Qualcomm Technologies, Inc.

QCarCam API

Reference

80-P2310-25 Rev. F

January 28, 2022

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

Revision history

Revision	Date	Description
A	June 2017	Initial release
B	June 2018	Updated Chapter 2 introduction and Section 3.8 as marked.
C	July 2020	Updated to QCarCam API v4.9. Numerous changes were made to this document; it should be read in its entirety.
D	February 2021	Updated ISP parameters.
E	June 2021	Updated to QCarCam API v4.12.
F	January 2022	Updated to QCarCam API v4.13

Contents

Figures

Tables

1 Introduction

1.1 Purpose

QCarCam is the QTI automotive proprietary API. This document details QCarCam utility, benefits, and basic API calls.

1.2 Conventions

Function declarations, function names, type declarations, attributes, and code samples appear in a different font, for example, `cp armcc armcpp`.

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

Commands to be entered appear in a different font, for example, **`copy a:*.* b:`**.

Button and key names appear in bold font, for example, click **Save** or press **Enter**.

Shading indicates content that has been added or changed in this revision of the document.

1.3 Technical assistance

For assistance or clarification on information in this document, submit a case to Qualcomm Technologies, Inc. (QTI) at https://createpoint.qti.qualcomm.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

QCarCam API is designed for the QTI Automotive Imaging System (AIS) and is optimized for automotive usecases. QCarCam API design benefits from AIS OS portability, multi-user access, input permissions, and bridge chip abstraction for entity access.

QCarCam API can interact with various imaging inputs, including MIPI sensors, bridge chips (HDMI/CVBS, FDPLink), and memory buffers under the AIS framework.

AIS can set access levels for specific users and applications so that any camera input can become privileged, allowing multiple user-access to the same camera input simultaneously. Users can also access non-camera input sources, such as HDMI or video input that use CSI inputs.

2.1 Feature support list

The QCarCam API supports the following features:

- Cross OS and Hypervisor Support (LA/LV/QNX/GHS)
- Multiclient support
- Hardware-assisted multistream of same input
- Query available input streams
- Open and close input stream handle
- Get and set stream parameters
- Start and stop stream
- Pause and resume stream
- Post processing framework
- Event and error notification
- Self-recovery of streams

3 QCarCam API

The QCarCam API definition is in `qcarcam.h` and `qcarcam_types.h` files.

3.1 API version

The QCarCam API version comprises of a major and minor version.

```
#define QCARCAM_VERSION_MAJOR              4
#define QCARCAM_VERSION_MINOR              12
#define QCARCAM_VERSION                    ((QCARCAM_VERSION_MAJOR << 8) |
QCARCAM_VERSION_MINOR)
```

The minimum compatible API version is also defined

```
#define QCARCAM_VERSION_MINOR_COMPATIBLE   7
#define QCARCAM_VERSION_MINIMUM_COMPATIBLE ((QCARCAM_VERSION_MAJOR << 8) |
QCARCAM_VERSION_MINOR_COMPATIBLE)
```

3.2 Return types

```
typedef enum
{
    QCARCAM_RET_OK = 0,
    QCARCAM_RET_FAILED,
    QCARCAM_RET_BADPARAM,
    QCARCAM_RET_BADSTATE,
    QCARCAM_RET_NOMEM,
    QCARCAM_RET_UNSUPPORTED,
    QCARCAM_RET_TIMEOUT,
} qcarcam_ret_t;
```

Table 3-1 QCarCam return codes

Return codes	Description
OK	Success
FAILED	Operation failed
BADPARAM	Invalid parameter
BADSTATE	Invalid state
NOMEM	Out of memory
UNSUPPORTED	Unsupported value, for example, unsupported parameter was set
TIMEOUT	Operation timedout, for example, get frame was called with a timeout value that expired

3.3 Unique input identifiers

Input identifiers are used as logical identifiers. The QCarCam API defines some common identifiers that a user may want to use or define on their own. These identifiers are mapped to physical inputs through an input mapping table in CameraConfig (see *SA6155/SA8155 Automotive Camera AIS Customization Guide* (80-PG469-93) for more details).

```
typedef enum
{
    QCARCAM_INPUT_TYPE_EXT_REAR      = 0,   ///< Rearview
    QCARCAM_INPUT_TYPE_EXT_FRONT     = 1,   ///< Exterior Front
    QCARCAM_INPUT_TYPE_EXT_LEFT      = 2,
    QCARCAM_INPUT_TYPE_EXT_RIGHT     = 3,
    QCARCAM_INPUT_TYPE_DRIVER        = 4,   ///< Driver monitor
    QCARCAM_INPUT_TYPE_LANE_WATCH    = 5,
    QCARCAM_INPUT_TYPE_DIGITAL_MEDIA = 6,
    QCARCAM_INPUT_TYPE_ANALOG_MEDIA  = 7,
    QCARCAM_INPUT_TYPE_GESTURE       = 8,
    QCARCAM_INPUT_TYPE_IRIS          = 9,
    QCARCAM_INPUT_TYPE_FINGERPRINT   = 10,
    QCARCAM_INPUT_TYPE_TESTPATTERN   = 255,
    QCARCAM_INPUT_TYPE_USER_DEFINED_START= 256,  /// User defined IDs start
at this index
    QCARCAM_INPUT_MAX = 0x7FFFFFFF,
} qcarcam_input_desc_t;
```

Figure 3-1 shows example camera positions and mapping to input descriptors.

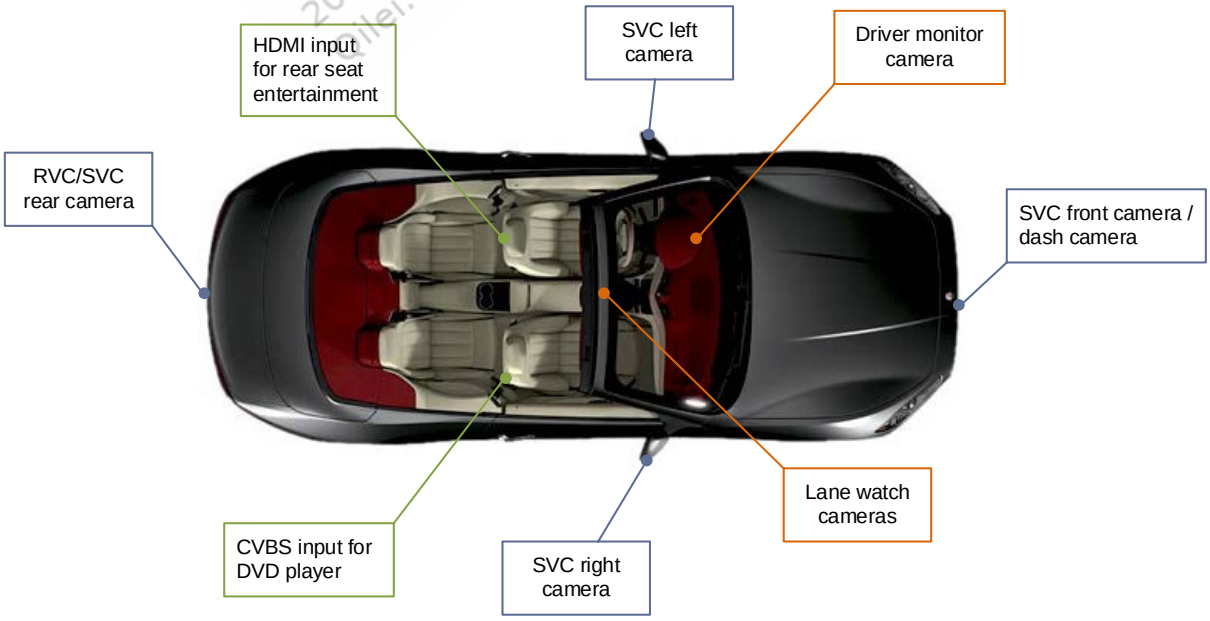


Figure 3-1 Example camera positions in a vehicle

3.4 Color format

The qcarcam_fmt_t is a mask comprised of:

- Color pattern (lower 16bits)
- Bit depth of each color channel (bits 16 to 23)
- Memory packing (bits 24 to 31)

```
#define QCARCAM_COLOR_FMT(_pattern_, _bitdepth_, _pack_) \
    (((_pack_ & 0xff) << 24) | ((_bitdepth_ & 0xff) << 16) | (_pattern_ &
0xffff))

#define QCARCAM_COLOR_GET_PATTERN(_color_) \
    ((qcarcam_color_pattern_t)(_color_ & 0xffff))

#define QCARCAM_COLOR_GET_BITDEPTH(_color_) \
    ((qcarcam_color_bitdepth_t)((_color_ & 0xff0000) >> 16))

#define QCARCAM_COLOR_GET_PACK(_color_) \
    ((qcarcam_color_pack_t)((_color_ & 0xff000000) >> 24))
```

3.4.1 QCarCam color pattern

Possible color patterns are defined in qcarcam_color_pattern_t.

```
/// @brief Color type
typedef enum
{
    QCARCAM_RAW = 0,

    QCARCAM_YUV_YUYV = 0x100,
    QCARCAM_YUV_YVYU,
    QCARCAM_YUV_UYVY,
    QCARCAM_YUV_VYUY,

    QCARCAM_YUV_NV12,
    QCARCAM_YUV_NV21,

    QCARCAM_BAYER_GBRG = 0x200,
    QCARCAM_BAYER_GRBG,
    QCARCAM_BAYER_RGGB,
    QCARCAM_BAYER_BGGR,

    QCARCAM_RGB = 0x300,

}qcarcam_color_pattern_t;
```

3.4.2 QCarCam color bit depth

Possible values for the bit depth of each color channel are defined in qcarcam_color_bitdepth_t.

```
/// @brief Bitdepth per color channel
typedef enum
{
    QCARCAM_BITDEPTH_8 = 8,
    QCARCAM_BITDEPTH_10 = 10,
    QCARCAM_BITDEPTH_12 = 12,
    QCARCAM_BITDEPTH_14 = 14,
    QCARCAM_BITDEPTH_16 = 16,
    QCARCAM_BITDEPTH_20 = 20
}qcarcam_color_bitdepth_t;
```

3.4.3 QCarCam color packing

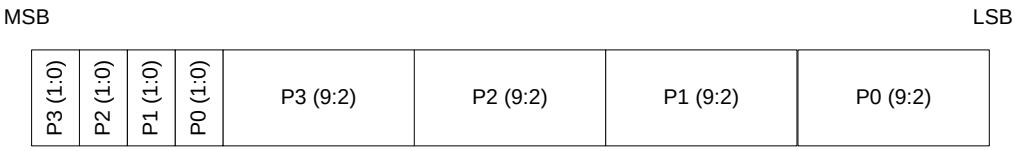
Possible memory packing options are defined in qcarcam_color_pack_t.

```
/// @brief Packing type
typedef enum
{
    QCARCAM_PACK_QTI = 0,
    QCARCAM_PACK_MIPI,
    QCARCAM_PACK_DPCM6,
    QCARCAM_PACK_DPCM8,
    QCARCAM_PACK_PLAIN8,
    QCARCAM_PACK_PLAIN16,
    QCARCAM_PACK_PLAIN32,
    QCARCAM_PACK_FOURCC
}qcarcam_color_pack_t;
```

Some common packing and bit depths include:

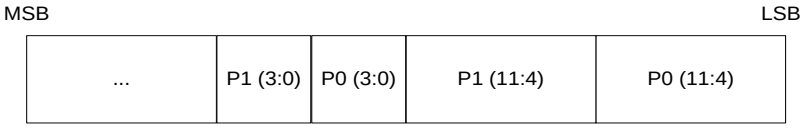
MIPI 10 bit

The 10-bit packing (shown in the following figure) is used to hold formats such as Raw10. Four pixels are held in every 5 bytes. The image output width must be a multiple of 4 pixels.

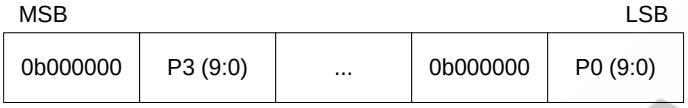


MIPI 12 bit

The 12-bit packing (shown in the following figure) is a format that holds Raw12. Two pixels are held in every 3 bytes. The image output width must be a multiple of 2 pixels.



Plain16 10-bit



Plain16 12-bit

3.4.4 QCarCam defined color formats

QCarCam API provides a definition of commonly used formats. This is nonexhaustive and users can use valid combinations of the previously described parameters to define a format using the QCARCAM_COLOR_FMT macro.

As a result, the following qcarcam_color_fmt_t definition for YUV422 8bit with UYVY ordering is

```
QCARCAM_FMT_UYVY_8 = QCARCAM_COLOR_FMT(QCARCAM_YUV_UYVY,
QCARCAM_BITDEPTH_8, QCARCAM_PACK_FOURCC),
```

The qcarcam_types.h file provides a defined list of commonly used formats.

```
typedef enum
{
    QCARCAM_FMT_MIPIRAW_8 = QCARCAM_COLOR_FMT(QCARCAM_RAW,
QCARCAM_BITDEPTH_8, QCARCAM_PACK_MIPI), ///< MIPI packed RAW 8bit
    QCARCAM_FMT_MIPIRAW_10 = QCARCAM_COLOR_FMT(QCARCAM_RAW,
QCARCAM_BITDEPTH_10, QCARCAM_PACK_MIPI),
    QCARCAM_FMT_MIPIRAW_12 = QCARCAM_COLOR_FMT(QCARCAM_RAW,
QCARCAM_BITDEPTH_12, QCARCAM_PACK_MIPI),
    QCARCAM_FMT_MIPIRAW_14 = QCARCAM_COLOR_FMT(QCARCAM_RAW,
QCARCAM_BITDEPTH_14, QCARCAM_PACK_MIPI),
    QCARCAM_FMT_MIPIRAW_16 = QCARCAM_COLOR_FMT(QCARCAM_RAW,
QCARCAM_BITDEPTH_16, QCARCAM_PACK_MIPI),
    QCARCAM_FMT_MIPIRAW_20 = QCARCAM_COLOR_FMT(QCARCAM_RAW,
QCARCAM_BITDEPTH_20, QCARCAM_PACK_MIPI),
```

```
    QCARCAM_FMT_PLAIN16_10 = QCARCAM_COLOR_FMT(QCARCAM_RAW,
QCARCAM_BITDEPTH_10, QCARCAM_PACK_PLAIN16),
    QCARCAM_FMT_PLAIN16_12 = QCARCAM_COLOR_FMT(QCARCAM_RAW,
QCARCAM_BITDEPTH_12, QCARCAM_PACK_PLAIN16),
    QCARCAM_FMT_PLAIN16_14 = QCARCAM_COLOR_FMT(QCARCAM_RAW,
QCARCAM_BITDEPTH_14, QCARCAM_PACK_PLAIN16),
    QCARCAM_FMT_PLAIN16_16 = QCARCAM_COLOR_FMT(QCARCAM_RAW,
QCARCAM_BITDEPTH_16, QCARCAM_PACK_PLAIN16),
    QCARCAM_FMT_PLAIN32_20 = QCARCAM_COLOR_FMT(QCARCAM_RAW,
QCARCAM_BITDEPTH_20, QCARCAM_PACK_PLAIN32),
    QCARCAM_FMT_RGB_888 = QCARCAM_COLOR_FMT(QCARCAM_RGB,
QCARCAM_BITDEPTH_8, QCARCAM_PACK_FOURCC),

    QCARCAM_FMT_UYVY_8 = QCARCAM_COLOR_FMT(QCARCAM_YUV_UYVY,
QCARCAM_BITDEPTH_8, QCARCAM_PACK_FOURCC),
    QCARCAM_FMT_UYVY_10 = QCARCAM_COLOR_FMT(QCARCAM_YUV_UYVY,
QCARCAM_BITDEPTH_10, QCARCAM_PACK_FOURCC),
    QCARCAM_FMT_UYVY_12 = QCARCAM_COLOR_FMT(QCARCAM_YUV_UYVY,
QCARCAM_BITDEPTH_12, QCARCAM_PACK_FOURCC),
    QCARCAM_FMT_YUYV_8 = QCARCAM_COLOR_FMT(QCARCAM_YUV_YUYV,
QCARCAM_BITDEPTH_8, QCARCAM_PACK_FOURCC),
    QCARCAM_FMT_YUYV_10 = QCARCAM_COLOR_FMT(QCARCAM_YUV_YUYV,
QCARCAM_BITDEPTH_10, QCARCAM_PACK_FOURCC),
    QCARCAM_FMT_YUYV_12 = QCARCAM_COLOR_FMT(QCARCAM_YUV_YUYV,
QCARCAM_BITDEPTH_12, QCARCAM_PACK_FOURCC),

    QCARCAM_FMT_NV12 = QCARCAM_COLOR_FMT(QCARCAM_YUV_NV12,
QCARCAM_BITDEPTH_8, QCARCAM_PACK_FOURCC),
    QCARCAM_FMT_NV21 = QCARCAM_COLOR_FMT(QCARCAM_YUV_NV21,
QCARCAM_BITDEPTH_8, QCARCAM_PACK_FOURCC),

    QCARCAM_FMT_MAX = 0x7FFFFFFF
} qcarcam_color_fmt_t;
```

3.5 Resolution

To set the camera resolution, set the appropriate width and height. The maximum resolution size is defined by the maximum resolution of the camera sensor.

```
/// @brief Resolution
typedef struct
{
    unsigned int width;
    unsigned int height;
    float fps;
} qcarcam_res_t;
```

3.6 Input mode

The input mode is a combination of color format and resolution.

```
/// @brief Input Mode
typedef struct {
    qcarcam_color_fmt_t fmt;
    qcarcam_res_t res;
} qcarcam_mode_t;
```

3.7 QCarCam input description

NOTE: This definition will be deprecated. Please use newer v2 definition.

Table 3-2 lists the QCarCam inputs.

Table 3-2 QCarCam input definition

Field	Description
desc	Unique QCarCam input identifier
name	Input name (could be NULL)
parent_name	Parent name (could be NULL)
res	Supported resolutions
num_res	Number of valid entries in res array
color_fmt	Supported color formats
num_color_fmt	Number of valid entries in color_fmt array
flags	Bitmask of qcarcam_input_flag_t

```
/// @brief Input description flags
typedef enum
{
    QCARCAM_INPUT_FLAG_CONTENT_PROTECTED = 1 << 0,  ///< Content protection
enabled
    QCARCAM_INPUT_FLAG_PAIRED            = 1 << 1   ///< Paired input
stream
} qcarcam_input_flag_t;

/// @brief Input description
typedef struct
{
    qcarcam_input_desc_t desc;            ///< Unique input identifier

    char name[QCARCAM_INPUT_NAME_LEN];        ///< Input name. May be NULL
    char parent_name[QCARCAM_INPUT_NAME_LEN]; ///< Parent name. May be NULL

    qcarcam_res_t res[QCARCAM_MAX_NUM_RESOLUTIONS]; ///< Array of supported
resolutions in pixels
```

```
    unsigned int num_res;                                ///< Number of
supported resolutions

    qcarcam_color_fmt_t color_fmt[QCARCAM_MAX_NUM_COLOR_FMTS]; ///< Array
of supported color formats
    unsigned int num_color_fmt;                                  ///< Number
of supported color formats

    unsigned int flags; ///< bitmask of qcarcam_input_flag_t
} qcarcam_input_t;
```

3.8 QCarCam input description v2

NOTE: This replaces deprecated qcarcam_input_t definition

Table 3-3 lists the QCarCam v2 inputs.

Table 3-3 QCarCam input v2 definition

Field	Description
desc	The unique QCarCam input identifier (Section 3.3)
name	Input name
modes	Supported input modes (qcarcam_mode_t Section 3.6)
num_modes	Number of valid entries in modes array
flags	Bitmask of qcarcam_input_flag_t

```
/// @brief Input description flags
typedef enum
{
    QCARCAM_INPUT_FLAG_CONTENT_PROTECTED = 1 << 0,  ///< Content protection
enabled
    QCARCAM_INPUT_FLAG_PAIRED            = 1 << 1   ///< Paired input
stream
} qcarcam_input_flag_t;

/// @brief Input description
typedef struct
{
    qcarcam_input_desc_t desc;            ///< Unique input identifier

    char name[QCARCAM_INPUT_NAME_LEN];        ///< Input name

    qcarcam_mode_t modes[QCARCAM_MAX_NUM_MODES]; ///< Array of supported
modes
    unsigned int num_modes;                       ///< Number of supported
modes

    unsigned int flags; ///< bitmask of qcarcam_input_flag_t
```

```
} qcarcam_input_v2_t;
```

3.9 QCarCam buffer

NOTE: This definition will be deprecated. Please use newer v2 definition.

3.9.1 Buffer plane definitions

Buffer plane definitions include plane width, height, stride, and size.

- Width – plane width in pixels
- Height – plane height in pixels
- Stride – plane stride in bytes (Note: must be at least 16 byte aligned)
- Size – total plane size in bytes
- p_buf – pointer to plane memory (depending on the OS, this may be a file descriptor, private memory handle, or virtual address)

```
typedef struct
{
    unsigned int width;  ///< width in pixels
    unsigned int height; ///< height in pixels
    unsigned int stride; ///< stride in bytes
    unsigned int size;   ///< size in bytes
    void*        p_buf;
} qcarcam_plane_t;
```

3.9.2 Buffer definition

The buffer is defined by the number of planes and their definitions.

```
typedef struct
{
    qcarcam_plane_t planes[QCARCAM_MAX_NUM_PLANES]; ///< Array of planes
    unsigned int n_planes;                  ///< Number of planes in list
} qcarcam_buffer_t;
```

3.9.3 Set buffers definitions

A list of buffers is defined by the color format, the number of buffers, and their definitions.

```
/// @brief Buffer flag bits
typedef enum
{
    QCARCAM_BUFFER_FLAG_SECURE  = 1 << 0,   ///< buffer is secured
    QCARCAM_BUFFER_FLAG_CACHE   = 1 << 1,   ///< buffer is cached
```

```
    QCARCAM_BUFFER_FLAG_OS_HNDL = 1 << 4,   ///< buffer pointer refers to
an OS memory handle
} qcarcam_buffer_flag_t;

/// @brief Set buffer definition
typedef struct
{
    qcarcam_color_fmt_t color_fmt;

    qcarcam_buffer_t* buffers; ///< Array of buffers
    unsigned int n_buffers;    ///< Number of buffers in list

    unsigned int flags; ///< bitmask of qcarcam_buffer_flag_t
} qcarcam_buffers_t;
```

3.10 QCarCam buffer v2

NOTE: This replaces deprecated qcarcam buffer definitions.

3.10.1 Buffer plane v2 definitions

```
/// @brief Buffer plane definition
typedef struct
{
    unsigned int width;  ///< width in pixels
    unsigned int height; ///< height in pixels
    unsigned int stride; ///< stride in bytes
    unsigned int size;   ///< size in bytes
    unsigned long long hndl;  ///< buffer handle
    unsigned int offset; ///< Buffer offset for plane
} qcarcam_plane_v2_t;
```

Table 3-4 Buffer plane v2 definition

Field	Description
width	Width of plane in pixels
height	Height of plane in pixels
stride	Stride of plane in bytes
Size	Size of plane in bytes
hndl	Buffer memory handle (depending on OS, this could be a file descriptor, memory handle, or virtual address)
offset	Offset into buffer in bytes

3.10.2 Buffer v2 definition

Buffer is defined by its number of planes and their definitions as follows:

```
/// @brief Buffer definition
```

```
typedef struct
{
    qcarcam_plane_v2_t planes[QCARCAM_MAX_NUM_PLANES];
    unsigned int n_planes;
} qcarcam_buffer_v2_t;
```

3.10.3 Buffer list definition

A list of buffers is defined by the color format, the number of buffers, and their definitions.

```
/// @brief Buffer flag bits
typedef enum
{
    QCARCAM_BUFFER_FLAG_SECURE  = 1 << 0,   ///< buffer is secured
    QCARCAM_BUFFER_FLAG_CACHE   = 1 << 1,   ///< buffer is cached
    QCARCAM_BUFFER_FLAG_OS_HNDL = 1 << 4,   ///< buffer pointer refers to
an OS memory handle
} qcarcam_buffer_flag_t;

/// @brief buffer list definition
/// @note  used with qcarcam_s_buffers_v2
typedef struct
{
    unsigned int id;   ///< buffer list ID
    qcarcam_color_fmt_t color_fmt;

    qcarcam_buffer_v2_t* buffers; ///< Array of buffers
    unsigned int n_buffers;    ///< Number of buffers in list

    unsigned int flags; ///< bitmask of qcarcam_buffer_flag_t
} qcarcam_bufferlist_t;
```

3.11 Frame info definition

NOTE: This definition will be deprecated. Please use newer v2 definition.

The frame info definitions include buffer index, frame sequence number, flags, and timestamps.

```
/// @brief Frame done payload
typedef struct
{
    int idx; ///< index into the qcarcam_buffers_t buffers table
    unsigned int flags;
    unsigned int seq_no;
    unsigned long long timestamp;        ///SOF monotonic timestamp
    unsigned long long timestamp_system; ///system timestamp
```

```
    unsigned long long sof_qtimestamp;   //SOF HW timestamp
    qcarcam_field_t field_type;
} qcarcam_frame_info_t;
```

3.12 Frame info v2 definition

NOTE: This definition replaces qcarcam_frame_info_t.

The frame info definitions include bufferlist ID, buffer index, frame sequence number, and timestamps for each batched frame, interlace field type, and flags.

```
/// @brief Frame done payload v2
typedef struct{
   unsigned int id;  ///< buffer list ID
    unsigned int idx; ///< index into the qcarcam_buffers_t buffers table
    unsigned int flags;
    unsigned int seq_no[QCARCAM_MAX_BATCH_FRAMES]; ///< sequence number
(i.e. frame ID)
    unsigned long long timestamp;         ///< monotonic timestamp
    unsigned long long timestamp_system; ///< system timestamp
    unsigned long long sof_qtimestamp[QCARCAM_MAX_BATCH_FRAMES]; ///< sof
qtimer timestamp
    qcarcam_field_t field_type;
} qcarcam_frame_info_v2_t;
```

3.13 QCarCam event

3.13.1 Event type

```
typedef enum
{
    QCARCAM_EVENT_FRAME_READY = 1 << 0,    ///< Frame ready to be dequeued
using get_frame API
    QCARCAM_EVENT_INPUT_SIGNAL = 1 << 1,  ///< Payload will contain
qcarcam_input_signal_t
    QCARCAM_EVENT_ERROR = 1 << 2,         ///< Error event with
                                              qcarcam_event_error_t payload
    QCARCAM_EVENT_VENDOR = 1 << 3,        ///< Vendor event
    QCARCAM_EVENT_PROPERTY_NOTIFY = 1 << 4,   ///< Property events
    QCARCAM_EVENT_FRAME_SOF = 1 << 5,         ///< SOF event
    QCARCAM_EVENT_RECOVERY = 1 << 6,          ///< Recovery in
                                                   progress event
    QCARCAM_EVENT_RECOVERY_SUCCESS = 1 << 7,  ///< Recovery successful
                                                   event
    QCARCAM_EVENT_ERROR_ABORTED = 1 << 8,     ///< Recovery failed
                                                   event
    QCARCAM_EVENT_FRAME_FREEZE = 1 << 9       ///< Frozen frame event
```

```
                                                    With
                                                    qcarcam_frame_freeze_t
                                                    payload
    QCARCAM_EVENT_FRAME_DROP = 1 << 10        ///< Frame drop event
} qcarcam_event_t;
```

3.13.2 Event payload

The payload event is a union for holding possible values. The value is filled depending on the event type.

```
/// @brief Input Event payload definition
typedef enum
{
    QCARCAM_INPUT_SIGNAL_VALID = 0,
    QCARCAM_INPUT_SIGNAL_LOST
} qcarcam_input_signal_t;

/// @brief Error event payload definition
typedef enum
{
    QCARCAM_FATAL_ERROR = 0,
    QCARCAM_CONN_ERROR,
    QCARCAM_IFE_OVERFLOW_ERROR,
    QCARCAM_FRAMESYNC_ERROR
} qcarcam_event_error_t;

////////////////////////////////////////////////////////////////////////
/// @brief Union to hold possible values to p_payload in qcarcam_event_cb_t
///
///        EVENT ID                 |       TYPE          |    NOTE
/// ------------------------------------------------------------------------
/// QCARCAM_EVENT_FRAME_READY       | N/A                 |
/// QCARCAM_EVENT_INPUT_SIGNAL      | uint_payload        | qcarcam_input_signal_t
/// QCARCAM_EVENT_ERROR             | uint_paylaod        | qcarcam_event_error_t
/// QCARCAM_EVENT_VENDOR            | array               |
/// QCARCAM_EVENT_PROPERTY_NOTIFY   | uint_paylaod        |
/// QCARCAM_EVENT_FRAME_SOF         | qcarcam_timestamp_t | timestamps
/// QCARCAM_EVENT_RECOVERY          | uint_payload        | qcarcam_event_error_t
/// QCARCAM_EVENT_RECOVERY_SUCCESS  | N/A                     |
/// QCARCAM_EVENT_ERROR_ABORTED     | N/A                     |
/// QCARCAM_EVENT_FRAME_FREEZE      | qcarcam_frame_freeze_t  |
////////////////////////////////////////////////////////////////////////
typedef union
{
    unsigned int uint_payload;                  ///< unsigned int type
    qcarcam_timestamp_t sof_timestamp;          ///< SOF timestamp
    qcarcam_frame_freeze_t frame_freeze;        ///< Frame freeze
```

```
    qcarcam_vendor_param_t vendor_data;             ///< vendor data paylaod
    qcarcam_frame_info_v2_t frame_info;             ///< Frame info
    unsigned int array[QCARCAM_MAX_PAYLOAD_SIZE];   ///< max event payload
} qcarcam_event_payload_t;
```

3.14 Parameters

This section lists the currently-defined parameters to get or set.

3.14.1 Parameter type

See Sections 3.19 and 3.20 for more details.

Table 3-5 QCarCam parameter definitions

Parameter	Description	Type
QCARCAM_PARAM_EVENT_CB	Event callback function	ptr_value
QCARCAM_PARAM_EVENT_MASK	Bitmask of events (qcarcam_event_t) for which callback is enabled (Section 3.13)	uint_value
QCARCAM_PARAM_COLOR_FMT	**Will be deprecated** Sets Color format of the input source	color_value
QCARCAM_PARAM_RESOLUTION	**Will be deprecated** Sets Resolution of the input source	res_value
QCARCAM_PARAM_BRIGHTNESS	Sets Brightness	float_value
QCARCAM_PARAM_CONTRAST	Sets Contrast	float_value
QCARCAM_PARAM_MIRROR_H	Enable Horizontal Mirroring	uint_value
QCARCAM_PARAM_MIRROR_V	Enable Vertical Mirroring	uint_value
QCARCAM_PARAM_FRAME_RATE	Frame Rate control configuration can set frame drop pattern.	frame_rate_config
QCARCAM_PARAM_VID_STD	Set video standard **Note: Not implemented**	uint_value
QCARCAM_PARAM_CURRENT_VID_STD	Query current detected video standard **Note: Not implemented**	uint_value
QCARCAM_PARAM_STATUS	Query video lock status **Note: Not implemented**	qcarcam_input_signal_t
QCARCAM_PARAM_LATENCY_MAX	Max number of buffers that are ready for client to dequeue before buffers are dropped. Default: 1	uint_value
QCARCAM_PARAM_LATENCY_REDUCE_RATE	Sets the number of buffers that will be dropped once latency_max is exceeded. Dropped frames are internally requeued to be filled. A value of 0 means no frames will be dropped. Default: 1	uint_value
QCARCAM_PARAM_PRIVATE_DATA	Private data pointer that is stored and queried by client if needed	ptr_value
QCARCAM_PARAM_INJECTION_START	Starts frame processing for an injection buffer	uint_value

Parameter	Description	Type
QCARCAM_PARAM_EXPOSURE	Exposure configuration	exposure_config
QCARCAM_PARAM_HUE	Hue configuration	float_value
QCARCAM_PARAM_SATURATION	Saturation configuration	float_value
QCARCAM_PARAM_HDR_EXPOSURE	HDR Exposutre configuration	hdr_exposure_config
QCARCAM_PARAM_GAMMA	Sets gamma curve either as an exponent or a table of kneepoints	gamma_config
QCARCAM_PARAM_OPMODE	Sets operating mode of the pipeline	qcarcam_opmode_type
QCARCAM_PARAM_ISP_CTRLS	Sets ISP parameters	isp_ctrls
QCARCAM_PARAM_VENDOR	Vendor parameter that is passed through to the sensor library	
QCARCAM_PARAM_INPUT_MODE	Sets index of mode for input source	uint_value
QCARCAM_PARAM_MASTER	Sets client as master of the input source	uint_value
QCARCAM_PARAM_EVENT_CHANGE_SUBSCRIBE	Sets mask of events for which client will be notified in case they are modified	uint_value
QCARCAM_PARAM_EVENT_CHANGE_UNSUBSCRIBE	Sets mask of events to disable notification in case they are modified	uint_value
QCARCAM_PARAM_RECOVERY	Enable self-recovery	uint_value
QCARCAM_PARAM_BATCH_MODE	Batch mode configuration	batch_config
QCARCAM_PARAM_ISP_USECASE	Sets ISP node usecase	qcarcam_isp_usecase_t

```
/// @brief Parameter settings
typedef enum
{
    QCARCAM_PARAM_EVENT_CB = 0x1,      ///< Event callback function.
    QCARCAM_PARAM_EVENT_MASK,          ///< Mask of events
    QCARCAM_PARAM_COLOR_FMT,           ///< Output color format.
    QCARCAM_PARAM_RESOLUTION,          ///< Input dev resolution.
    QCARCAM_PARAM_BRIGHTNESS,
    QCARCAM_PARAM_CONTRAST,
    QCARCAM_PARAM_MIRROR_H,            ///< Horizontal mirror.
    QCARCAM_PARAM_MIRROR_V,            ///< Vertical mirror.
    QCARCAM_PARAM_FRAME_RATE,
    QCARCAM_PARAM_VID_STD,             ///< Video standard
    QCARCAM_PARAM_CURRENT_VID_STD,     ///< Video standard
    QCARCAM_PARAM_STATUS,              ///< Video lock status
    QCARCAM_PARAM_LATENCY_MAX,         ///< Max buffer latency in frame done Q
    QCARCAM_PARAM_LATENCY_REDUCE_RATE, ///< Number of buffers to drop when max latency
reached
    QCARCAM_PARAM_PRIVATE_DATA,
    QCARCAM_PARAM_INJECTION_START,
    QCARCAM_PARAM_EXPOSURE,            ///< exposure setting
    QCARCAM_PARAM_HUE,                 ///< hue setting
    QCARCAM_PARAM_SATURATION,          ///< saturation setting
    QCARCAM_PARAM_HDR_EXPOSURE,
    QCARCAM_PARAM_GAMMA,               ///< gamma setting
    QCARCAM_PARAM_OPMODE,              ///< operation mode
```

```
    QCARCAM_PARAM_ISP_CTRLS,            ///< ISP controls
    QCARCAM_PARAM_VENDOR,               ///< vendor param
    QCARCAM_PARAM_INPUT_MODE,           ///< Input device mode.
    QCARCAM_PARAM_MASTER,               ///< Set the client as master
    QCARCAM_PARAM_EVENT_CHANGE_SUBSCRIBE,      ///< Event subscription
    QCARCAM_PARAM_EVENT_CHANGE_UNSUBSCRIBE,    ///< Event unsubscribe
    QCARCAM_PARAM_RECOVERY,             ///< Should recovery mechanism be active or not.
    QCARCAM_PARAM_BATCH_MODE,           ///< Configures batch mode through
qcarcam_batch_mode_config_t
    QCARCAM_PARAM_ISP_USECASE,          ///< Configures ISP usecase type
    QCARCAM_PARAM_NUM,                  ///< total number of valid parameters.

    QCARCAM_PARAM_MAX = 0x7FFFFFFF
} qcarcam_param_t;
```

3.14.2 Parameter value

```
/// @brief Union to hold possible values to p_value in qcarcam_s_param and
qcarcam_g_param
typedef union
{
    void* ptr_value;                            ///< pointer type
    float float_value;                          ///< float type
    unsigned int uint_value;                    ///< unsigned int type
    qcarcam_res_t res_value;                    ///< resolution type
    qcarcam_color_fmt_t color_value;            ///< color type
    qcarcam_exposure_config_t exposure_config;  ///< Exposure settings
    qcarcam_hdr_exposure_config_t hdr_exposure_config; ///< HDR Exposure
settings
    qcarcam_gamma_config_t gamma_config;        ///< Gamma settings
    qcarcam_frame_rate_t frame_rate_config;     ///< Frame rate
settings
    qcarcam_param_isp_ctrls_t isp_ctrls;        ///< Used to control
isp sensor settings
    qcarcam_vendor_param_t vendor_param;        ///< vendor param
    qcarcam_batch_mode_config_t batch_config;   ///< batch mode config
    unsigned long long uint64_value;            ///< unsigned uint64
value
    qcarcam_isp_usecase_config_t isp_config;    ///< isp instance
config
    int arr_padding[QCARCAM_MAX_PAYLOAD_SIZE];  ///< Used to ensure
union size won't change
}qcarcam_param_value_t;
```

3.15 QCarCam functions

3.15.1 Initialize QCarCam library

The client must first initialize the QCarCam library using the qcarcam_initialize() function.

```
////////////////////////////////////////////////////////////////////////
////
/// qcarcam_initialize
///
/// @brief Initialize QCarCam. Must be first call to library.
///
/// @return QCARCAM_RET_OK if successful
////////////////////////////////////////////////////////////////////////
////
qcarcam_ret_t qcarcam_initialize(qcarcam_init_t* p_init_params);
```

The p_init_params argument can be optionally filled.

```
/// @brief Initialization parameters
typedef struct
{
    unsigned int flags;
    unsigned int version;
    const char* debug_tag;
    unsigned int reserved[4];
} qcarcam_init_t;
```

- version – QCarCam client API version
- debug_tag – client string to be used in debug logs

Example

```
qcarcam_init_t qcarcam_init = {};
qcarcam_init.version = QCARCAM_VERSION;
qcarcam_init.debug_tag = (char *)"qcarcam_test";

ret = qcarcam_initialize(&qcarcam_init);
if (ret != QCARCAM_RET_OK)
{
    QCARCAM_ERRORMSG("qcarcam_initialize failed %d", ret);
    exit(-1);
}
```

3.15.2 Uninitialize library

The last call to the library to uninitialize must be the qcarcam_uninitialize() function.

```
////////////////////////////////////////////////////////////////////////
////
/// qcarcam_uninitialize
///
/// @brief De-initialize QCarCam. Last call to library.
///
/// @return QCARCAM_RET_OK if successful
////////////////////////////////////////////////////////////////////////
////
qcarcam_ret_t qcarcam_uninitialize(void);
```

Example

```
qcarcam_uninitialize();
```

3.15.3 Query inputs

NOTE: qcarcam_query_inputs() is deprecated in favor of qcarcam_query_inputs_v2().

The qcarcam_query_inputs() and qcarcam_query_inputs_v2() function queries available inputs to the user.

The function returns QCARCAM_RET_BUSY if inputs are still being detected. The function returns QCARCAM_RET_OK if detection process has completed.

```
////////////////////////////////////////////////////////////////////////
////
/// qcarcam_query_inputs / qcarcam_query_inputs_v2
///
/// @brief Queries available inputs. To get the number of available inputs
to query, call with p_inputs set to NULL.
///
/// @param p_inputs   Pointer to array inputs. If NULL, then ret_size
returns number of available inputs to query
/// @param size       Number of elements in array
/// @param ret_size   If p_inputs is set, number of elements in array that
were filled
///                   If p_inputs is NULL, number of available inputs to
query
///
/// @return QCARCAM_RET_OK if successful.
///         QCARCAM_RET_BUSY if engine has not finished detection of all
available inputs. Will only return available
///                          inputs up to this point in time.
////////////////////////////////////////////////////////////////////////
////
```

```
qcarcam_ret_t qcarcam_query_inputs(qcarcam_input_t* p_inputs, unsigned int
size, unsigned int* ret_size);

qcarcam_ret_t qcarcam_query_inputs_v2(qcarcam_input_v2_t* p_inputs,
unsigned int size, unsigned int* ret_size);
```

Example

```
qcarcam_input_v2_t *pInputs;
unsigned int queryNumInputs = 0, queryFilled = 0;

//query number of available inputs
ret = qcarcam_query_inputs_v2(NULL, 0, &queryNumInputs);
if (QCARCAM_RET_OK != ret || queryNumInputs == 0)
{
    QCARCAM_ERRORMSG("Failed qcarcam_query_inputs number of inputs with
ret %d", ret);
}
else
{
    pInputs = (qcarcam_input_t *)calloc(queryNumInputs, sizeof(*pInputs));
    if (!pInputs)
    {
        QCARCAM_ERRORMSG("Failed to allocate pInputs");
        exit(-1);
    }

    ret = qcarcam_query_inputs_v2(pInputs, queryNumInputs, &queryFilled);
    if (QCARCAM_RET_OK != ret || queryFilled != queryNumInputs)
    {
        QCARCAM_ERRORMSG("Failed qcarcam_query_inputs with ret %d %d %d",
ret, queryFilled, queryNumInputs);
        exit(-1);
    }
}
```

3.15.4 Open

The qcarcam_open() function opens the handle to a camera input.

```
////////////////////////////////////////////////////////////////////////
/// qcarcam_open
///
/// @brief Opens handle to input
///
/// @param desc   Unique identifier of input to be opened
///
/// @return NOT NULL if successful; NULL on failure
```

```
////////////////////////////////////////////////////////////////////////////
////
qcarcam_hndl_t qcarcam_open(qcarcam_input_desc_t desc);
```

Example

```
qcarcam_hndl_t rvc_hndl = qcarcam_open(QCARCAM_INPUT_TYPE_RVC);
if (!rvc_hndl)
{
        QCARCAM_ERRORMSG("Failed to open RVC");
}
```

3.15.5 Close

The qcarcam_close() function closes the handle to a camera input.

```
////////////////////////////////////////////////////////////////////////////
////
/// qcarcam_close
///
/// @brief Closes handle to input
///
/// @param hndl   Handle of input that was opened
///
/// @return QCARCAM_RET_OK if successful
////////////////////////////////////////////////////////////////////////////
qcarcam_ret_t qcarcam_close(qcarcam_hndl_t hndl)
```

Example

```
qcarcam_ret_t ret = qcarcam_close(rvc_hndl);
```

3.15.6 Start

The qcarcam_start() function starts camera input processing.

```
////////////////////////////////////////////////////////////////////////////
////
/// qcarcam_start
///
/// @brief Start input
///
/// @param hndl       Handle of input
///
/// @return QCARCAM_RET_OK if successful
////////////////////////////////////////////////////////////////////////////
////
qcarcam_ret_t qcarcam_start(qcarcam_hndl_t hndl);
```

Example

```
qcarcam_ret_t ret = qcarcam_start(rvc_hndl);
```

3.15.7 Stop

Use the qcarcam_stop() function to stop input streaming and release its resources. This can be called from a started state or paused state.

```
////////////////////////////////////////////////////////////////////////
/// qcarcam_stop
///
/// @brief Stop input that was started
///
/// @param hndl       Handle of input
///
/// @return QCARCAM_RET_OK if successful
////////////////////////////////////////////////////////////////////////
////
qcarcam_ret_t qcarcam_stop(qcarcam_hndl_t hndl);
```

Example

```
qcarcam_ret_t ret = qcarcam_stop(rvc_hndl);
```

3.15.8 Pause

The qcarcam_pause() function pauses input streaming, but does not release resources. Resuming from this state is much quicker, as all resources are kept.

```
////////////////////////////////////////////////////////////////////////
////
/// qcarcam_pause
///
/// @brief Pause input that was started. Does not relinquish resource
///
/// @param hndl       Handle of input
///
/// @return QCARCAM_RET_OK if successful
////////////////////////////////////////////////////////////////////////
////
qcarcam_ret_t qcarcam_pause(qcarcam_hndl_t hndl);
```

Example

```
qcarcam_ret_t ret = qcarcam_pause(rvc_hndl);
```

3.15.9 Resume

The qcarcam_resume() function resumes camera input processing from a pause state. Because resources are held, the stream resume is quick.

```
////////////////////////////////////////////////////////////////////////
////
/// qcarcam_resume
///
/// @brief Resumes input that was paused
///
/// @param hndl       Handle of input
///
/// @return QCARCAM_RET_OK if successful
////////////////////////////////////////////////////////////////////////
////
qcarcam_ret_t qcarcam_resume(qcarcam_hndl_t hndl);
```

Example

```
qcarcam_ret_t ret = qcarcam_resume(rvc_hndl);
```

3.16 Frame processing

3.16.1 Set buffer list

NOTE: qcarcam_s_buffers() is deprecated in favor of qcarcam_s_buffers_v2().

The qcarcam_s_buffers() and qcarcam_s_buffers_v2() function sets buffer list to be consumed.

This is called prior to the stream start.

Calling it again unmaps previous buffers and use ones that are last set.

```
////////////////////////////////////////////////////////////////////////
////
/// qcarcam_s_buffers
///
/// @brief Set buffers
///
/// @param hndl       Handle of input
/// @param p_buffers  Pointer to set buffers structure
///
/// @return QCARCAM_RET_OK if successful
////////////////////////////////////////////////////////////////////////
////
qcarcam_ret_t qcarcam_s_buffers(qcarcam_hndl_t hndl, qcarcam_buffers_t*
p_buffers);
```

```
////////////////////////////////////////////////////////////////////////////
////
/// qcarcam_s_buffers_v2
///
/// @brief Set buffers for specifc buffer list
///
/// @param hndl       Handle of input
/// @param p_bufferlist  Pointer to bufferlist
///
/// @return QCARCAM_RET_OK if successful
////////////////////////////////////////////////////////////////////////////
////
qcarcam_ret_t qcarcam_s_buffers_v2(qcarcam_hndl_t hndl, const
qcarcam_bufferlist_t* p_bufferlist);
```

Example

```
//setting 3 1280x720 UYVY buffers with memory handles p_buf1, _buf2 and
p_buf3
qcarcam_buffelist_t bufferlist = {};
qcarcam_buffer_v2_t buffer[3] = {{1280, 720, 2560, 0xf0000, p_buf1, 0},
{{1280, 720, 2560, 0xf0000, p_buf2, 0}, {{1280, 720, 2560, 0xf0000, p_buf3,
0}};

bufferlist.id = 0;
bufferlist.color_fmt = QCARCAM_FMT_UYVY_8;
bufferlist.buffers = &buffer;
bufferlist.n_buffers = 3;
ret = qcarcam_s_buffers_v2(rvc_hndl, &bufferlist);
```

3.16.2 Get frame

NOTE: qcarcam_get_frame() is deprecated in favor of qcarcam_get_frame_v2().

The qcarcam_get_frame() and qcarcam_get_frame_v2() dequeue an available frame when it is ready. The call will block up to the timeout specified if a frame is not immediately available. A timeout value of 0 will return immediately.

The function shall return QCARCAM_RET_TIMEOUT if a frame is not ready by the expiration of the timeout value.

```
////////////////////////////////////////////////////////////////////////////
////
/// qcarcam_get_frame
///
/// @brief Get available frame
///
/// @param hndl          Handle of input
/// @param p_frame_info  Pointer to frame information that will be filled
```

```
/// @param timeout       Max wait time in ms for frame to be available
before timeout
/// @param flags         Flags
///
/// @return QCARCAM_RET_OK if successful; QCARCAM_RET_TIMEOUT if timeout
////////////////////////////////////////////////////////////////////////
////
qcarcam_ret_t qcarcam_get_frame(qcarcam_hndl_t hndl, qcarcam_frame_info_t*
p_frame_info,
        unsigned long long int timeout, unsigned int flags);

////////////////////////////////////////////////////////////////////////
////
/// qcarcam_get_frame_v2
///
/// @brief Get available frame
///
/// @param hndl          Handle of input
/// @param p_frame_info  Pointer to frame information that will be filled
/// @param timeout       Max wait time in ns for frame to be available
before timeout
/// @param flags         Flags
///
/// @return QCARCAM_RET_OK if successful; QCARCAM_RET_TIMEOUT if timeout
////////////////////////////////////////////////////////////////////////
////
qcarcam_ret_t qcarcam_get_frame_v2(qcarcam_hndl_t hndl,
qcarcam_frame_info_v2_t* p_frame_info,
        unsigned long long int timeout, unsigned int flags);
```

Example

```
qcarcam_ret_t ret = qcarcam_start(rvc_hndl);
…
while(running) {
   qcarcam_frame_info_v2_t frame_info;
   ret = qcarcam_get_frame_v2(rvc_hndl, &frame_info, TIMEOUT_INFINITE, 0);
   … //post to display or postprocess frames
   ret = qcarcam_release_frame_v2(rvc_hndl, frame_info.id, frame_info.idx);
}
```

3.16.3 Release frame

NOTE: qcarcam_release_frame () is deprecated in favor of qcarcam_release_frame_v2().

The qcarcam_release_frame() and qcarcam_release_frame_v2() functions release a frame buffer and re-enqueue it to QCarCam to be filled.

```
////////////////////////////////////////////////////////////////////////
////
/// qcarcam_release_frame
///
/// @brief Re-enqueue frame buffers
///
/// @param hndl       Handle of input
/// @param idx        Index into the qcarcam_buffers_t buffers table to
reenqueue
///
/// @return QCARCAM_RET_OK if successful
////////////////////////////////////////////////////////////////////////
////
qcarcam_ret_t qcarcam_release_frame(qcarcam_hndl_t hndl, unsigned int idx);

////////////////////////////////////////////////////////////////////////
/// qcarcam_release_frame_v2
///
/// @brief Re-enqueue frame buffers
///
/// @param hndl       Handle of input
/// @param id         bufferlist id
/// @param idx        Index into the qcarcam_buffers_t buffers table to
reenqueue
///
/// @return QCARCAM_RET_OK if successful
////////////////////////////////////////////////////////////////////////
////
qcarcam_ret_t qcarcam_release_frame_v2(qcarcam_hndl_t hndl, unsigned int
id, unsigned int idx);
```

Example

```
qcarcam_ret_t ret = qcarcam_start(rvc_hndl);
…
while(running) {
   qcarcam_frame_info_v2_t frame_info;
   ret = qcarcam_get_frame_v2(rvc_hndl, &frame_info, TIMEOUT_INFINITE, 0);
   … //post to display or postprocess frames
   ret = qcarcam_release_frame_v2(rvc_hndl, frame_info.id, frame_info.idx);
}
```

3.17 Get parameters

The qcarcam_g_param function gets parameters for the input handle. Retrieved parameters can include parameter definitions listed in Section 3.14.1.

```
////////////////////////////////////////////////////////////////////////
////
/// qcarcam_g_param
///
/// @brief Get parameter value
///
/// @param hndl     Handle of input
/// @param param    Parameter to get
/// @param p_value  Pointer to structure of value that will be retrieved
///
/// @return QCARCAM_RET_OK if successful
////////////////////////////////////////////////////////////////////////
////
qcarcam_ret_t qcarcam_g_param(qcarcam_hndl_t hndl, qcarcam_param_t param,
qcarcam_param_value_t* p_value);
```

Example

```
qcarcam_param_value_t brightness;
qcarcam_ret_t ret = qcarcam_g_param (rvc_hndl, QCARCAM_PARAM_BRIGHTNESS,
&brightness);
```

3.18 Set parameters

The qcarcam_s_param function sets a parameter for the input handle.

```
////////////////////////////////////////////////////////////////////////
////
/// qcarcam_s_param
///
/// @brief Set parameter
///
/// @param hndl     Handle of input
/// @param param    Parameter to set
/// @param p_value  Pointer to structure of value that will be set
///
/// @return QCARCAM_RET_OK if successful
////////////////////////////////////////////////////////////////////////
////
qcarcam_ret_t qcarcam_s_param(qcarcam_hndl_t hndl, qcarcam_param_t param,
const qcarcam_param_value_t* p_value);
```

Example

```
qcarcam_param_value_t brightness;
```

```
brightness.uint_value = 255;
ret = qcarcam_s_param(rvc_hndl, QCARCAM_PARAM_BRIGHTNESS, &brightness);
```

3.19 Internal ISP parameters

QCARCAM_PARAM_ISP_CTRLS is used to set/get internal ISP related parameters. Internal ISP parameters values are reset to default when qcarcam_open is enabled if qcarcam_start/qcarcam_stop will not reset the param values.

```
typedef enum
{
    QCARCAM_CONTROL_AE_LOCK = 0x0,
    QCARCAM_CONTROL_AE_MODE,
    QCARCAM_CONTROL_AWB_LOCK,
    QCARCAM_CONTROL_AWB_MODE,
    QCARCAM_CONTROL_EFFECT_MODE,
    QCARCAM_CONTROL_MODE,
    QCARCAM_CONTROL_SCENE_MODE,
    QCARCAM_CONTROL_AE_ANTIBANDING_MODE,
    QCARCAM_CONTROL_DUMP_FRAME,
    QCARCAM_CONTROL_CONTRAST_LEVEL,
    QCARCAM_CONTROL_SATURATION,
QCARCAM_CONTROL_AE_COMPENSATION,
QCARCAM_CONTROL_AE_REGIONS,
QCARCAM_ISP_PARAM_NUM /// total number of valid parameters.
}qcarcam_isp_param_t;

typedef struct
{
    unsigned long long int param_mask;  ///< Mask to indicate setting qcarc
am_isp_param_t
    qcarcam_ctrl_ae_lock_t ae_lock;
    qcarcam_ctrl_ae_mode_t ae_mode;
    qcarcam_ctrl_awb_lock_t awb_lock;
    qcarcam_ctrl_awb_mode_t awb_mode;
    qcarcam_ctrl_control_effect_mode_t effect_mode;
    qcarcam_ctrl_control_mode_t ctrl_mode;
    qcarcam_ctrl_control_scene_mode_t scene_mode;
    qcarcam_ctrl_ae_antibanding_mode_t ae_antibanding_mode;
    float contrast_level;
    float saturation;
    qcarcam_ctrl_gammainfo_t gammainfo;
   float ae_compensation;
   qcarcam_ctrl_ae_regions_t ae_regions;
}qcarcam_param_isp_ctrls_t;
```

3.19.1 Param_mask

Mask to indicate qcarcam_isp_param_t settings.

Example

```
SET_BIT(param->isp_ctrls.param_mask, QCARCAM_CONTROL_GAMMAINFO);
CHECK_BIT(param->isp_ctrls.param_mask, QCARCAM_CONTROL_GAMMAINFO);
```

3.19.2 AE_lock

Decides whether auto-exposure (AE) is currently locked to the latest calculated values.

When set to true (ON), the AE algorithm is locked to the latest parameters, and will not change exposure settings until the lock is set to false (OFF).

```
// CONTROL_AE_LOCK
typedef enum qcarcam_ctrl_ae_lock {
QCARCAM_CONTROL_AE_LOCK_OFF,
    QCARCAM_CONTROL_AE_LOCK_ON,
} qcarcam_ctrl_ae_lock_t;

Default value: QCARCAM_CONTROL_AE_LOCK_OFF
```

3.19.3 AE_mode

When set to AUTO mode, the camera device's AE routine is enabled. When set to MANUAL mode, the application's selected exposure time overwrites the algorithm output.

```
// CONTROL_AE_MODE
typedef enum qcarcam_ctrl_ae_mode {
    QCARCAM_CONTROL_AE_MODE_MANUAL,
    QCARCAM_CONTROL_AE_MODE_AUTO,
} qcarcam_ctrl_ae_mode_t;

Default value: QCARCAM_CONTROL_AE_MODE_AUTO
```

3.19.4 AWB_lock

Decides whether auto-white balance (AWB) is currently locked to the latest calculated values.

When set to true (ON), the AWB algorithm is locked to the latest parameters, and will not change color balance settings until the lock is set to false (OFF).

```
// CONTROL_AWB_LOCK
typedef enum qcarcam_ctrl_awb_lock {
```

```
    QCARCAM_CONTROL_AWB_LOCK_OFF,
    QCARCAM_CONTROL_AWB_LOCK_ON,
} qcarcam_ctrl_awb_lock_t;

Default value: QCARCAM_CONTROL_AWB_LOCK_OFF
```

3.19.5 AWB_mode

Decides whether AWB is currently setting the color transform fields, and what the illumination target is.

```
// CONTROL_AWB_MODE
typedef enum qcarcam_ctrl_awb_mode {
    QCARCAM_CONTROL_AWB_MODE_OFF,
    QCARCAM_CONTROL_AWB_MODE_AUTO,
    QCARCAM_CONTROL_AWB_MODE_INCANDESCENT,
    QCARCAM_CONTROL_AWB_MODE_FLUORESCENT,
    QCARCAM_CONTROL_AWB_MODE_WARM_FLUORESCENT,
    QCARCAM_CONTROL_AWB_MODE_DAYLIGHT,
    QCARCAM_CONTROL_AWB_MODE_CLOUDY_DAYLIGHT,
    QCARCAM_CONTROL_AWB_MODE_TWILIGHT,
    QCARCAM_CONTROL_AWB_MODE_SHADE,
} qcarcam_ctrl_awb_mode_t;

Default value: QCARCAM_CONTROL_AWB_MODE_AUTO
```

3.19.6 Effect_mode

A special color effect to apply.

```
// CONTROL_EFFECT_MODE
typedef enum qcarcam_ctrl_control_effect_mode {
    QCARCAM_CONTROL_EFFECT_MODE_OFF,
    QCARCAM_CONTROL_EFFECT_MODE_MONO,
    QCARCAM_CONTROL_EFFECT_MODE_NEGATIVE,
    QCARCAM_CONTROL_EFFECT_MODE_SOLARIZE,
    QCARCAM_CONTROL_EFFECT_MODE_SEPIA,
    QCARCAM_CONTROL_EFFECT_MODE_POSTERIZE,
    QCARCAM_CONTROL_EFFECT_MODE_WHITEBOARD,
    QCARCAM_CONTROL_EFFECT_MODE_BLACKBOARD,
    QCARCAM_CONTROL_EFFECT_MODE_AQUA,
} qcarcam_ctrl_control_effect_mode_t;

Default value: QCARCAM_CONTROL_EFFECT_MODE_OFF
```

3.19.7 Control_mode

Overall mode of 3A (AE, AWB, and auto-focus) control routines.

This is a top-level 3A control switch. When set to OFF, all 3A control by the camera device is disabled. The application must set the fields for capture parameters itself.

```
////////////////////////////////////////////////////////////////////////////////
              control_mode                |                 description

///  ------------------------------------|----------------------------------------
///   QCARCAM_CONTROL_MODE_OFF           | All control by the device's metering

///                                      | and 3A routines is disabled.

///                                      |

///   QCARCAM_CONTROL_MODE_ON            | Use settings for each individual 3A
                                           routine.
///                                      |

///   QCARCAM_CONTROL_MODE_USE_SCENE_MODE | Use a specific scene mode.

///                                      | Enabling this disables,

///                                      | (except for FACE_PRIORITY scene mode).

///                                      |

///   QCARCAM_CONTROL_MODE_OFF_KEEP_STATE | Same as OFF mode, except that this

///                                      | capture will not be used by camera
                                           device
///                                      | background 3A to update their
                                           statistics.
////////////////////////////////////////////////////////////////////////////////

typedef enum qcarcam_ctrl_control_mode {
    QCARCAM_CONTROL_MODE_OFF,
    QCARCAM_CONTROL_MODE_AUTO,
    QCARCAM_CONTROL_MODE_USE_SCENE_MODE,
    QCARCAM_CONTROL_MODE_OFF_KEEP_STATE,
} qcarcam_ctrl_control_mode_t;

Default value: QCARCAM_CONTROL_MODE_AUTO
```

3.19.8 Scene_mode

Control for the scene mode that is currently active.

Scene modes are custom camera modes optimized for a certain set of conditions and capture settings.

```
// CONTROL_SCENE_MODE
typedef enum qcarcam_ctrl_control_scene_mode {
    QCARCAM_CONTROL_SCENE_MODE_DISABLED                         = 0,
    QCARCAM_CONTROL_SCENE_MODE_FACE_PRIORITY,
    QCARCAM_CONTROL_SCENE_MODE_ACTION,
    QCARCAM_CONTROL_SCENE_MODE_PORTRAIT,
    QCARCAM_CONTROL_SCENE_MODE_LANDSCAPE,
    QCARCAM_CONTROL_SCENE_MODE_NIGHT,
    QCARCAM_CONTROL_SCENE_MODE_NIGHT_PORTRAIT,
    QCARCAM_CONTROL_SCENE_MODE_THEATRE,
    QCARCAM_CONTROL_SCENE_MODE_BEACH,
    QCARCAM_CONTROL_SCENE_MODE_SNOW,
    QCARCAM_CONTROL_SCENE_MODE_SUNSET,
    QCARCAM_CONTROL_SCENE_MODE_STEADYPHOTO,
    QCARCAM_CONTROL_SCENE_MODE_FIREWORKS,
    QCARCAM_CONTROL_SCENE_MODE_SPORTS,
    QCARCAM_CONTROL_SCENE_MODE_PARTY,
    QCARCAM_CONTROL_SCENE_MODE_CANDLELIGHT,
    QCARCAM_CONTROL_SCENE_MODE_BARCODE,
    QCARCAM_CONTROL_SCENE_MODE_HIGH_SPEED_VIDEO,
    QCARCAM_CONTROL_SCENE_MODE_HDR,
    QCARCAM_CONTROL_SCENE_MODE_FACE_PRIORITY_LOW_LIGHT,
    QCARCAM_CONTROL_SCENE_MODE_DEVICE_CUSTOM_START              = 100,
    QCARCAM_CONTROL_SCENE_MODE_DEVICE_CUSTOM_END                = 127,
} qcarcam_ctrl_control_scene_mode_t;

Default value: QCARCAM_CONTROL_SCENE_MODE_FACE_PRIORITY
```

3.19.9 AE_antibanding_mode

The preferred setting for the camera device's AE algorithm's antibanding compensation.

```
// CONTROL_AE_ANTIBANDING_MODE
typedef enum qcarcam_ctrl_ae_antibanding_mode {
    QCARCAM_CONTROL_AE_ANTIBANDING_MODE_OFF,
    QCARCAM_CONTROL_AE_ANTIBANDING_MODE_50HZ,
    QCARCAM_CONTROL_AE_ANTIBANDING_MODE_60HZ,
    QCARCAM_CONTROL_AE_ANTIBANDING_MODE_AUTO,
} qcarcam_ctrl_ae_antibanding_mode_t;

Default value: QCARCAM_CONTROL_AE_ANTIBANDING_MODE_AUTO
```

3.19.10 AE_regions

Metering areas to use for auto-exposure adjustment.

```
// QCARCAM_CONTROL_AE_REGIONS
typedef struct
{
int xMin;         ///< Top-left X-coordinate
int yMin;         ///< Top-left y-coordinate
int xMax;         ///< Bottom-right x-coordinate
int yMax;         ///< Bottom-right y-coordinate
int weight;       ///< Weight of the region, The weight must be within [0,
1000], and represents a weight for every pixel in the area
} qcarcam_ctrl_ae_regions_t;

Default values: xMin = 0; yMin = 0; xMax = 0, yMax = 0, weight = 0
```

Note: The coordinate should be within the active image.

3.20 Other parameters

NOTE: Numerous changes were made in this section.

3.20.1 Operation mode

`qcarcam_opmode_type` defines the operation mode and processing chain for the input. The following is a description of each processing chain:

```
/// @brief Input operation modes
typedef enum {
    QCARCAM_OPMODE_RAW_DUMP,
    QCARCAM_OPMODE_SHDR,
    QCARCAM_OPMODE_INJECT,
    QCARCAM_OPMODE_PAIRED_INPUT,
    QCARCAM_OPMODE_DEINTERLACE,
    QCARCAM_OPMODE_TRANSFORMER,
    QCARCAM_OPMODE_RGBIR,
    QCARCAM_OPMODE_ISP,
    QCARCAM_OPMODE_2_STREAMS,

    QCARCAM_OPMODE_MAX
} qcarcam_opmode_type;
```

3.20.1.1 QCARCAM_OPMODE_RAW_DUMP

The simplest operation mode is a RAW dump where the data is written out to the client buffer as-is from the sensor.

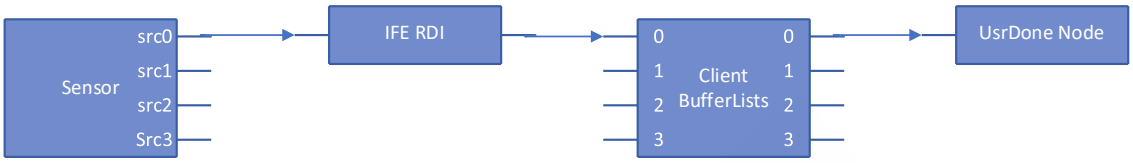


Figure 3-2 Graph of RAW dump operation mode

3.20.1.2 QCARCAM_OPMODE_ISP

The ISP processing chain has the sensor data first written out to an internal buffer. That internal buffer is then processed through the ISP to output an NV12 image to the client buffer. A JPEG image is written out to an internal buffer for debugging.

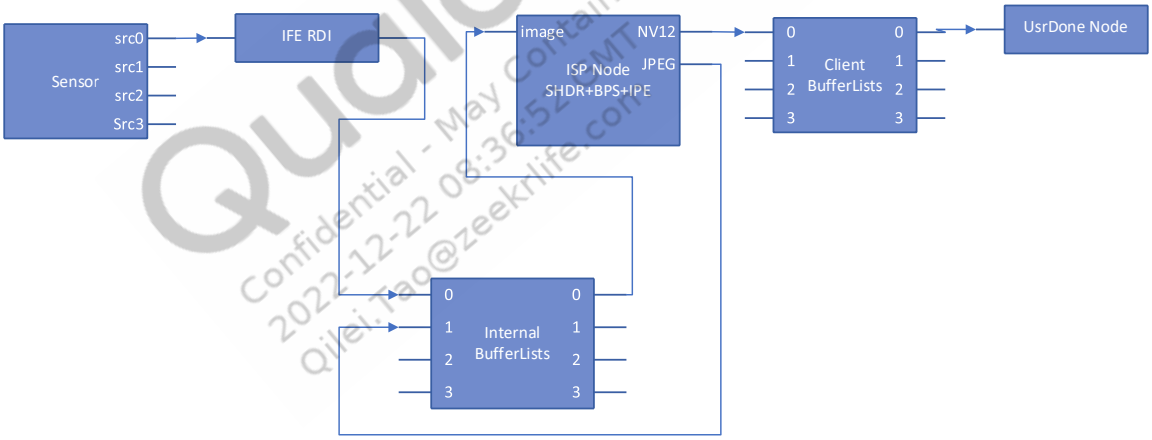


Figure 3-3 Graph of ISP processing operation mode

The ISP node usecase is set by qcarcam_isp_usecase_t with the QCARCAM_PARAM_ISP_USECASE parameter. By default it is set to the QCARCAM_ISP_USECASE_SHDR_BPS_IPE_AEC_AWB.

3.20.1.3 QCARCAM_OPMODE_SHDR

The SHDR processing chain is a special case of the QCARCAM_OPMODE_ISP operating mode and it is equivalent to QCARCAM_OPMODE_ISP + QCARCAM_ISP_USECASE_SHDR_BPS_IPE_AEC_AWB.

3.20.1.4 QCARCAM_OPMODE_PAIRED_INPUT

The paired operating mode processing chain is designed for bonded CSI inputs to have the data recombined into a single buffer. The two streams are written out to the same client buffer byt the IFE (1 RDI for left stream and another for the right stream). The framesync node perfoms a check against the timestamps of the two streams to ensure they belong to the same frame in case the processing goes out of sync for the streams.

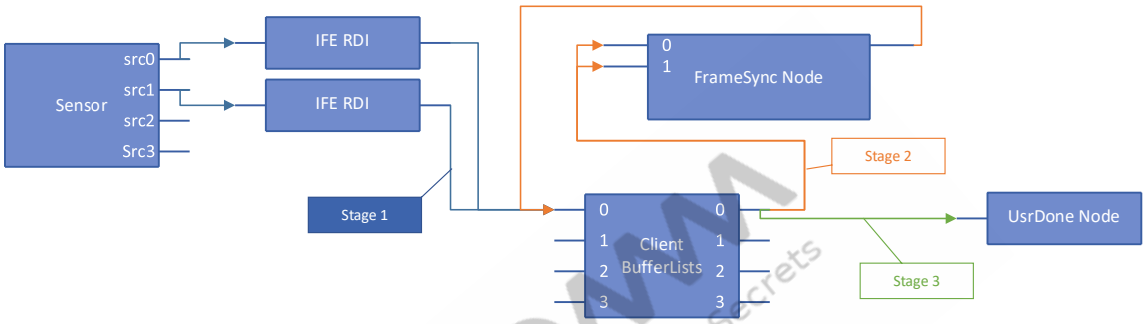


Figure 3-4 Graph of paired input operation mode

3.20.1.5 QCARCAM_OPMODE_DEINTERLACE

The deinterlace operating mode utilizes the GPU to perform a deinterlace operation. The input from the sensor is written out to an internal buffer that is consumed by the GPU node. The deinterlaced image is written out to the client buffer.

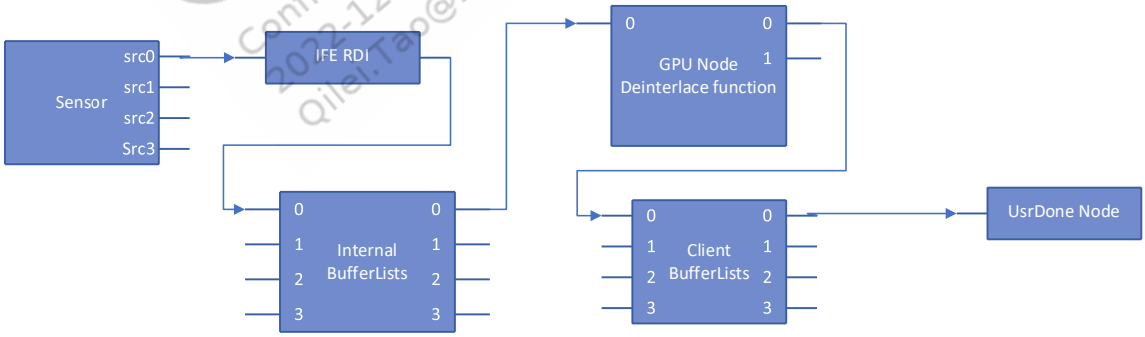


Figure 3-5 Graph of deinterlace operation mode

3.20.1.6 QCARCAM_OPMODE_TRANSFORMER

The transformer operating mode utilizes the GPU to perform a variety of image transformations, such as color conversion or scaling based on the input color format defined by the sensor and the output color format defined by the client buffer. The input from the sensor is written out to an internal buffer that is consumed by the GPU node to generate an output image into the client buffer.

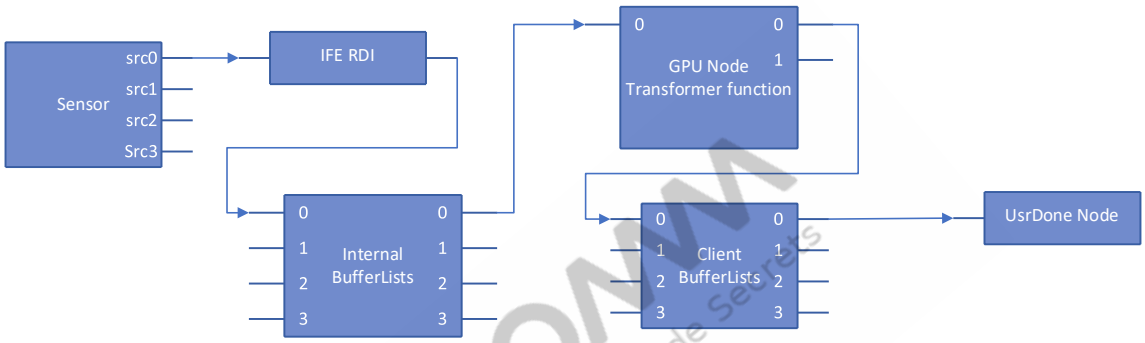


Figure 3-6 Graph of transformer operation mode

3.20.1.7 QCARCAM_OPMODE_RGBIR

The RGBIR processing chain is defined to allow preprocessing of RGBIR, where it first writes out the input stream to an internal buffer.

The buffer is then consumed by an RGBIR node that processes the input to produce three buffers corresponding to the RGB, IR, and full scale IR images. The RGB and IR images are then processed by the ISP node to output NV12 images to the client buffers.

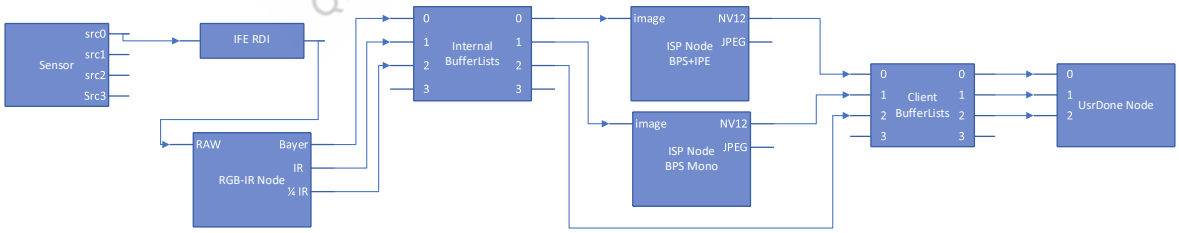


Figure 3-7 Graph of RGBIR operation mode

3.20.1.8 QCARCAM_OPMODE_2_STREAMS

The two streams processing chain is to perform a raw dump of two sensor streams to client buffers.

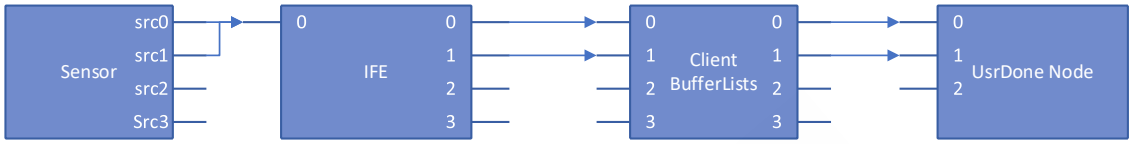


Figure 3-8 Graph of two streams operation mode

3.20.1.9 QCARCAM_OPMODE_RDI_CONVERSION

The RDI conversion operation mode provides the ability to perform a color conversion from YUV422 to NV12 utilizing the native hardware capabilities. The YUV422 input is sent across two RDIs. One RDI will output only the Y plane. The other RDI will output only the downsampled UV plane into the same buffer. There is a framesync node that ensures both planes belong to the same frame within the client buffer.

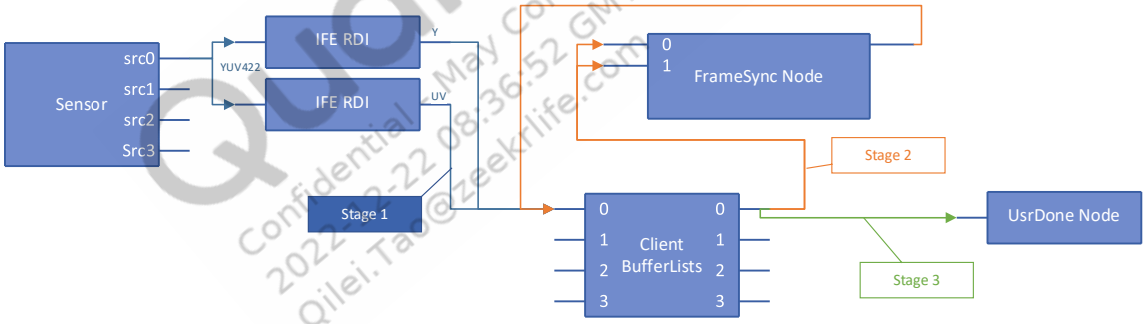


Figure 3-9 Graph of RDI conversion operation mode

3.20.2 Contrast level

Contrast is defined as the separation between the darkest and brightest areas of the image. Increasing the contrast increases the separation between dark and bright, making shadows darker and highlights brighter. Decreasing the contrast brings the shadows up and the highlights down to make them closer to one another.

```
// QCARCAM_CONTROL_CONTRAST_LEVEL
Range: [-1.0, 1.0];
Default value : 0.0
```

3.20.3 Saturation

Saturation increases the separation between colors. The reference step for test is 0.2.

```
// QCARCAM_CONTROL_SATURATION
Range: [-1.0, 1.0];
Default value: 0.0
```

3.20.4 Brightness

Exposure compensation is used to alter exposure from the value selected by the camera, making the image brighter or darker. The reference step for test is 0.1.

```
// QCARCAM_CONTROL_BRIGHTNESS
Range: [-1.0, 1.0];
Default value: 0.0
```

3.20.5 Hue

```
// QCARCAM_PARAM_HUE
Range: [-1.0, 1.0];
Default value: 0.0
```

3.20.6 Saturation

```
// QCARCAM_PARAM_SATURATION
Range: [-1.0, 1.0];
Default value: 0.0
```

3.20.7 Gamma config

```
// QCARCAM_PARAM_GAMMA

/// @brief structure to hold manual gamma parameters
typedef struct
{
    qcarcam_gamma_type_t config_type;  ///< gamma configure mode
    union {
        float f_value;
        struct {
            unsigned int length;
            unsigned int *p_value;
        } table;
    } gamma;
}qcarcam_gamma_config_t;
```

There are two types of setting gamma table (qcarcam_gamma_type_t) : [1] QCARCAM_GAMMA_EXPONENT and [2] QCARCAM_GAMMA_KNEEPOINTS.

1. QCARCAM_GAMMA_EXPONENT – A positive float value is passed to be set as an exponent of the gamma curve.

2. QCARCAM_GAMMA_KNEEPOINTS – A table of knee points is set for the gamma curve.

3.20.8 Exposure config

```
// QCARCAM_PARAM_EXPOSURE
// QCARCAM_PARAM_HDR_EXPOSURE
typedef enum  {
    QCARCAM_EXPOSURE_AUTO,
    QCARCAM_EXPOSURE_MANUAL,
    QCARCAM_EXPOSURE_SEMI_AUTO,
    QCARCAM_EXPOSURE_LUX_IDX
} qcarcam_exposure_mode_t;

/// @brief structure to hold manual exposure parameters
typedef struct
{
    qcarcam_exposure_mode_t exposure_mode_type;
    float exposure_time;   ///< time in ms
    float gain;            ///< 1.0 to Max supported in sensor
    float target;          ///< for SEMI_AUTO mode
    float lux_index;       ///< for LUX_IDX mode
    unsigned int reserved[4];    ///< extra params that may be needed
}qcarcam_exposure_config_t;
```

1. QCARCAM_EXPOSURE_AUTO – Automatic exposure control by AEC algorithm.
2. QCARCAM_EXPOSURE_MANUAL – Manual exposure setting parameters for exposure time and gain.
3. QCARCAM_EXPOSURE_SEMI_AUTO – Specify the exposure target.
4. QCARCAM_EXPOSURE_LUX_IDX – Set the lux index.

The default type is QCARCAM_EXPOSURE_AUTO (automatic exposure). Sensor library implementation may support one or more of these modes.

3.20.9 HDR exposure config

```
/// @brief structure to hold manual exposure parameters
typedef struct
{
    qcarcam_exposure_mode_t exposure_mode_type;
    unsigned int hdr_mode;
    unsigned int num_exposures;
    float exposure_time[QCARCAM_HDR_NUM_EXPOSURES];  ///< time in ms
    float exposure_ratio[QCARCAM_HDR_NUM_EXPOSURES];
    float gain[QCARCAM_HDR_NUM_EXPOSURES];           ///< 1.0 to Max
supported in sensor
    float target;           ///< for SEMI_AUTO mode
```

```
    float lux_index;        ///< for LUX_IDX mode
}qcarcam_hdr_exposure_config_t;
```

1. QCARCAM_EXPOSURE_AUTO – Automatic exposure control by AEC algorithm.
2. QCARCAM_EXPOSURE_MANUAL – Manual exposure setting parameters for exposure time and gain.
3. QCARCAM_EXPOSURE_SEMI_AUTO – Specify the exposure target.
4. QCARCAM_EXPOSURE_LUX_IDX – Set the lux index.

The default type is QCARCAM_EXPOSURE_AUTO (automatic exposure). Sensor library implementation may support one or more of these modes.

3.20.10 Vendor parameter

Vendor parameters are used by OEMs for custom parameter settings in their sensor library. QCARCAM_PRAM_VENDOR provides a data array of 64 unsigned integers (256 bytes) that can be used for this purpose.

```
/// @brief structure to hold vendor param and vendor event payload
typedef struct
{
    unsigned int data[QCARCAM_MAX_VENDOR_PAYLOAD_SIZE];
}qcarcam_vendor_param_t;
```

3.20.11 Frame rate config

The frame rate config parameter, QCARCAM_PARAM_FRAME_RATE, allows the client to control the frame rate.

```
/// @brief Frame drop modes
typedef enum  {
    QCARCAM_KEEP_ALL_FRAMES,    ///< Max fps
    QCARCAM_KEEP_EVERY_2FRAMES, ///< 1/2 Max fps
    QCARCAM_KEEP_EVERY_3FRAMES, ///< 1/3 Max fps
    QCARCAM_KEEP_EVERY_4FRAMES, ///< 1/4 Max fps
    QCARCAM_DROP_ALL_FRAMES,    ///< 0 fps
    QCARCAM_FRAMEDROP_MANUAL    /// Set period/pattern manually
} qcarcam_frame_drop_mode_t;

/// @brief structure to hold frame rate parameters
typedef struct
{
    qcarcam_frame_drop_mode_t frame_drop_mode;
    unsigned char frame_drop_period;   ///< only effective when
frame_drop_mode = QCARCAM_FRAMEDROP_MANUAL, max value 31
```

```
    unsigned int frame_drop_pattern;   ///< only effective when
frame_drop_mode = QCARCAM_FRAMEDROP_MANUAL
}qcarcam_frame_rate_t;
```

3.20.12 Batch mode config

Batch mode configuration allows the client to batch multiple frames together into a single buffer.

```
/// @brief Batch mode types
typedef enum
{
    QCARCAM_BATCH_MODE_DEFAULT = 0,  ///< frame info filled for each
batched frame
}qcarcam_batch_mode_type_t;

/// @brief Batch mode configuration
typedef struct
{
    qcarcam_batch_mode_type_t batch_mode;
    unsigned int num_batch_frames;
    unsigned int frame_increment;  ///< offset in bytes frame N first
pixel to frame N+1
    unsigned int detect_first_phase_timer;
}qcarcam_batch_mode_config_t;
```

3.20.13 ISP usecase config

QCARCAM_PARAM_ISP_USECASE allows the client to define which ISP usecase and camera ID for tuning is used for each ISP instance.

```
/// @brief ISP instance config parameters
typedef struct {
    unsigned int  id;                  ///< ISP instance id
    unsigned int camera_id;            ///< ISP camera id
    qcarcam_isp_usecase_t use_case;    ///< ISP use case
}qcarcam_isp_usecase_config_t;
```

4 Call flow

Figure 4-1 and Figure 4-2 show example call flows using QCarCam API with event callback or with polling for frames.

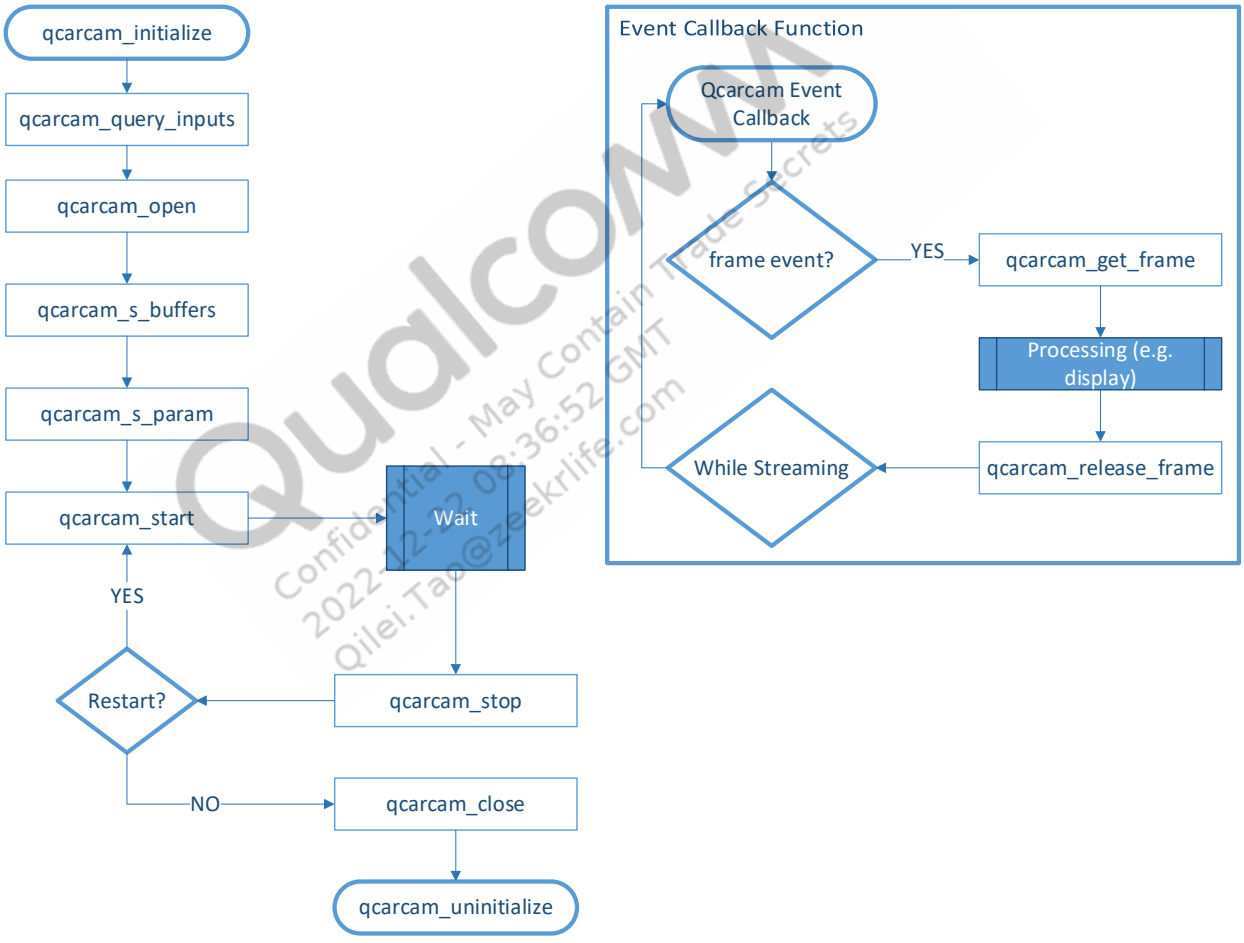


Figure 4-1 QCarCam call flow with event callback

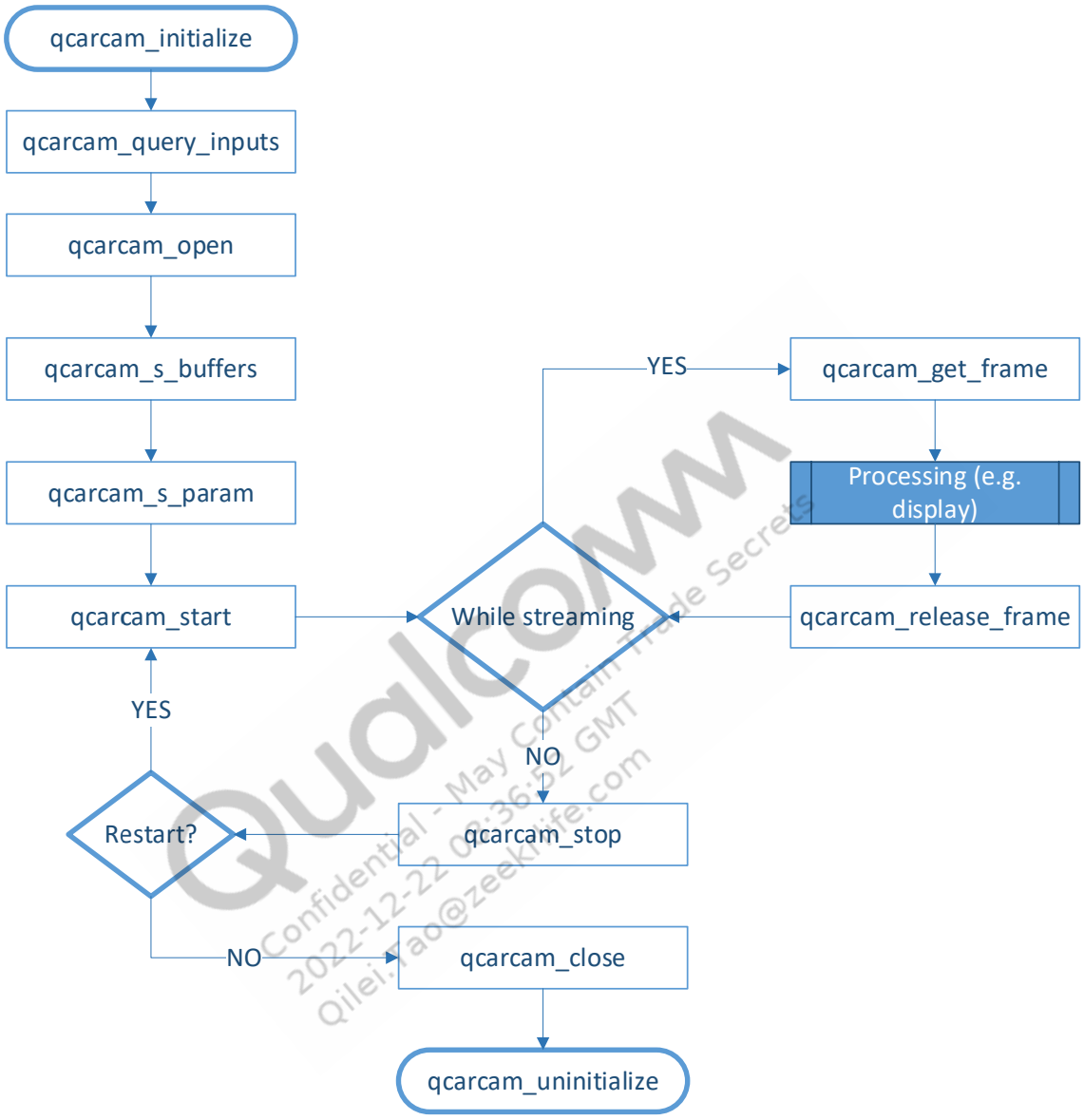


Figure 4-2 QCarCam call flow with polling

5 Demo applications

For reference on how the QCarCam API can be used, refer to the qcarcam_test demo application included in the build.

The qcarcam_test native demo application is a cross-OS development application to easily test multiple cameras and render them on display. The OS-specific buffer and windowing functions are abstracted in the test_util library.

See *SA6155/SA8155 Automotive Camera AIS Customization Guide* (80-PG469-93) for more information on the XML and command line arguments.

5.1 Android

To run the demo application on Android:

```
adb root
adb shell

#Run ais_server if it is not running by default
ais_server &

#one camera test for camera at channel 0
qcarcam_test -config=/vendor/bin/1cam.xml
```

5.2 AGL

To run the demo application on AGL:

```
#Run ais_server if it is not running by default
ais_server &

cd /usr/bin/
#one camera test for camera at channel 0
qcarcam_test -config=/vendor/bin/1cam.xml
```

5.3 QNX

To run the demo application on QNX:

```
#Run ais_server if it is not running by default
ais_server &

cd bin/camera/qcarcam_test
// one camera test for camera at channel 0
./qcarcam_test -config=1cam.xml
```