. Documentation

Release master

Author

Mar 13, 2021

ROBOT PROGRAMMING

This is documentation for the python bindings for cscore, a high performance camera access and streaming library introduced by FIRST in 2017. It can be used to:

- Stream a USB/HTTP camera to SmartDashboard or the LabVIEW dashboard via HTTP
- Capture images from USB or HTTP camera, modify them using OpenCV/Numpy, and send them via HTTP to SmartDashboard, the LabVIEW dashboard, or a web browser.

CHAPTER

ONE

ROBOTPY CSCORE

1.1 CameraServer

class cscore.**CameraServer**

Singleton class for creating and keeping camera servers.

This is a higher level wrapper around the cscore functionality, and also publishes camera information to NetworkTables so that dashboards can easily find and display the camera streams.

addAxisCamera(*host*, *name='Axis Camera'*)

Adds an Axis IP camera.

Parameters

- **host** (Union[str, Sequence[str]]) – Camera host IP/DNS name or list of camera host IPs/DNS names
- **name** (str) – The name to give the camera (optional)

Return type *AxisCamera*

Returns Axis camera object

addCamera(*camera*)

Adds an already created camera.

Parameters **camera** (*VideoSource*) – Camera object

Return type None

addServer(*, *name: str*, *port: Optional[int] = 'None'*) → *cscore.MjpegServer*
addServer(*, *server:* cscore.VideoSink) → *cscore.MjpegServer*

Adds a MJPEG server

Parameters

- **name** – Server name
- **port** (Optional[int]) – Port of server (if None, use next available port)
- **server** – Server

Return type *MjpegServer*

Returns server object

All arguments must be specified as keyword arguments. The following combinations are accepted:

- name
- name, port

- server

addSwitchedCamera(*name*)
Adds a virtual camera for switching between two streams. Unlike the other addCamera methods, this returns a VideoSink rather than a VideoSource. Calling setSource() on the returned object can be used to switch the actual source of the stream.

Parameters **name** (str) – Name of camera

Return type *MjpegServer*

Returns Server object

static enableLogging(*level=20*)

classmethod getInstance()
Get the CameraServer instance.

Return type CameraServer

getServer(*name=None*)
Get server by name, or for the primary camera feed if no name is specified.

This is only valid to call after a camera feed has been added with *startAutomaticCapture()* or *addServer()*.

Parameters **name** (Optional[str]) – Server name

Return type *VideoSource*

Returns server object

getVideo() → *cscore.CvSink*
getVideo(*, *name: str*) → *cscore.CvSink*
getVideo(*, *camera:* cscore.VideoSource) → *cscore.CvSink*
Get OpenCV access to specified camera. This allows you to get images from the camera for image processing.

Parameters

- **name** – Name of camera to retrieve video for
- **camera** – Camera object

Return type *CvSink*

Returns CvSink object corresponding to camera

All arguments must be specified as keyword arguments. The following combinations are permitted:

- (no args)
- name
- camera

If there are no arguments, then this will retrieve access to the primary camera. No arguments will fail if a camera feed has not already been added via *startAutomaticCapture()* or *addCamera()*

kBasePort = 1181

kPublishName = '/CameraPublisher'

putVideo(*name*, *width*, *height*)
Create a MJPEG stream with OpenCV input. This can be called to pass custom annotated images to the dashboard.

Parameters

- **name** (str) – Name to give the stream
- **width** (int) – Width of the image being sent
- **height** (int) – Height of the image being sent

Return type *CvSource*

Returns CvSource object that you can publish images to

removeCamera(*name*)
Removes a camera by name.

Parameters **name** (str) – Camera name

Return type None

removeServer(*name*)
Removes a server by name.

Parameters **name** (str) – Server name

Return type None

startAutomaticCapture(*, *return_server: bool = 'False'*) → Union[*cscore.UsbCamera*, *cscore.MjpegServer*]
startAutomaticCapture(*, *dev: int*, *name: Optional[str] = 'None'*, *return_server: bool = 'False'*) → Union[*cscore.UsbCamera*, *cscore.MjpegServer*]
startAutomaticCapture(*, *name: str*, *path: str*, *return_server: bool = 'False'*) → Union[*cscore.UsbCamera*, *cscore.MjpegServer*]
startAutomaticCapture(*, *camera:* cscore.VideoSource, *return_server: bool = 'False'*) → Union[*cscore.VideoSource*, *cscore.MjpegServer*]
Start automatically capturing images to send to the dashboard.

You should call this method to see a camera feed on the dashboard. If you also want to perform vision processing on the roboRIO, use *getVideo()* to get access to the camera images.

Parameters

- **dev** – If specified, the device number to use
- **name** – If specified, the name to use for the camera (dev must be specified)
- **path** – If specified, device path (e.g. "/dev/video0") of the camera
- **camera** – If specified, an existing camera object to use
- **return_server** – If specified, return the server instead of the camera

Returns USB Camera object, or the camera argument, or the created server

Return type *VideoSource* or *MjpegServer*

The following argument combinations are accepted – all arguments must be specified as keyword arguments:

- (no args)
- dev
- dev, name
- name, path
- camera

The first time this is called with no arguments, a USB Camera from device 0 is created. Subsequent calls increment the device number (e.g. 1, 2, etc).

Note: USB Cameras are not available on all platforms. If it is not available on your platform, *VideoException* is thrown

waitForever()
Infinitely loops until the process dies

class cscore.**VideoException**

1.2 Objects

1.2.1 AxisCamera

class cscore.**AxisCamera**(**args*, ***kwargs*)
Bases: *cscore.HttpCamera*

A source that represents an Axis IP camera.

Overloaded function.

1. __init__(name: str, host: str) -> None

Create a source for a MJPEG-over-HTTP (IP) camera.

Parameters

- **name** – Source name (arbitrary unique identifier)
- **host** – Camera host IP or DNS name (e.g. "10.x.x.11")
- **kind** – Camera kind (e.g. kAxis)

2. __init__(name: str, hosts: List[str]) -> None

Create a source for a MJPEG-over-HTTP (IP) camera.

Parameters

- **name** – Source name (arbitrary unique identifier)
- **hosts** – Array of Camera host IPs/DNS names
- **kind** – Camera kind (e.g. kAxis)

1.2.2 CvSink

class cscore.**CvSink**(*name: str*)
Bases: *cscore.ImageSink*

A sink for user code to accept video frames as OpenCV images.

Create a sink for accepting OpenCV images. *grabFrame()* must be called on the created sink to get each new image

Parameters **name** – Source name (arbitrary unique identifier)

grabFrame (*image: numpy.ndarray*, *timeout: float = 0.225*) → Tuple[int, numpy.ndarray]
Wait for the next frame and get the image. Times out (returning 0) after timeout seconds. The provided image will have three 8-bit channels stored in BGR order.

Returns Frame time, or 0 on error (call `getError()` to obtain the error message), returned image The frame time is in 1us increments

grabFrameNoTimeout (*image: numpy.ndarray*) → Tuple[int, numpy.ndarray]
Wait for the next frame and get the image. May block forever. The provided image will have three 8-bit channels stored in BGR order.

Returns Frame time, or 0 on error (call `getError()` to obtain the error message), returned image The frame time is in 1us increments

setDescription (*description: str*) → None
Set sink description.

Parameters **description** – Description

1.2.3 CvSource

class cscore.**CvSource** (**args*, ***kwargs*)
Bases: *cscore.ImageSource*

A source for user code to provide OpenCV images as video frames.

Overloaded function.

1. __init__(name: str, mode: cscore.VideoMode) -> None

Create an OpenCV source.

Parameters

- **name** – Source name (arbitrary unique identifier)
- **mode** – Video mode being generated

2. __init__(name: str, pixelFormat: cscore.VideoMode.PixelFormat, width: int, height: int, fps: int) -> None

Create an OpenCV source.

Parameters

- **name** – Source name (arbitrary unique identifier)
- **pixelFormat** – Pixel format
- **width** – width
- **height** – height
- **fps** – fps

putFrame (*image: numpy.ndarray*) → None
Put an OpenCV image and notify sinks.

Only 8-bit single-channel or 3-channel (with BGR channel order) images are supported. If the format, depth or channel order is different, use `cv2.convertTo()` and/or `cv2.cvtColor()` to convert it first.

Parameters **image** – OpenCV image

1.2.4 HttpCamera

class cscore.**HttpCamera**(**args*, ***kwargs*)
Bases: *cscore.VideoCamera*

A source that represents a MJPEG-over-HTTP (IP) camera.

Overloaded function.

1. __init__(name: str, url: str, kind: cscore.HttpCamera.HttpCameraKind = <HttpCameraKind.kUnknown: 0>) -> None

Create a source for a MJPEG-over-HTTP (IP) camera.

Parameters

- **name** – Source name (arbitrary unique identifier)
- **url** – Camera URL (e.g. "http://10.x.y.11/video/stream.mjpg")
- **kind** – Camera kind (e.g. kAxis)

2. __init__(name: str, urls: List[str], kind: cscore.HttpCamera.HttpCameraKind = <HttpCameraKind.kUnknown: 0>) -> None

Create a source for a MJPEG-over-HTTP (IP) camera.

Parameters

- **name** – Source name (arbitrary unique identifier)
- **urls** – Array of Camera URLs
- **kind** – Camera kind (e.g. kAxis)

class HttpCameraKind(*value: int*) → None
Bases: pybind11_builtins.pybind11_object

Members:

kUnknown

kMJPGStreamer

kCSCore

kAxis

kAxis = <HttpCameraKind.kAxis: 3>

kCSCore = <HttpCameraKind.kCSCore: 2>

kMJPGStreamer = <HttpCameraKind.kMJPGStreamer: 1>

kUnknown = <HttpCameraKind.kUnknown: 0>

property name

getHttpCameraKind() → *cscore.HttpCamera.HttpCameraKind*
Get the kind of HTTP camera. Autodetection can result in returning a different value than the camera was created with.

getUrls() → List[str]
Get the URLs used to connect to the camera.

setUrls(*urls: List[str]*) → None
Change the URLs used to connect to the camera.

1.2.5 ImageSink

class cscore.**ImageSink**
Bases: *cscore.VideoSink*

A base class for single image reading sinks.

getError() → str
Get error string. Call this if `waitForFrame()` returns 0 to determine what the error is.

setDescription(*description: str*) → None
Set sink description.

Parameters **description** – Description

setEnabled(*enabled: bool*) → None
Enable or disable getting new frames. Disabling will cause processFrame (for callback-based ImageSinks) to not be called and `waitForFrame()` to not return. This can be used to save processor resources when frames are not needed.

1.2.6 ImageSource

class cscore.**ImageSource**
Bases: *cscore.VideoSource*

A base class for single image providing sources.

createBooleanProperty(*name: str*, *defaultValue: bool*, *value: bool*) → *cscore.VideoProperty*
Create a property.

Parameters

- **name** – Property name
- **defaultValue** – Default value
- **value** – Current value

Returns Property

createIntegerProperty(*name: str*, *minimum: int*, *maximum: int*, *step: int*, *defaultValue: int*, *value: int*) → *cscore.VideoProperty*
Create a property.

Parameters

- **name** – Property name
- **minimum** – Minimum value
- **maximum** – Maximum value
- **step** – Step value
- **defaultValue** – Default value
- **value** – Current value

Returns Property

createProperty(*name: str*, *kind:* cscore.VideoProperty.Kind, *minimum: int*, *maximum: int*, *step: int*, *defaultValue: int*, *value: int*) → *cscore.VideoProperty*
Create a property.

Parameters

- **name** – Property name
- **kind** – Property kind
- **minimum** – Minimum value
- **maximum** – Maximum value
- **step** – Step value
- **defaultValue** – Default value
- **value** – Current value

Returns Property

createStringProperty(*name: str*, *value: str*) → *cscore.VideoProperty*
Create a property.

Parameters

- **name** – Property name
- **value** – Current value

Returns Property

notifyError(*msg: str*) → None
Signal sinks that an error has occurred. This should be called instead of `notifyFrame()` when an error occurs.

setConnected(*connected: bool*) → None
Set source connection status. Defaults to true.

Parameters **connected** – True for connected, false for disconnected

setDescription(*description: str*) → None
Set source description.

Parameters **description** – Description

setEnumPropertyChoices(*property:* cscore.VideoProperty, *choices: List[str]*) → None
Configure enum property choices.

Parameters

- **property** – Property
- **choices** – Choices

1.2.7 MjpegServer

class cscore.**MjpegServer**(**args*, ***kwargs*)
Bases: *cscore.VideoSink*

A sink that acts as a MJPEG-over-HTTP network server.

Overloaded function.

1. __init__(name: str, listenAddress: str, port: int) -> None

Create a MJPEG-over-HTTP server sink.

Parameters

- **name** – Sink name (arbitrary unique identifier)

- **listenAddress** – TCP listen address (empty string for all addresses)
- **port** – TCP port number

2. __init__(name: str, port: int) -> None

Create a MJPEG-over-HTTP server sink.

Parameters

- **name** – Sink name (arbitrary unique identifier)
- **port** – TCP port number

getListenAddress() → str
Get the listen address of the server.

getPort() → int
Get the port number of the server.

setCompression(*quality: int*) → None
Set the compression for clients that don't specify it.

Setting this will result in increased CPU usage for MJPEG source cameras as it will decompress and recompress the image instead of using the camera's MJPEG image directly.

Parameters **quality** – JPEG compression quality (0-100), -1 for unspecified

setDefaultCompression(*quality: int*) → None
Set the default compression used for non-MJPEG sources. If not set, 80 is used. This function has no effect on MJPEG source cameras; use setCompression() instead to force recompression of MJPEG source images.

Parameters **quality** – JPEG compression quality (0-100)

setFPS(*fps: int*) → None
Set the stream frames per second (FPS) for clients that don't specify it.

It is not necessary to set this if it is the same as the source FPS.

Parameters **fps** – FPS, 0 for unspecified

setResolution(*width: int*, *height: int*) → None
Set the stream resolution for clients that don't specify it.

It is not necessary to set this if it is the same as the source resolution.

Setting this different than the source resolution will result in increased CPU usage, particularly for MJPEG source cameras, as it will decompress, resize, and recompress the image, instead of using the camera's MJPEG image directly.

Parameters

- **width** – width, 0 for unspecified
- **height** – height, 0 for unspecified

1.2.8 RawEvent

class cscore.**RawEvent**

Listener event

class Kind(*value: int*) → None

Members:

kSourceCreated

kSourceDestroyed

kSourceConnected

kSourceDisconnected

kSourceVideoModesUpdated

kSourceVideoModeChanged

kSourcePropertyCreated

kSourcePropertyValueUpdated

kSourcePropertyChoicesUpdated

kSinkSourceChanged

kSinkCreated

kSinkDestroyed

kSinkEnabled

kSinkDisabled

kNetworkInterfacesChanged

kTelemetryUpdated

kSinkPropertyCreated

kSinkPropertyValueUpdated

kSinkPropertyChoicesUpdated

kNetworkInterfacesChanged = <Kind.kNetworkInterfacesChanged: 16384>

kSinkCreated = <Kind.kSinkCreated: 1024>

kSinkDestroyed = <Kind.kSinkDestroyed: 2048>

kSinkDisabled = <Kind.kSinkDisabled: 8192>

kSinkEnabled = <Kind.kSinkEnabled: 4096>

kSinkPropertyChoicesUpdated = <Kind.kSinkPropertyChoicesUpdated: 262144>

kSinkPropertyCreated = <Kind.kSinkPropertyCreated: 65536>

kSinkPropertyValueUpdated = <Kind.kSinkPropertyValueUpdated: 131072>

kSinkSourceChanged = <Kind.kSinkSourceChanged: 512>

kSourceConnected = <Kind.kSourceConnected: 4>

kSourceCreated = <Kind.kSourceCreated: 1>

kSourceDestroyed = <Kind.kSourceDestroyed: 2>

kSourceDisconnected = <Kind.kSourceDisconnected: 8>

kSourcePropertyChoicesUpdated = <Kind.kSourcePropertyChoicesUpdated: 256>

kSourcePropertyCreated = <Kind.kSourcePropertyCreated: 64>

kSourcePropertyValueUpdated = <Kind.kSourcePropertyValueUpdated: 128>

kSourceVideoModeChanged = <Kind.kSourceVideoModeChanged: 32>

kSourceVideoModesUpdated = <Kind.kSourceVideoModesUpdated: 16>

kTelemetryUpdated = <Kind.kTelemetryUpdated: 32768>

property name

property kind

property mode

property name

property sinkHandle

property sourceHandle

property value

property valueStr

1.2.9 UsbCamera

class cscore.**UsbCamera**(**args*, ***kwargs*)

Bases: *cscore.VideoCamera*

A source that represents a USB camera.

Overloaded function.

1. __init__(name: str, dev: int) -> None

Create a source for a USB camera based on device number.

Parameters

- **name** – Source name (arbitrary unique identifier)
- **dev** – Device number (e.g. 0 for `/dev/video0`)

2. __init__(name: str, path: str) -> None

Create a source for a USB camera based on device path.

Parameters

- **name** – Source name (arbitrary unique identifier)
- **path** – Path to device (e.g. `/dev/video0` on Linux)

static enumerateUsbCameras() → List[*cscore.UsbCameraInfo*]

Enumerate USB cameras on the local system.

Returns list of USB camera information (one for each camera)

getInfo() → *cscore.UsbCameraInfo*

Get the full camera information for the device.

getPath() → str
Get the path to the device.

setConnectVerbose(*level: int*) → None
Set how verbose the camera connection messages are.

Parameters **level** – 0=don't display Connecting message, 1=do display message

setPath(*path: str*) → None
Change the path to the device.

1.2.10 UsbCameraInfo

class cscore.**UsbCameraInfo**
USB camera information

property dev

property name

property otherPaths

property path

property productId

property vendorId

1.2.11 VideoCamera

class cscore.**VideoCamera**
Bases: *cscore.VideoSource*

A source that represents a video camera.

class WhiteBalance(*value: int*) → None
Bases: pybind11_builtins.pybind11_object

Members:

kFixedIndoor

kFixedOutdoor1

kFixedOutdoor2

kFixedFluorescent1

kFixedFlourescent2

kFixedFlourescent2 = <WhiteBalance.kFixedFlourescent2: 5200>

kFixedFluorescent1 = <WhiteBalance.kFixedFluorescent1: 5100>

kFixedIndoor = <WhiteBalance.kFixedIndoor: 3000>

kFixedOutdoor1 = <WhiteBalance.kFixedOutdoor1: 4000>

kFixedOutdoor2 = <WhiteBalance.kFixedOutdoor2: 5000>

property name

getBrightness() → int
Get the brightness, as a percentage (0-100).

setBrightness(*brightness: int*) → None
Set the brightness, as a percentage (0-100).

setExposureAuto() → None
Set the exposure to auto aperature.

setExposureHoldCurrent() → None
Set the exposure to hold current.

setExposureManual(*value: int*) → None
Set the exposure to manual, as a percentage (0-100).

setWhiteBalanceAuto() → None
Set the white balance to auto.

setWhiteBalanceHoldCurrent() → None
Set the white balance to hold current.

setWhiteBalanceManual(*value: int*) → None
Set the white balance to manual, with specified color temperature.

1.2.12 VideoEvent

class cscore.**VideoEvent**
Bases: *cscore.RawEvent*

An event generated by the library and provided to event listeners.

getProperty() → *cscore.VideoProperty*

getSink() → *cscore.VideoSink*

getSource() → *cscore.VideoSource*

1.2.13 VideoListener

class cscore.**VideoListener**(*callback: Callable[[*cscore.VideoEvent*], None]*, *eventMask: int*, *immediateNotify: bool*)
An event listener. This calls back to a desigated callback function when an event matching the specified mask is generated by the library.

Create an event listener.

Parameters

- **callback** – Callback function
- **eventMask** – Bitmask of VideoEvent.Kind values
- **immediateNotify** – Whether callback should be immediately called with a representative set of events for the current library state.

1.2.14 VideoMode

class cscore.**VideoMode**(*pixelFormat: cscore.VideoMode.PixelFormat*, *width: int*, *height: int*, *fps: int*)

Video mode

class PixelFormat(*value: int*) → None

Members:

kUnknown

kMJPEG

kYUYV

kRGB565

kBGR

kGray

kBGR = <PixelFormat.kBGR: 4>

kGray = <PixelFormat.kGray: 5>

kMJPEG = <PixelFormat.kMJPEG: 1>

kRGB565 = <PixelFormat.kRGB565: 3>

kUnknown = <PixelFormat.kUnknown: 0>

kYUYV = <PixelFormat.kYUYV: 2>

property name

property fps

property height

property pixelFormat

property width

1.2.15 VideoProperty

class cscore.**VideoProperty**

A source or sink property.

class Kind(*value: int*) → None

Members:

kNone

kBoolean

kInteger

kString

kEnum

kBoolean = <Kind.kBoolean: 1>

kEnum = <Kind.kEnum: 8>

kInteger = <Kind.kInteger: 2>

kNone = <Kind.kNone: 0>

kString = <Kind.kString: 4>

property name

get() → int

getChoices() → List[str]

getDefault() → int

getKind() → *cscore.VideoProperty.Kind*

getLastStatus() → int

getMax() → int

getMin() → int

getName() → str

getStep() → int

getString() → str

isBoolean() → bool

isEnum() → bool

isInteger() → bool

isString() → bool

set(*value: int*) → None

setString(*value: str*) → None

1.2.16 VideoSink

class cscore.**VideoSink**(*sink:* cscore.VideoSink)

A sink for video that accepts a sequence of frames.

class Kind(*value: int*) → None

Members:

kUnknown

kMjpeg

kCv

kCv = <Kind.kCv: 4>

kMjpeg = <Kind.kMjpeg: 2>

kUnknown = <Kind.kUnknown: 0>

property name

enumerateProperties() → List[*cscore.VideoProperty*]

Enumerate all properties of this sink

static enumerateSinks() → List[*cscore.VideoSink*]

Enumerate all existing sinks.

Returns list of sinks.

getConfigJson() → str
Get a JSON configuration string.

Returns JSON configuration string

getDescription() → str
Get the sink description. This is sink-kind specific.

getHandle() → int

getKind() → *cscore.VideoSink.Kind*
Get the kind of the sink.

getLastStatus() → int

getName() → str
Get the name of the sink. The name is an arbitrary identifier provided when the sink is created, and should be unique.

getProperty(*name: str*) → *cscore.VideoProperty*
Get a property.

Parameters **name** – Property name

Returns Property contents (VideoSource.Kind.kNone if no property with the given name exists)

getSource() → *cscore.VideoSource*
Get the connected source.

Returns Connected source (empty if none connected).

getSourceProperty(*name: str*) → *cscore.VideoProperty*
Get a property of the associated source.

Parameters **name** – Property name

Returns Property (VideoSink.Kind.kNone if no property with the given name exists or no source connected)

setConfigJson(*config: str*) → bool
Set properties from a JSON configuration string.

The format of the JSON input is:

```
{
  "properties": [
    {
      "name": "property name",
      "value": "property value"
    }
  ]
}
```

Parameters **config** – configuration

Returns True if set successfully

setSource(*source:* cscore.VideoSource) → None
Configure which source should provide frames to this sink. Each sink can accept frames from only a single source, but a single source can provide frames to multiple clients.

Parameters **source** – Source

1.2.17 VideoSource

class cscore.**VideoSource**(*source:* cscore.VideoSource)

A source for video that provides a sequence of frames.

class ConnectionStrategy(*value: int*) → None

Members:

kAutoManage

kKeepOpen

kForceClose

kAutoManage = <ConnectionStrategy.kAutoManage: 0>

kForceClose = <ConnectionStrategy.kForceClose: 2>

kKeepOpen = <ConnectionStrategy.kKeepOpen: 1>

property name

class Kind(*value: int*) → None

Members:

kUnknown

kUsb

kHttp

kCv

kCv = <Kind.kCv: 4>

kHttp = <Kind.kHttp: 2>

kUnknown = <Kind.kUnknown: 0>

kUsb = <Kind.kUsb: 1>

property name

enumerateProperties() → List[*cscore.VideoProperty*]

Enumerate all properties of this source

enumerateSinks() → List[*cscore.VideoSink*]

Enumerate all sinks connected to this source.

Returns list of sinks.

static enumerateSources() → List[*cscore.VideoSource*]

Enumerate all existing sources.

Returns list of sources.

enumerateVideoModes() → List[*cscore.VideoMode*]

Enumerate all known video modes for this source.

getActualDataRate() → float

Get the data rate (in bytes per second).

setTelemetryPeriod() must be called for this to be valid.

Returns Data rate averaged over the telemetry period.

getActualFPS() → float
Get the actual FPS.

setTelemetryPeriod() must be called for this to be valid.

Returns Actual FPS averaged over the telemetry period.

getConfigJson() → str
Get a JSON configuration string.

Returns JSON string

getDescription() → str
Get the source description. This is source-kind specific.

getHandle() → int

getKind() → *cscore.VideoSource.Kind*
Get the kind of the source

getLastFrameTime() → int
Get the last time a frame was captured.

getLastStatus() → int

getName() → str
Get the name of the source. The name is an arbitrary identifier provided when the source is created, and should be unique.

getProperty(*name: str*) → *cscore.VideoProperty*
Get a property.

Parameters **name** – Property name

Returns Property contents (VideoSource.Kind.kNone if no property with the given name exists)

getVideoMode() → *cscore.VideoMode*
Get the current video mode.

isConnected() → bool
Is the source currently connected to whatever is providing the images?

setConfigJson(*config: str*) → bool
Set video mode and properties from a JSON configuration string.

Parameters **config** – Configuration

Returns True if set successfully

setConnectionStrategy(*strategy:* cscore.VideoSource.ConnectionStrategy) → None
Set the connection strategy. By default, the source will automatically connect or disconnect based on whether any sinks are connected.

Parameters **strategy** – connection strategy (see ConnectionStrategy)

setFPS(*fps: int*) → bool
Set the frames per second (FPS).

Parameters **fps** – desired FPS

Returns True if set successfully

setPixelFormat(*pixelFormat:* cscore.VideoMode.PixelFormat) → bool
Set the pixel format.

Parameters **pixelFormat** – desired pixel format

Returns True if set successfully

setResolution(*width: int*, *height: int*) → bool
Set the resolution.

Parameters

- **width** – desired width
- **height** – desired height

Returns True if set successfully

setVideoMode(**args*, ***kwargs*)
Overloaded function.

1. setVideoMode(mode: cscore.VideoMode) -> bool

Set the video mode.

Parameters **mode** – Video mode

2. setVideoMode(pixelFormat: cscore.VideoMode.PixelFormat, width: int, height: int, fps: int) -> bool

Set the video mode.

Parameters

- **pixelFormat** – desired pixel format
- **width** – desired width
- **height** – desired height
- **fps** – desired FPS

Returns True if set successfully

1.2.18 Utility functions

cscore.**getHttpCameraUrls**(*source: int*) → List[str]

cscore.**getNetworkInterfaces**() → List[str]

cscore.**getTelemetryElapsedTime**() → float

cscore.**getUsbCameraPath**(*source: int*) → str

cscore.**setLogger**(*func: Callable[[int, str, int, str], None]*, *min_level: int*) → None

cscore.**setTelemetryPeriod**(*seconds: float*) → None

1.3 Utilities

class cscore.imagewriter.**ImageWriter**(*, *location_root='/media/sda1/camera'*, *capture_period=0.5*, *image_format='jpg'*)
Creates a thread that periodically writes images to a specified directory. Useful for looking at images after a match has completed.

The default location is `/media/sda1/camera`. The folder `/media/sda1` is the default location that USB drives inserted into the RoboRIO are mounted at. The USB drive must have a directory in it named `camera`.

Note: It is recommended to only write images when something useful (such as targeting) is happening, otherwise you'll end up with a lot of images written to disk that you probably aren't interested in.

Intended usage is:

```
self.image_writer = ImageWriter()

..

while True:

    img = ..

    if self.logging_enabled:
        self.image_writer.setImage(img)
```

Parameters

- **location_root** – Directory to write images to. A subdirectory with the current time will be created, and timestamped images will be written to the subdirectory.
- **capture_period** – How often to write images to disk
- **image_format** – File extension of files to write

setImage(*img*)

Call this function when you wish to write the image to disk. Not every image is written to disk. Makes a copy of the image.

Parameters **img** – A numpy array representing an OpenCV image

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

H

I

K

W