,...), is interrupted brutaly (CTRL-C without handling this event) or for some other reason does not calldc1394_capture_stop() as it should then you will have to cleanup the mess manually.

To do so you can use the function dc1394_cleanup_iso_channels_and_bandwidth(). This function clears all bandwidth booking and ISO channel allocation that was performed by the capture setup function. It should be called once for every camera whose capture was not properly terminated. Since it clears everything it does not matter if it is called from another program, after the camera has changed its video mode, etc... Since you normally don't know when your (buggy) program will fail you can't call this function right before the failure (that's obvious). So the only place where you can put it is at the beginning of your program, to ensure that it will start with a (very)clean and white slate.

Using this function may create serious problems if you use several programs capturing at the same time. In that case a first program failing will not have freed the allocated ressources, but calling the cleanup function will erase the allocations of the other programs that are still running. Hence the allocated ressources will not correspond to the ones that are used. Also, if all your programs start with the cleanup function then you can never have more than one program running at a time because the second instance will clear the ressources booked by the first program. There is currently no solution to that issue. Only a ressource manager at the kernel level could maybe handle these situations but there is no such thing in Linux (yet).

Sidenote: Coriander does perform a cleanup at boot time, at least in some 2.0.0 PRE or RC versions. This will be replaced by a button soon but in the meantime Coriander will *always* mess with ressource allocations.

How-when do I have to use dc1394_capture_set_dma_device_filename()?

This function should not be used "unless you know what you're doing". Libdc1394 v2.x will probe your system for the most common video1394 device filenames, namely /dev/video1394/X, /dev/video1394-X and /dev/video1394.

If the nqmes of the video1394 devices are different on your system you can use the function but you have to remember that the number of the video1394 device (e.g the 2 in /dev/video1394/2) refers to the PORT number, not the camera number. The port number can be found in the camera struct (camera->port). Thus, two cameras on a single 1394 card will share the same video1394 filename.

How-when do I have to use dc1394_cleanup_iso_channels_and_bandwidth?

Cleaning up iso channels and bandwidth is not necessary if you stop the capture properly with dc1394_capture_stop(). If your program fails (segfault,...) then you may have some problem because the ressources are not freed, resulting in a failure to setup the capture the next time you run your program (or sometimes a few segfaults later). You can then use the cleanup function at the beginning of your program to start with a clean state. BUT, this is dangerous if another program is using the camera at that time. Cleanup should only be used when no other programs using IIDC capture are running.

Do I have to start and stop ISO before and after each capture?

Oh no! Doing this will very likely result in a strong decrease in performance. Just start ISO transmission before and after your capture loop, but not in it. If you don't want the camera to stream data at all times it may be a better idea to use single shot mode.

Can I install and/or use libdc1394 v0.x, v1.x and v2.x at the same time on a single machine?

You can ***install*** all libdc1394 binaries (IOW .so, .a, .la and friends), system wide, on a single system. This is possible since all libraries have a different name (containing the release version). Moreover, version 0.x and 1.x use libdc1394_control.xx while version 2.x uses libdc1394.xx.

You can ***compile*** code using libdc1394 v2.x and code using [libdc1394 v0.x OR libdc1394 v1.x]. The version 2.x has different filenames for the include files ($PATH/include/dc1394/xxx.h) than the version 1.x or 0.x (which use the same $PATH/include/libdc1394/dc1394_xxxx.h). Since the include files are different they will not be overwritten when you install another version of the library.

You ***cannot compile*** code that uses libdc1394 v1.x at the same time as you compile code using libdc1394 v0.x (and conversely). Both versions of the library share the same names for the include files, hence the impossibility. This should not be a problem since v1.x is the continuation of v0.x and both are thus almost 100% compatible.

What is better: capturing in the main thread or in a separate threads? (MAC OSX)

David Moore wrote:

Capturing in the main thread has the advantage that you don't need to use synchronization primitives like mutexes or message queues to protect shared data structures in separate threads from race conditions. However, you have the disadvantage that if you perform long-running computations during callbacks, you will hurt the responsiveness of your GUI.

To do this with libdc1394 on Mac OS X, you would set up a callback function like this:

```
/* Our callback function */
static void FrameCallback (dc1394_camera_t * c, void * data)
{
    dc1394video_frame_t * frame;
    frame = dc1394_capture_dequeue_dma (c, DC1394_VIDEO1394_POLL);
    if (frame) {
            /* do something with the data here */

            dc1394_capture_enqueue_dma (c, frame);
    }
}
```

Then, this would go in your main application:

```
/* First, set up the camera parameters (video mode, ROI, etc.) */

/* Then, inform libdc1394 about your callback and runloop */
dc1394_capture_schedule_with_runloop (c, CFRunLoopGetCurrent (),
    kCFRunLoopDefaultMode);
dc1394_capture_set_callback (c, FrameCallback, NULL);
dc1394_capture_set_dma (c, 8);
dc1394_video_set_transmissions (c, DC1394_ON);
```

```
/* Run the main loop */
RunApplicationEventLoop();  /* or equivalent */

/* Clean up */
dc1394_video_set_transmission (c, DC1394_OFF);
dc1394_capture_stop (c);
```

During the call to RunApplicationEventLoop, you would get your GUI callbacks in addition to libdc1394 callbacks of the form FrameCallback(). The callback is called anytime there is a new frame available to be dequeued.

You can still use multiple threads if you want, but I like this event-driven style because of its reduced complexity.

Why do I see black images when capturing with an x86-64 box that has more than 3GB RAM? (LINUX)

It's difficult to explain why in details. There's a simple solution though: have a look at this thread which contains a kernel patch. This bug should be fixed soon after 2.6.20 is released.

How to cleanly stop a blocking capture? (LINUX)

When using the capture policy DC1394_CAPTURE_POLICY_WAIT, the capture function will lock the program (or thread) until a frame is grabbed. If the camera is not sending frames (e.g. no external trigger, etc...), then the progam will be locked. One solution is to use the polling policy DC1394_CAPTURE_POLICY_POLL but it may be impractical for other reasons.

If your kernel is new enough (2.6.20?), you can use the dc1394_capture_get_fileno() function to get a file descriptor for the capture process. After that, you can use the normal Unix mechanisms of select() and poll() to determine when it's time to call dc1394_capture_dequeue and know that it won't block.

If your kernel is too old, you will find that select() and poll() will always return immediately.

Thanks

A special thanks to Johann Schoonees for maintaining the FAQ until October 2006.
Contributors include: Rohit Agarwal, Vikram B, Arne Caspari, Renaud Dardenne, Dan Dennedy, Damien Douxchamps, Daniel Drake, David Moore, Don Murray, Stephan Richter and Olaf Ronneberger. Thanks to all and of course contributions are always welcome.