**RASPIVID e PICAMERA**

I must admit I wasn't aware of the 2Gb limitation in the stock build of raspivid (mentioned in Linus' answer). An alternative (if you don't fancy recompiling userland) would be to use [picamera](http://picamera.readthedocs.org/) (Python supports 64-bit file pointers out of the box). For example, the following should record a wide-screen 360p video in H.264 happily for 24 hours:

import picamera

with picamera.PiCamera() as camera:

camera.resolution = (640, 360)

camera.framerate = 24

camera.start\_recording('one\_day.h264')

camera.wait\_recording(24 \* 60 \* 60)

camera.stop\_recording()

The next part of the question is whether that will fit on a 64Gb SD card. My hunch is "probably", but let's verify that...

The Pi's H.264 encoder can be given a bitrate limit with the bitrate parameter in picamera's [start\_recording](http://picamera.readthedocs.org/en/release-1.9/api.html" \l "picamera.PiCamera.start_recording) method, or the --bitrate parameter in raspivid. In both raspivid and picamera this defaults to 17Mbps (megabits per second) so theoretically a 24-hour video recorded with the default setting couldn't be larger than:

24 hours

\* 60 minutes per hour

\* 60 seconds per minute

\* 17000000 bits per second

/ 8 bits per byte

/ 1073741824 bytes per gig

----------

170.990825 Gb

Hmm ... that's larger than I was expecting, but okay. One thing to bear in mind is that the default of 17Mbps is meant to be useful at the default recording resolution, which is full 1080p in the case of raspivid (although picamera defaults to the display resolution or 720p in the case of no display as that seemed "friendlier" when I wrote it). If you're only recording at 360p you can probably get away with a much lower bitrate limit.

The other thing to bear in mind is that the bitrate limit is just that: an upper limit. If the encoder doesn't need all 17 million bits to produce a good enough representation of one second's worth of motion, it won't use that many. By fiddling with the encoder's quantization (which is the qualityparameter in picamera, and the --qp parameter in raspivid) we can also adjust the encoder's idea of what "good enough" means. The quality is represented by a value between 0 and 40. Lower values mean better quality, so 1 is insanely good, and 40 is ridiculously bad. Typical "good enough" values are around 20-25. The value 0 (which is also the default) appears to be special; I'm not sure exactly what it means to the encoder (you'd have to ask the firmware devs that), but it seems to produce similar quality to values around 15-20 (i.e. very good).

So, assuming average quality (let's say 20), what sort of bitrate do we need to record wide-screen 360p video? I ran the following raspivid command line twice to record 30 seconds worth of video, then spent the first recording waving the camera around (under the assumption that more motion means more bandwidth required, and we want to test the limits here), and the second with the scene absolutely static:

raspivid --width 640 --height 360 --framerate 24 --bitrate 17000000 --qp 20 --timeout 30000 --output test.h264

The resulting files were 673675 bytes (658Kb) and 2804555 bytes (2.7Mb) in size respectively, so the bitrates produced by the encoder were:

673675 bytes

\* 8 bits per byte

/ 30 seconds

--------

179646.6 bits per second (static scene)

2804555 bytes

\* 8 bits per byte

/ 30 seconds

--------

747881.3 bits per second (full motion scene)

Hence, plugging those values into the equation above, we can realistically expect 24 hours worth of video using similar settings to come in somewhere between 1.8Gb and 7.5Gb in size. We can make sure it definitely won't be larger than that by setting bitrate to something like 750000 which we know will give the encoder enough room to reproduce motion at our desired quality (20), or you could experiment with lower qualities (e.g. 25) to see whether they would be acceptable, and then lower the bitrate limit accordingly. That said, it is worth bearing in mind that you're likely to break 2Gb with the file, so as mentioned above you're likely to run into the 64-bit file pointer issue unless you use Python or recompile userland.

For reference, here's the Python script from above modified to include the limits just discussed:

import picamera

with picamera.PiCamera() as camera:

camera.resolution = (640, 360)

camera.framerate = 24

camera.start\_recording('one\_day.h264', quality=20, bitrate=750000)

camera.wait\_recording(24 \* 60 \* 60)

camera.stop\_recording()

Finally, just to respond to goldilocks' comment on Linus' answer: splitting the video file into multiple parts is quite easy (and would easily alleviate any 64-bit file pointer concerns). With raspivid, you can use the --segment parameter to specify that a new file should be opened every *n* milliseconds, e.g. to record one file for every hour (the %02d in the filename will be substituted for a number, e.g. 01, 02, 03, ...):

raspivid --width 640 --height 360 --framerate 24 --bitrate 750000 --qp 20 --timeout $((24\*60\*60\*1000)) --segment $((1\*60\*60\*1000)) --output hour%02d.h264

Alternatively, with picamera you could use the [record\_sequence](http://picamera.readthedocs.org/en/release-1.9/api.html" \l "picamera.PiCamera.record_sequence) method to split based on time:

import picamera

with picamera.PiCamera() as camera:

camera.resolution = (640, 360)

camera.framerate = 24

for filename in camera.record\_sequence([

'hour%02d.h264' % (h + 1)

for h in range(24)

], quality=20, bitrate=750000):

camera.wait\_recording(60 \* 60)

Or based on file-size. In the example below, I've set it to produce 100 files rolling over once each reaches >1Mb, and placed the output iterator in its own function just to demonstrate it's possible to use infinite iterators with record\_sequence too:

import io

import itertools

import picamera

def outputs():

for i in itertools.count(1):

yield io.open('file%02d.h264' % i, 'wb')

with picamera.PiCamera() as camera:

camera.resolution = (640, 360)

camera.framerate = 24

for output in camera.record\_sequence(

outputs(), quality=20, bitrate=750000):

while output.tell() < 1048576:

camera.wait\_recording(0.1)

if output.name == 'file99.h264':

break

Or ... well, whatever limit you can think of the code for!