## VideoCapture::VideoCapture

VideoCapture constructors.

**C++:** `VideoCapture::`**`VideoCapture`**`()`

**C++:** `VideoCapture::`**`VideoCapture`**`(const string& `**`filename`**`)`

**C++:** `VideoCapture::`**`VideoCapture`**`(int `**`device`**`)`

**Python:** `cv2.`**`VideoCapture`**`()` → <VideoCapture object>

**Python:** `cv2.`**`VideoCapture`**`(filename)` → <VideoCapture object>

**Python:** `cv2.`**`VideoCapture`**`(device)` → <VideoCapture object>

**C:** CvCapture* **`cvCaptureFromCAM`**`(int `**`device`**`)`

**Python:** `cv.`**`CaptureFromCAM`**`(device)` → CvCapture

**C:** CvCapture* **`cvCaptureFromFile`**`(const char* `**`filename`**`)`

**Python:** `cv.`**`CaptureFromFile`**`(filename)` → CvCapture

**Parameters**

**filename** – name of the opened video file

**device** – id of the opened video capturing device (i.e. a camera index). If there is a single camera connected, just pass 0.

---

**Note:** In C API, when you finished working with video, release `CvCapture` structure with `cvReleaseCapture()`, or use `Ptr<CvCapture>` that calls `cvReleaseCapture()` automatically in the destructor.

---

## VideoCapture::open

Open video file or a capturing device for video capturing

**C++:** bool `VideoCapture::`**`open`**`(const string& `**`filename`**`)`

**C++:** bool `VideoCapture::`**`open`**`(int `**`device`**`)`

**Python:** `cv2.VideoCapture.`**`open`**`(filename)` → successFlag

**Python:** `cv2.VideoCapture.`**`open`**`(device)` → successFlag

**Parameters**

**filename** – name of the opened video file

**device** – id of the opened video capturing device (i.e. a camera index).

The methods first call `VideoCapture::release()` to close the already opened file or camera.

## VideoCapture::isOpened

Returns true if video capturing has been initialized already.

**C++:** bool `VideoCapture::`**`isOpened`**`()`

**Python:** `cv2.VideoCapture.`**`isOpened`**`()` → flag

If the previous call to `VideoCapture` constructor or `VideoCapture::open` succeeded, the method returns true.

## VideoCapture::release

Closes video file or capturing device.

**C++:** void `VideoCapture::`**`release`**`()`

**Python:** `cv2.VideoCapture.`**`release`**`()`

**C:** void **`cvReleaseCapture`**`(`CvCapture** **capture**`)`

The methods are automatically called by subsequent `VideoCapture::open()` and by `VideoCapture` destructor.

The C function also deallocates memory and clears `*capture` pointer.

## VideoCapture::grab

Grabs the next frame from video file or capturing device.

**C++:** bool `VideoCapture::`**`grab`**`()`

**Python:** `cv2.VideoCapture.`**`grab`**`()` → successFlag

**C:** int **`cvGrabFrame`**`(`CvCapture* **capture**`)`

**Python:** `cv.`**`GrabFrame`**`(`capture`)` → int

The methods/functions grab the next frame from video file or camera and return true (non-zero) in the case of success.

The primary use of the function is in multi-camera environments, especially when the cameras do not have hardware synchronization. That is, you call `VideoCapture::grab()` for each camera and after that call the slower method `VideoCapture::retrieve()` to decode and get frame from each camera. This way the overhead on demosaicing or motion jpeg decompression etc. is eliminated and the retrieved frames from different cameras will be closer in time.

Also, when a connected camera is multi-head (for example, a stereo camera or a Kinect device), the correct way of retrieving data from it is to call *VideoCapture::grab* first and then call `VideoCapture::retrieve()` one or more times with different values of the `channel` parameter. See http://code.opencv.org/svn/opencv/trunk/opencv/samples/cpp/kinect_maps.cpp

## VideoCapture::retrieve

Decodes and returns the grabbed video frame.

**C++:** bool `VideoCapture::`**`retrieve`**`(`Mat& **image**, int **channel**=0`)`

**Python:** `cv2.VideoCapture.`**`retrieve`**`(`[image[, channel]])` → successFlag, image

**C:** IplImage* **`cvRetrieveFrame`**`(`CvCapture* **capture**`)`

**Python:** `cv.`**`RetrieveFrame`**`(`capture`)` → iplimage

The methods/functions decode and return the just grabbed frame. If no frames has been grabbed (camera has been disconnected, or there are no more frames in video file), the methods return false and the functions return NULL pointer.

---

**Note:** OpenCV 1.x functions `cvRetrieveFrame` and `cv.RetrieveFrame` return image stored inside the video capturing structure. It is not allowed to modify or release the image! You can copy the frame using `cvCloneImage()` and then do whatever you want with the copy.

---

## VideoCapture::read

Grabs, decodes and returns the next video frame.

**C++:** VideoCapture& `VideoCapture::`**`operator>>`**(Mat& **image**)

**C++:** bool `VideoCapture::`**`read`**(Mat& **image**)

**Python:** `cv2.VideoCapture.`**`read`**([image]) → successFlag, image

**C:** IplImage* **`cvQueryFrame`**(CvCapture* **capture**)

**Python:** `cv.`**`QueryFrame`**(capture) → iplimage

The methods/functions combine `VideoCapture::grab()` and `VideoCapture::retrieve()` in one call. This is the most convenient method for reading video files or capturing data from decode and return the just grabbed frame. If no frames has been grabbed (camera has been disconnected, or there are no more frames in video file), the methods return false and the functions return NULL pointer.

---

**Note:** OpenCV 1.x functions `cvRetrieveFrame` and `cv.RetrieveFrame` return image stored inside the video capturing structure. It is not allowed to modify or release the image! You can copy the frame using `cvCloneImage()` and then do whatever you want with the copy.

---

## VideoCapture::get

Returns the specified `VideoCapture` property

**C++:** double `VideoCapture::`**`get`**(int **propId**)

**Python:** `cv2.VideoCapture.`**`get`**(propId) → retval

**C:** double **`cvGetCaptureProperty`**(CvCapture* **capture**, int **propId**)

**Python:** `cv.`**`GetCaptureProperty`**(capture, propId) → double

**Parameters**

**propId** – Property identifier. It can be one of the following:

- **CV_CAP_PROP_POS_MSEC** Current position of the video file in milliseconds or video capture timestamp.
- **CV_CAP_PROP_POS_FRAMES** 0-based index of the frame to be decoded/captured next.
- **CV_CAP_PROP_POS_AVI_RATIO** Relative position of the video file: 0 - start of the film, 1 - end of the film.
- **CV_CAP_PROP_FRAME_WIDTH** Width of the frames in the video stream.
- **CV_CAP_PROP_FRAME_HEIGHT** Height of the frames in the video stream.
- **CV_CAP_PROP_FPS** Frame rate.
- **CV_CAP_PROP_FOURCC** 4-character code of codec.
- **CV_CAP_PROP_FRAME_COUNT** Number of frames in the video file.
- **CV_CAP_PROP_FORMAT** Format of the Mat objects returned by `retrieve()` .
- **CV_CAP_PROP_MODE** Backend-specific value indicating the current capture mode.
- **CV_CAP_PROP_BRIGHTNESS** Brightness of the image (only for cameras).

- **CV_CAP_PROP_CONTRAST** Contrast of the image (only for cameras).
- **CV_CAP_PROP_SATURATION** Saturation of the image (only for cameras).
- **CV_CAP_PROP_HUE** Hue of the image (only for cameras).
- **CV_CAP_PROP_GAIN** Gain of the image (only for cameras).
- **CV_CAP_PROP_EXPOSURE** Exposure (only for cameras).
- **CV_CAP_PROP_CONVERT_RGB** Boolean flags indicating whether images should be converted to RGB.
- **CV_CAP_PROP_WHITE_BALANCE** Currently not supported
- **CV_CAP_PROP_RECTIFICATION** Rectification flag for stereo cameras (note: only supported by DC1394 v 2.x backend currently)

**Note**: When querying a property that is not supported by the backend used by the `VideoCapture` class, value 0 is returned.

## VideoCapture::set

Sets a property in the `VideoCapture`.

**C++:** bool `VideoCapture::set`(int **propertyId**, double **value**)

**Python:** `cv2.VideoCapture.set`(propId, value) → retval

**C:** int **cvSetCaptureProperty**(CvCapture* **capture**, int **propId**, double **value**)

**Python:** `cv.SetCaptureProperty`(capture, propId, value) → None

**Parameters**

**propId** – Property identifier. It can be one of the following:

- **CV_CAP_PROP_POS_MSEC** Current position of the video file in milliseconds.
- **CV_CAP_PROP_POS_FRAMES** 0-based index of the frame to be decoded/captured next.
- **CV_CAP_PROP_POS_AVI_RATIO** Relative position of the video file: 0 - start of the film, 1 - end of the film.
- **CV_CAP_PROP_FRAME_WIDTH** Width of the frames in the video stream.
- **CV_CAP_PROP_FRAME_HEIGHT** Height of the frames in the video stream.
- **CV_CAP_PROP_FPS** Frame rate.
- **CV_CAP_PROP_FOURCC** 4-character code of codec.
- **CV_CAP_PROP_FRAME_COUNT** Number of frames in the video file.
- **CV_CAP_PROP_FORMAT** Format of the Mat objects returned by `retrieve()` .
- **CV_CAP_PROP_MODE** Backend-specific value indicating the current capture mode.
- **CV_CAP_PROP_BRIGHTNESS** Brightness of the image (only for cameras).
- **CV_CAP_PROP_CONTRAST** Contrast of the image (only for cameras).
- **CV_CAP_PROP_SATURATION** Saturation of the image (only for cameras).
- **CV_CAP_PROP_HUE** Hue of the image (only for cameras).
- **CV_CAP_PROP_GAIN** Gain of the image (only for cameras).

- **CV_CAP_PROP_EXPOSURE** Exposure (only for cameras).
- **CV_CAP_PROP_CONVERT_RGB** Boolean flags indicating whether images should be converted to RGB.
- **CV_CAP_PROP_WHITE_BALANCE** Currently unsupported
- **CV_CAP_PROP_RECTIFICATION** Rectification flag for stereo cameras (note: only supported by DC1394 v 2.x backend currently)

**value** – Value of the property.

## VideoWriter

**class `VideoWriter`**

Video writer class.

## VideoWriter::VideoWriter

VideoWriter constructors

**C++:** `VideoWriter::VideoWriter()`

**C++:** `VideoWriter::VideoWriter(const string& filename, int fourcc, double fps, Size frameSize, bool isColor=true)`

**Python:** `cv2.VideoWriter([filename, fourcc, fps, frameSize[, isColor]]) → <VideoWriter object>`

**C:** CvVideoWriter* `cvCreateVideoWriter(const char* filename, int fourcc, double fps, CvSize frameSize, int isColor=1 )`

**Python:** `cv.CreateVideoWriter(filename, fourcc, fps, frameSize, isColor) → CvVideoWriter`

**Python:** `cv2.VideoWriter.isOpened() → retval`

**Python:** `cv2.VideoWriter.open(filename, fourcc, fps, frameSize[, isColor]) → retval`

**Python:** `cv2.VideoWriter.write(image) → None`

**Parameters**

**filename** – Name of the output video file.

**fourcc** – 4-character code of codec used to compress the frames. For example, `CV_FOURCC('P','I','M','1')` is a MPEG-1 codec, `CV_FOURCC('M','J','P','G')` is a motion-jpeg codec etc.

**fps** – Framerate of the created video stream.

**frameSize** – Size of the video frames.

**isColor** – If it is not zero, the encoder will expect and encode color frames, otherwise it will work with grayscale frames (the flag is currently supported on Windows only).

The constructors/functions initialize video writers. On Linux FFMPEG is used to write videos; on Windows FFMPEG or VFW is used; on MacOSX QTKit is used.