Qualcomm Technologies, Inc.

Snapdragon Linux Camera

Interface Specification

80-P3945-1 A

January 18, 2016

Qualcomm Technologies, Inc.
5775 Morehouse Drive
San Diego, CA 92121
U.S.A.

Revision History

Revision	Date	Description
A	Jan 2016	Initial release

Contents

1 Introduction

1.1 Purpose

This document describes how to interface with a Snapdragon Camera device using a native C/C++ based application.

The camera interface is provided using the libcamera library.

1.2 Conventions

Function declarations, function names, type declarations, and code samples appear in a different font. For example, `#include`.

Code variables appear in angle brackets. For example, <`number`>.

1.3 Technical Assistance

For assistance or clarification on information in this guide, submit a case to Qualcomm Technologies, Inc. at https://support.cdmatech.com.

If you do not have access to the CDMATech Support website, register for access or send email to support.cdmatech@qti.qualcomm.com.

2 Overview

This document describes how to interface with a Snapdragon Camera device using a native C/C++ based application.

The camera interface is provided using the libcamera library.

2.1 Initializing the Camera

The following is required to initialize the camera:

- Use the camera::getNumberOfCameras() function to check available cameras for the platform.
- camera::getCameraInfo() is used to enumerate the available cameras. The ID provided to this function should be in [0, N-1] if N is returned from camera::getNumberOfCameras().
- Use the camera::ICameraDevice::createInstance() function to create a camera instance for a given ID.
- Create a class that implements the camera::ICameraListener interface.
- Add an instance of this class to the camera device using the camera::ICameraDevice::addListener() function.

2.2 Setting Parameters

2.2.1 Getting Current Parameter Values

When the camera is opened, it is initialized with a default set of parameters. The current value of any parameter can be queried using the get∗ method. For example, camera::CameraParams::getPreviewSize() returns an ImageSize object representing the current preview resolution.

2.2.2 Changing Parameter Values

Use the following steps to change parameter values:

1. Query supported values using getSupported∗ APIs. For example, camera::CameraParams::getSupportedPreviewSizes() will return a vector of camera::ImageSize(s) supported by the preview stream.
2. Set a valid value using set∗ APIs. For example, call the camera::CameraParams::setPreviewSize() function with one of the camera::ImageSize(s) returned from the previous query.
3. Repeat this with all other parameters that need to be changed.

2.2.3 Committing Parameters

The set API above updates parameter values in a local object, but it does not apply these values to the camera hardware. Use the camera::CameraParams::commit() function to update the hardware settings with the current state of the parameter object.

2.3 Using the Preview Stream

The camera::ICameraDevice::startPreview() method starts the preview stream on the given camera device. While preview is running, the ICameraListener::onPreviewFrame() method is invoked for every frame. The camera::ICameraDevice::stopPreview() method stops the preview stream.

2.4 Using the Video Stream

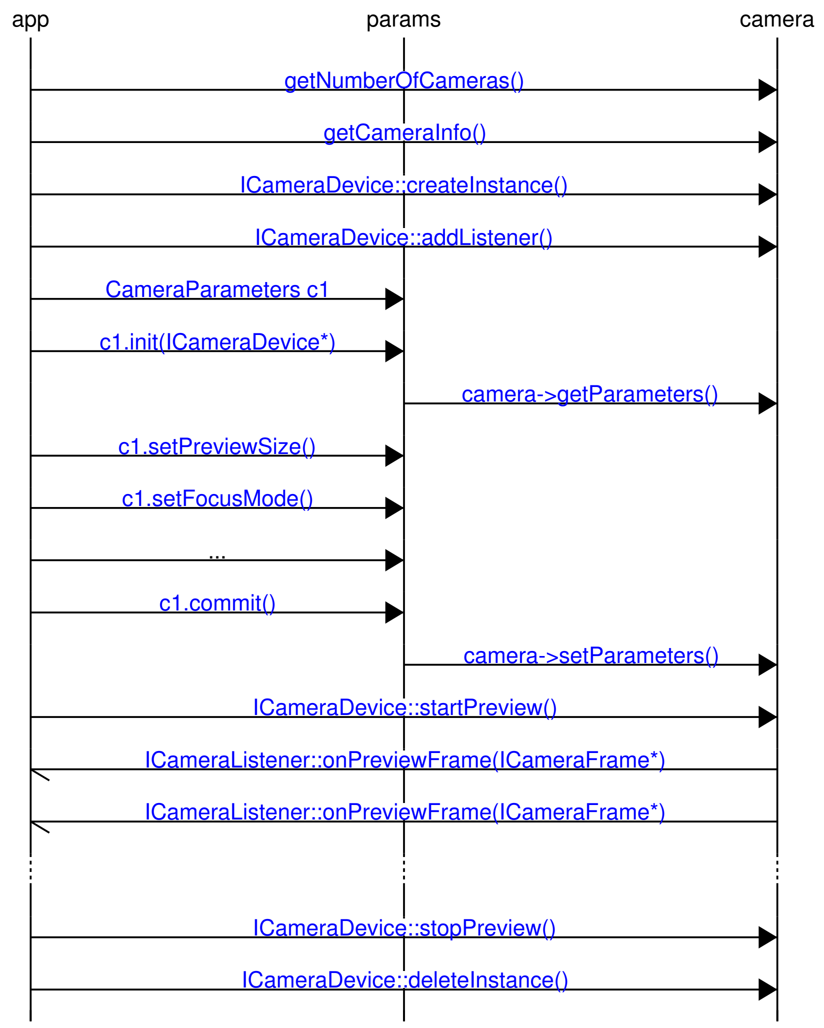
The camera::ICameraDevice::startRecording() method starts the video stream on the given camera device. While video is running, the camera::ICameraListener::onVideoFrame() method is invoked for every frame. The camera::ICameraDevice::stopRecording() method stops the video stream.

2.5 Taking Pictures

The following is required to take a picture:

- The camera::ICameraDevice::takePicture() function triggers the capture of a snapshot.
- The caller must wait for the snapshot frame to be delivered before additional pictures can be taken.
- The output snapshot image is delivered in the camera::ICameraListener::onPictureFrame() callback. This function should awaken the waiting thread.
- The snapshot dimensions and format can be set using the corresponding parameters defined in camera::CameraParams.

2.6 Call Flow Sequence Diagram for a Typical Camera Use Case


app
params
camera
getNumberOfCameras()
getCameraInfo()
ICameraDevice::createInstance()
ICameraDevice::addListener()
CameraParameters c1
c1.init(ICameraDevice*)
camera->getParameters()
c1.setPreviewSize()
c1.setFocusMode()
...
c1.commit()
camera->setParameters()
ICameraDevice::startPreview()
ICameraListener::onPreviewFrame(ICameraFrame*)
ICameraListener::onPreviewFrame(ICameraFrame*)
ICameraDevice::stopPreview()
ICameraDevice::deleteInstance()

2.7 More Information

For specific use cases of the API, such as stereo camera streaming, raw preview, and high framerate support, refer to the application note in the example test application. This application is located at:

hardware/qcom/camera/libcamera/test/camera_test.cpp

3 Camera Interface - API Reference

- Camera Device APIs
- Camera Parameter APIs

3.1 Camera Device APIs

This section contains the main camera APIs for identifying the connected camera devices and performing operations.

3.1.1 Class Documentation

3.1.1.1 struct camera::ControlEvent

Data structure to represent a single control event.

Data fields

Type	Field	Description
int	ext1	Control event 1.
int	ext2	Control event 2.
EventType	type	Control event type.

3.1.1.2 class camera::ICameraFrame

Interface to an object that represents a single image/metadata frame.

Public member functions

- virtual uint32_t acquireRef ()=0
- virtual uint32_t releaseRef ()=0

Public Attributes

- uint64_t timeStamp
- uint32_t size
- uint8_t * data
- int fd
- void * metadata

Protected Attributes

- uint32_t **refs_**

3.1.1.2.1 Member Function Documentation

3.1.1.2.1.1 virtual uint32_t camera::ICameraFrame::acquireRef ( ) `[pure virtual]`

Acquires a reference to the frame. This is required if the client wants to hold the frame for further processing after the camera callback returns.

Returns

uint32_t – The number of references to the frame.

3.1.1.2.1.2 virtual uint32_t camera::ICameraFrame::releaseRef ( ) `[pure virtual]`

Releases a reference to the frame object. This releases memory associated with image data as well.

Returns

uint32_t – The number of references to the frame.

3.1.1.2.2 Member Data Documentation

3.1.1.2.2.1 uint64_t camera::ICameraFrame::timeStamp

Frame timestamp.

3.1.1.2.2.2 uint32_t camera::ICameraFrame::size

Frame data size in bytes.

3.1.1.2.2.3 uint8_t* camera::ICameraFrame::data

Pointer to the start of the image data

3.1.1.2.2.4 int camera::ICameraFrame::fd

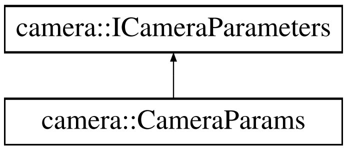
File descriptor for the frame memory. If the value is -1, it indicates that the memory is not shared.

3.1.1.2.2.5 void* camera::ICameraFrame::metadata

Opaque metadata about the frame memory. This is used for sharing memory between camera and video encoder.

3.1.1.3 class camera::ICameraParameters

Interface to a parameters object. Inheritance diagram for camera::ICameraParameters:



Public member functions

- virtual int writeObject (std::ostream &ps) const =0

3.1.1.3.1 Member Function Documentation

3.1.1.3.1.1 virtual int camera::ICameraParameters::writeObject (std::ostream & *ps*) const `[pure virtual]`

Gets a serialized representation of the parameter object in an ostream.

Parameters

`out`	*ps*	Reference to an ostream object.

Returns

int – 0 on success.

Implemented in camera::CameraParams.

3.1.1.4 class camera::ICameraListener

Interface for a camera listener object. The client must implement this interface to get access to camera data and control events. The methods in listener can be invoked from multiple different threads. The client must make sure that implementation is thread safe.

Public member functions

- virtual void onError ()
- virtual void onControl (const ControlEvent &control)
- virtual void onPreviewFrame (ICameraFrame ∗frame)
- virtual void onVideoFrame (ICameraFrame ∗frame)
- virtual void onPictureFrame (ICameraFrame ∗frame)
- virtual void onMetadataFrame (ICameraFrame ∗frame)

3.1.1.4.1 Member Function Documentation

3.1.1.4.1.1 virtual void camera::ICameraListener::onError ( ) `[virtual]`

Function called when an error is generated by the camera driver.

3.1.1.4.1.2 virtual void camera::ICameraListener::onControl (const ControlEvent & *control*) `[virtual]`

Function called when a control event is to be delivered to a client.

Parameters

`in`	*control*	Control event object.

3.1.1.4.1.3 virtual void camera::ICameraListener::onPreviewFrame (ICameraFrame ∗ *frame*) `[virtual]`

Function called when a preview frame is generated by the camera.

Parameters

`in`	*frame*	Pointer to an existing ICameraFrame generated by the camera.

3.1.1.4.1.4 virtual void camera::ICameraListener::onVideoFrame (ICameraFrame ∗ *frame*) `[virtual]`

Function called when a video frame is generated by the camera.

Parameters

`in`	*frame*	Pointer to an existing ICameraFrame generated by the camera.

3.1.1.4.1.5 virtual void camera::ICameraListener::onPictureFrame (ICameraFrame ∗ *frame*) `[virtual]`

Function called when a snapshot frame is generated by the camera.

Parameters

`in`	*frame*	Pointer to an existing ICameraFrame generated by the camera.

3.1.1.4.1.6 virtual void camera::ICameraListener::onMetadataFrame (ICameraFrame ∗ *frame*) [virtual]

Function called when a metadata frame is generated by the camera.

Parameters

in	*frame*	Pointer to an existing ICameraFrame generated by the camera.

3.1.1.5 class camera::ICameraDevice

Interface to a camera device. The client uses the API provided by this interface to interact with a camera device.

Public member functions

- virtual void addListener (ICameraListener ∗listener)=0
- virtual void removeListener (ICameraListener ∗listener)=0
- virtual void subscribe (uint32_t eventMask)=0
- virtual void unsubscribe (uint32_t eventMask)=0
- virtual int setParameters (const ICameraParameters &params)=0
- virtual int getParameters (uint8_t ∗buf, uint32_t bufSize, int ∗bufSizeRequired=NULL)=0
- virtual int startPreview ()=0
- virtual void stopPreview ()=0
- virtual int startRecording ()=0
- virtual void stopRecording ()=0
- virtual int takePicture ()=0

Static Public Member Functions

- static int createInstance (int index, ICameraDevice ∗∗device)
- static void deleteInstance (ICameraDevice ∗∗device)

3.1.1.5.1 Member Function Documentation

3.1.1.5.1.1 static int camera::ICameraDevice::createInstance (int *index,* ICameraDevice ∗∗ *device*) [static]

Factory method to create an instance of ICameraDevice.

Parameters

in	*index*	Camera ID.
out	*device*	Pointer to an ICameraDevice∗ to be created.

Returns

int – 0 on success.

3.1.1.5.1.2 static void camera::ICameraDevice::deleteInstance (ICameraDevice ** *device*) `[static]`

Deletes an instance of ICameraDevice.

Parameters

`out`	*device*	Pointer to an ICameraDevice* to be deleted.

3.1.1.5.1.3 virtual void camera::ICameraDevice::addListener (ICameraListener * *listener*) `[pure virtual]`

Adds a listener to handle various notifications and data frames from the camera device.

Parameters

`in`	*listener*	Listener to be added.

3.1.1.5.1.4 virtual void camera::ICameraDevice::removeListener (ICameraListener * *listener*) `[pure virtual]`

Removes a previously added listener from a camera device.

Parameters

`in`	*listener*	Listener to be removed.

3.1.1.5.1.5 virtual void camera::ICameraDevice::subscribe (uint32_t *eventMask*) `[pure virtual]`

Subscribes for camera control events.

Parameters

`in`	*eventMask*	Bitmask of values in enum EventType.

Returns

int – 0 on success.

3.1.1.5.1.6 virtual void camera::ICameraDevice::unsubscribe (uint32_t *eventMask*) `[pure virtual]`

Unsubscribes from previously subscribed control events.

Parameters

`in`	*eventMask*	Bitmask of values in enum EventType.

Returns

int – 0 on success.

3.1.1.5.1.7 virtual int camera::ICameraDevice::setParameters (const ICameraParameters & *params*) `[pure virtual]`

Sets parameters in the camera device. The camera device state is updated with new settings when this call returns.

Parameters

`in`	*params*	Populated parameters object with new camera parameters to be set.

Returns

int – 0 on success.

3.1.1.5.1.8 virtual int camera::ICameraDevice::getParameters (uint8_t ∗ *buf,* uint32_t *bufSize,* int ∗ *bufSizeRequired* = *`NULL`*) `[pure virtual]`

Retrieves the octet stream of parameters as name and value pairs. Note that parameters fetched in to be buffered may be partial to the extent of bufSize. Use bufSizeRequired to determine the total length of buffer required to get all the parameters.

Parameters

`out`	*buf*	Buffer that will be populated with the octet stream of parameters.
`in`	*bufSize*	Size of the memory at buf.
`out`	*bufSizeRequired*	Optional; if provided, it will be populated with the total buffer size required to fetch all the parameters.

Returns

int – 0 on success.

3.1.1.5.1.9 virtual int camera::ICameraDevice::startPreview ( ) `[pure virtual]`

Starts a preview stream on the camera.

Returns

int – 0 on success.

3.1.1.5.1.10 virtual void camera::ICameraDevice::stopPreview ( ) `[pure virtual]`

Stops preview streaming.

3.1.1.5.1.11 virtual int camera::ICameraDevice::startRecording ( ) `[pure virtual]`

Starts a video recording stream.

Returns

int – 0 on success.

3.1.1.5.1.12 virtual void camera::ICameraDevice::stopRecording ( ) `[pure virtual]`

Stops video recording.

3.1.1.5.1.13 virtual int camera::ICameraDevice::takePicture ( ) `[pure virtual]`

Takes a picture.

Preview must be running when this method is called. The video recording may or may not be running. If successful, the onPictureFrame() callback is called with image data for the picture. The client must wait for this callback before taking any more pictures.

Returns

int

3.1.1.6 struct camera::CameraInfo

Structure to hold information about a single camera device.

Data fields

Type	Field	Description
int	func	Defines the function of a respective camera in a specified platform. The value depends on the type of platform (front camera vs. back camera, high resolution camera vs. stereo camera, etc.).

3.1.2 Enumeration Type Documentation

3.1.2.1 enum camera::EventType

Type of control event received from the camera.

Enumerator

CAMERA_EVT_NONE	No control event.
CAMERA_EVT_FOCUS	Focus contol event.

3.1.3 Function Documentation

3.1.3.1 int camera::getNumberOfCameras ( )

Gets the number of camera sensors detected on a device.

Returns

int

3.1.3.2 int camera::getCameraInfo (int *idx*, struct CameraInfo & info)

Gets the additional information about a camera by index. index 0 identifies the first camera.

Parameters

`in`	*idx*	Index of the camera device. Index 0 identifies the first camera.
`out`	*info*	Information about the camera device.

Returns

int – 0 is success.

3.2 Camera Parameter APIs

This section contains utility APIs to query and program several parameters and commit to a camera device.

3.2.1 Class Documentation

3.2.1.1 struct camera::ImageSize

Image frame dimensions.

Public member functions

- **ImageSize** (int w, int h)

Public Attributes

- int width
- int height

3.2.1.1.1 Member Data Documentation

3.2.1.1.1.1 int camera::ImageSize::width

Image width in pixels.

3.2.1.1.1.2 int camera::ImageSize::height

Image height in pixels.

3.2.1.2 struct camera::Range

Structure for storing values for ranged parameters, such as brightness, contrast, fps, etc.

Public member functions

- **Range** (int m, int x, int s)

Public Attributes

- int min
- int max
- int step

3.2.1.2.1 Member Data Documentation

3.2.1.2.1.1 int camera::Range::min

Minimum value.

3.2.1.2.1.2 int camera::Range::max

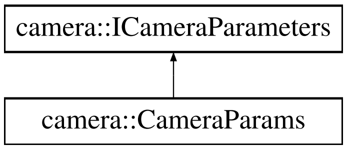
Maximum value.

3.2.1.2.1.3 int camera::Range::step

Step increment for intermediate values.

3.2.1.3 class camera::CameraParams

Inheritance diagram for camera::CameraParams:



Public member functions

- int init (ICameraDevice ∗device)
- virtual int writeObject (std::ostream &ps) const
- std::string toString () const
- int commit ()
- std::vector< ImageSize > getSupportedPreviewSizes () const
- void setPreviewSize (const ImageSize &size)
- ImageSize getPreviewSize () const
- std::vector< ImageSize > getSupportedVideoSizes () const
- ImageSize getVideoSize () const
- void setVideoSize (const ImageSize &size)
- std::vector< ImageSize > getSupportedPictureSizes () const
- ImageSize getPictureSize () const
- void setPictureSize (const ImageSize &size)
- virtual std::string get (const std::string &key) const
- virtual void set (const std::string &key, const std::string &value)
- std::vector< std::string > getSupportedFocusModes () const

- std::string getFocusMode () const
- void setFocusMode (const std::string &value)
- std::vector< std::string > getSupportedWhiteBalance () const
- std::string getWhiteBalance () const
- void setWhiteBalance (const std::string &value)
- std::vector< std::string > getSupportedISO () const
- std::string getISO () const
- void setISO (const std::string &value)
- Range getSupportedSharpness () const
- int getSharpness () const
- void setSharpness (int value)
- Range getSupportedBrightness () const
- int getBrightness () const
- void setBrightness (int value)
- Range getSupportedContrast () const
- int getContrast () const
- void setContrast (int value)
- std::vector< Range > getSupportedPreviewFpsRanges () const
- Range getPreviewFpsRange () const
- void setPreviewFpsRange (const Range &value)
- std::vector< VideoFPS > getSupportedVideoFps () const
- VideoFPS getVideoFPS () const
- void setVideoFPS (VideoFPS value)
- std::vector< std::string > getSupportedPreviewFormats () const
- std::string getPreviewFormat () const
- void setPreviewFormat (const std::string &value)
- void setManualExposure (int value)
- void setManualGain (int value)
- void setVerticalFlip (bool value)
- void setHorizontalMirror (bool value)

3.2.1.3.1 Member Function Documentation

3.2.1.3.1.1 int camera::CameraParams::init (ICameraDevice ∗ *device*)

Initializes an object by getting the current state of parameters from the device.

Parameters

`in`	*device*	A valid camera device object.

Returns

int – 0 on success.

3.2.1.3.1.2 virtual int camera::CameraParams::writeObject (std::ostream & *ps*) const `[virtual]`

Gets a serialized representation of the parameter object in an ostream.

Parameters

`out`	*ps*	Reference to an ostream object.

Returns

int – 0 on success.

Implements camera::ICameraParameters.

3.2.1.3.1.3 std::string camera::CameraParams::toString ( ) const

Gets a string representation of the object.

Returns

std::string

3.2.1.3.1.4 int camera::CameraParams::commit ( )

Updates the current state of the parameters to a camera device. Fails with any invalid entries.

Returns

int – 0 on success.

3.2.1.3.1.5 std::vector<ImageSize> camera::CameraParams::getSupportedPreviewSizes ( ) const

Gets the preview sizes supported by the camera.

Returns

std::vector<ImageSize> – List of preview sizes.

3.2.1.3.1.6 void camera::CameraParams::setPreviewSize (const ImageSize & *size*)

Sets the preview size.

Parameters

in	*size*	Preview size.

3.2.1.3.1.7 ImageSize camera::CameraParams::getPreviewSize ( ) const

Gets the current preview size.

Returns

ImageSize

3.2.1.3.1.8 std::vector<ImageSize> camera::CameraParams::getSupportedVideoSizes ( ) const

Gets the video sizes supported by the camera.

Returns

std::vector<ImageSize> – List of video sizes.

3.2.1.3.1.9 ImageSize camera::CameraParams::getVideoSize ( ) const

Gets the current video size.

Returns

ImageSize

3.2.1.3.1.10 void camera::CameraParams::setVideoSize (const ImageSize & *size*)

Sets the video size.

Parameters

in	*size*	Video size.

3.2.1.3.1.11 std::vector<ImageSize> camera::CameraParams::getSupportedPictureSizes ( ) const

Gets the picture sizes supported by the camera.

Returns

std::vector<ImageSize> – List of picture sizes.

3.2.1.3.1.12 ImageSize camera::CameraParams::getPictureSize ( ) const

Gets the current picture size.

Returns

ImageSize

3.2.1.3.1.13 void camera::CameraParams::setPictureSize (const ImageSize & *size*)

Sets the picture size.

See takePicture().

Parameters

`in`	*size*	Picture size.

3.2.1.3.1.14 virtual std::string camera::CameraParams::get (const std::string & *key*) const `[virtual]`

Generic get function to get a string representation of the value of a parameter using a key.

Parameters

`in`	*key*	Key to use to get the parameter value.

Returns

std::string – Parameter value.

3.2.1.3.1.15 virtual void camera::CameraParams::set (const std::string & *key*, const std::string & *value*) `[virtual]`

Generic set function to set the value of a parameter using a key-value pair.

Parameters

`in`	*key*	Key to use to set the parameter value.
`in`	*value*	Parameter value to set.

3.2.1.3.1.16 std::vector<std::string> camera::CameraParams::getSupportedFocusModes ( ) const

Gets a list of supported focus modes.

Returns

vector<string> – Focus mode values.

3.2.1.3.1.17 std::string camera::CameraParams::getFocusMode ( ) const

Gets the current focus mode value.

Returns

std::string – Focus mode value.

3.2.1.3.1.18 void camera::CameraParams::setFocusMode (const std::string & *value*)

Sets the focus mode value.

Parameters

in	*value*	Focus mode value to set.

3.2.1.3.1.19 std::vector<std::string> camera::CameraParams::getSupportedWhiteBalance ( ) const

Gets a list of supported white balance modes.

Returns

vector<string> – White balance values.

3.2.1.3.1.20 std::string camera::CameraParams::getWhiteBalance ( ) const

Gets the current value of the white balance mode.

Returns

std::string – White balance mode value.

3.2.1.3.1.21 void camera::CameraParams::setWhiteBalance (const std::string & *value*)

Sets the white balance mode value.

Parameters

in	*value*	White balance mode value to set.

3.2.1.3.1.22 std::vector<std::string> camera::CameraParams::getSupportedISO ( ) const

Gets a list of supported ISO modes.

Returns

vector<string> – ISO values.

3.2.1.3.1.23 std::string camera::CameraParams::getISO ( ) const

Gets the current ISO mode value.

Returns

std::string – ISO mode value.

3.2.1.3.1.24 void camera::CameraParams::setISO (const std::string & *value*)

Sets the ISO mode value.

Parameters

`in`	*value*	ISO mode value.

3.2.1.3.1.25 Range camera::CameraParams::getSupportedSharpness ( ) const

Gets a range of supported sharpness values.

Returns

Range – Sharpness value range.

3.2.1.3.1.26 int camera::CameraParams::getSharpness ( ) const

Gets the current sharpness value.

Returns

int – Sharpness value.

3.2.1.3.1.27 void camera::CameraParams::setSharpness (int *value*)

Sets the sharpness value.

Parameters

`in`	*value*	Sharpness value.

3.2.1.3.1.28 Range camera::CameraParams::getSupportedBrightness ( ) const

Gets a range of supported brightness values.

Returns

Range – Brightness range.

3.2.1.3.1.29 int camera::CameraParams::getBrightness ( ) const

Gets the current brightness value.

Returns

int – Brightness value.

3.2.1.3.1.30 void camera::CameraParams::setBrightness (int *value*)

Sets the brightness value.

Parameters

in	*value*	Brightness value.

3.2.1.3.1.31 Range camera::CameraParams::getSupportedContrast ( ) const

Gets a range of supported contrast values.

Returns

Range – Contrast range.

3.2.1.3.1.32 int camera::CameraParams::getContrast ( ) const

Gets the current contrast value.

Returns

int – Contrast value.

3.2.1.3.1.33 void camera::CameraParams::setContrast (int *value*)

Sets the contrast value.

Parameters

in	*value*	Contrast value.

3.2.1.3.1.34 std::vector<Range> camera::CameraParams::getSupportedPreviewFpsRanges ( ) const

Get the supported ranges for the preview FPS. The FPS range has valid minimum and maximum values. The actual fixed point FPS value is calculated by dividing the min and max values by 1000. For example, a maximum value of 26123 represents 26.123 fps.

Returns

vector<Range> – Preview FPS ranges.

3.2.1.3.1.35 Range camera::CameraParams::getPreviewFpsRange ( ) const

Gets the current preview fps range value.

Returns

Range – FPS range value.

3.2.1.3.1.36 void camera::CameraParams::setPreviewFpsRange (const Range & *value*)

Sets the preview FPS range value.

Parameters

`in`	*value*	FPS range value.

3.2.1.3.1.37 std::vector<VideoFPS> camera::CameraParams::getSupportedVideoFps ( ) const

Gets a list of fixed FPS values for the video stream.

Returns

vector<VideoFPS> – Video FPS values.

3.2.1.3.1.38 VideoFPS camera::CameraParams::getVideoFPS ( ) const

Gets the current video FPS mode value.

Returns

VideoFPS – Video FPS mode value.

3.2.1.3.1.39 void camera::CameraParams::setVideoFPS (VideoFPS *value*)

Sets the video FPS mode value.

Note

Setting the mode to a high frame rate will override preview FPS settings. See VideoFPS for high frame rate values.

Parameters

in	*value*	Video FPS mode value.

3.2.1.3.1.40 std::vector<std::string> camera::CameraParams::getSupportedPreviewFormats ( ) const

Gets a list of supported preview formats.

Returns

std::vector<std::string> – List of preview formats.

3.2.1.3.1.41 std::string camera::CameraParams::getPreviewFormat ( ) const

Gets the current preview format.

Returns

std::string – Preview format.

3.2.1.3.1.42 void camera::CameraParams::setPreviewFormat (const std::string & *value*)

Sets the preview format.

Parameters

in	*value*	Preview format value.

3.2.1.3.1.43 void camera::CameraParams::setManualExposure (int *value*)

Sets the manual exposure value in the sensor.

Parameters

in	*value*	Manual exposure value.

3.2.1.3.1.44 void camera::CameraParams::setManualGain (int *value*)

Sets the manual gain value in the sensor.

Parameters

`in`	*value*	Manual gain value.

3.2.1.3.1.45 void camera::CameraParams::setVerticalFlip (bool *value*)

Sets the vertical flip bit value in the sensor.

Parameters

`in`	*value*	Vertical flip bit value.

3.2.1.3.1.46 void camera::CameraParams::setHorizontalMirror (bool *value*)

Sets the horizontal mirror bit value in the sensor.

Parameters

`in`	*value*	Horizontal mirror bit value.

3.2.2 Enumeration Type Documentation

3.2.2.1 enum camera::VideoFPS

Available values for video FPS

Enumerator

VIDEO_FPS_30	30 fps regular Framerate mode.
VIDEO_FPS_60	60 fps high Framerate mode.
VIDEO_FPS_90	90 fps high Framerate mode.
VIDEO_FPS_120	120 fps high Framerate mode.
VIDEO_FPS_150	150 fps high Framerate mode.

A References

A.1 Acronyms and Terms

Acronym or term	Definition
AEC	Auto exposure control
AF	Auto focus
AWB	Auto white balance
FPS	Frames per second (also written fps)