Automotive Camera Architecture on Hypervisor

80-16205-1 Rev. AA

July 15, 2020

Revision history

Revision	Date	Description
AA	July 2020	Initial release

Note: There is no Rev. I, O, Q, S, X, or Z per Mil. standards.

Contents

Figures

Tables

1 Introduction

1.1 Purpose

This document describes how to configure the automotive imaging system (AIS) on Hypervisor automotive platforms. This document details the following:

- AIS architecture
- Data flow between AIS client and server running on different guest virtual machines (GVMs) and physical virtual machines (PVMs)
- Camera driver porting and source files
- 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

AIS is designed to manage video input streams, such as camera and High Definition Multimedia Interface (HDMI). AIS is optimized for the Qualcomm® Snapdragon™ 820 Automotive chipset.

AIS interacts with various imaging inputs, including:

- Mobile Industry Processor Interface (MIPI) sensors
- Bridge chips
- Composite Video Broadcast Signal (CVBS)/HDMI
- Flat Panel Display Link (FPD-Link)

AIS manages memory buffers under the AIS framework.

AIS sets access levels for specific users and applications so that any camera input can become privileged. By using this feature, multiple users can simultaneously access the same camera input. Users can also access non-camera input sources, such as, HDMI and video input.

2.1 Features

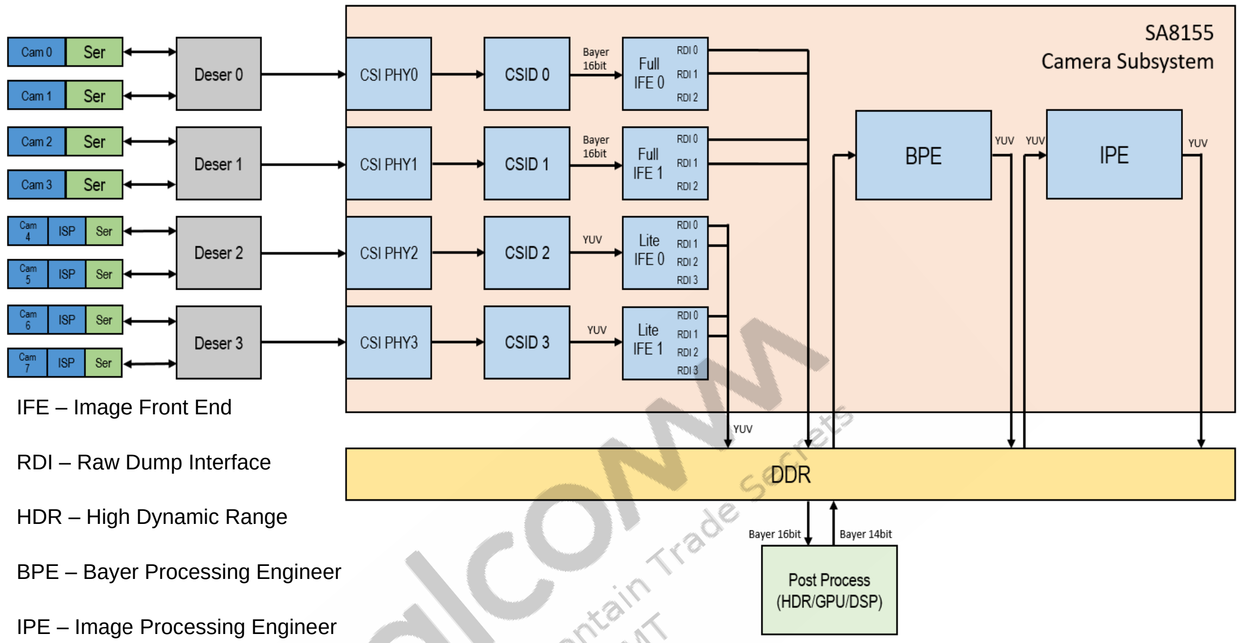
AIS supports the following features:

- OS portability
- Multi-client support
- Input permissions
- Bridge chip abstraction for entity access
- Hardware assisted split stream
- Zero buffer copy
- Frame transformation
- Frame processing
- Up to eight streams per session

2.2 Camera hardware architecture

Figure 2-1 shows the SA8155 camera hardware architecture. SA8155 supports four Camera Serial Interface (CSI) input ports with up to four lanes for each. Each port connects to a camera or bridge chip. CSI0-4 connects to a bridge chip.

The bridge chip collects two camera data streams respectively to one CSI port through MIPI lanes. Figure 2-1 shows four Bayer camera sensors and four YUV camera sensors.



IFE – Image Front End

RDI – Raw Dump Interface

HDR – High Dynamic Range

BPE – Bayer Processing Engineer

IPE – Image Processing Engineer

Figure 2-1 Typical camera hardware architecture

2.3 AIS software architecture

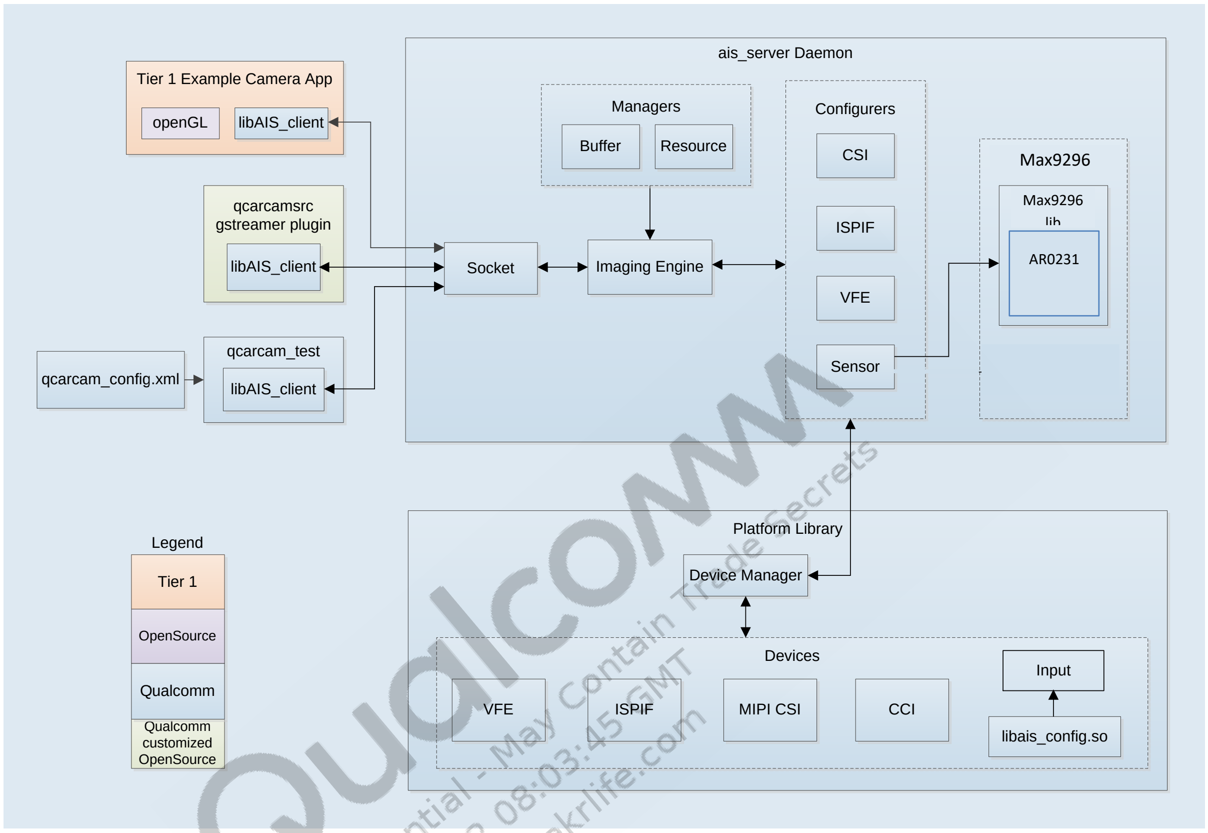


Figure 2-2 AIS software architecture

Figure 2-2 shows that the AIS server runs on user space as a daemon. The server communicates with kernel space hardware drivers through platform-related libraries. The server then executes most management tasks.

AIS clients use the socket protocol to communicate with the AIS server to query camera information, control the camera handle, and retrieve camera data.

3 PVM/Hypervisor/GVM solution

3.1 High-level software design

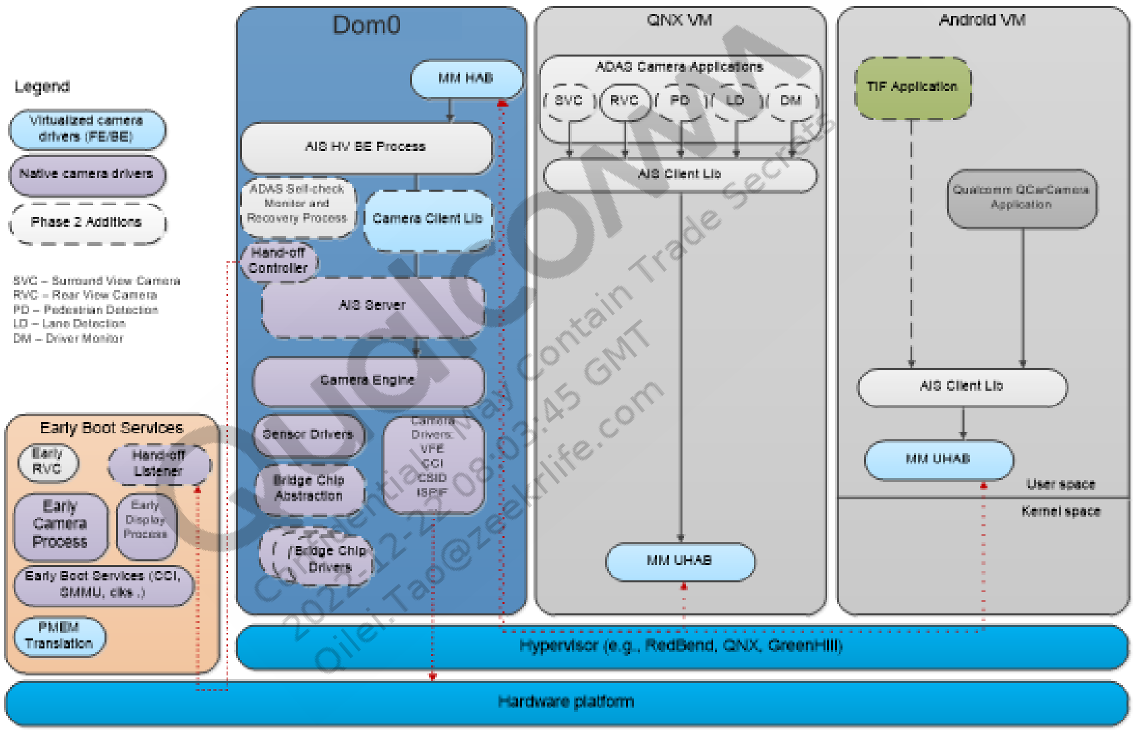


Figure 3-1 Camera virtualization with RTOS PVM and Android and QNX GVM

Camera back-end and front-end communication depends on mmHAB functions (open, close, send, receive, export, and import) on both GVM and on PVM.

- The camera back-end driver depends on the Real Time Operating System (RTOS) camera driver and firmware.
- The camera front-end driver depends on high-level operating system memory.
  - Examples: IP Service Over AllJoyn in Linux Android, Generic Buffer Management in Linux Vehicle, persistent memory in QNX, and Green Hills (GHS) for memory allocation and sharing with back-end driver.
- The camera front-end driver depends on the native user mode media framework implemented within AIS.

3.2 Sequence of camera Hypervisor communication call

The RTOS camera back-end server is a 64-bit application that keeps listening to any front-end connection request. It calls the Dom 0 Hypervisor Abstract Communication driver (HAB) using `habmm_socket_open()` with the MM_CAMERA tag.

The camera Hypervisor communication call sequence is as follows:

1. HAB abstracts the details of the Hypervisor connection, send, and receive functions. The call waits until HAB detects a camera front-end connection request.

   The front-end connection request is caused by `habmm_socket_open()` with the MM_CAMERA tag.
2. Upon receiving the “be virtual channel address” from `habmm_socket_open()`, the camera back-end Master creates a thread to handle the client requests for this session.
3. Once the communication channel is up, the front-end sends messages to the back-end process and then blocks to wait for the return value.
4. The back-end process receives the Hypervisor message and translates the message into the AIS client calls to the local AIS server for processing. This is the same as the normal call process from a local client.
5. The call returns parameters from camera IP to the back-end process. The parameters are translated into Hypervisor messages before being sent back to the front-end.
6. The front-end translates the returned parameters from the Hypervisor message to the return data to the caller, using the QCarCam API that the clients use.
7. At the end of the communication, the front-end synchronizes with the back-end process to close the virtual channel using the `habmm_socket_close()`.
8. The back-end thread exits and cleans up any leftover items.

   The original back-end process keeps watching for any new connections.

3.2.1 API

The API is the same for applications either in Hypervisor or in metal.

Back-end in a client of the AIS server is the same as in any other client.

3.3 Communication implementation

3.3.1 Hyp_msg definition

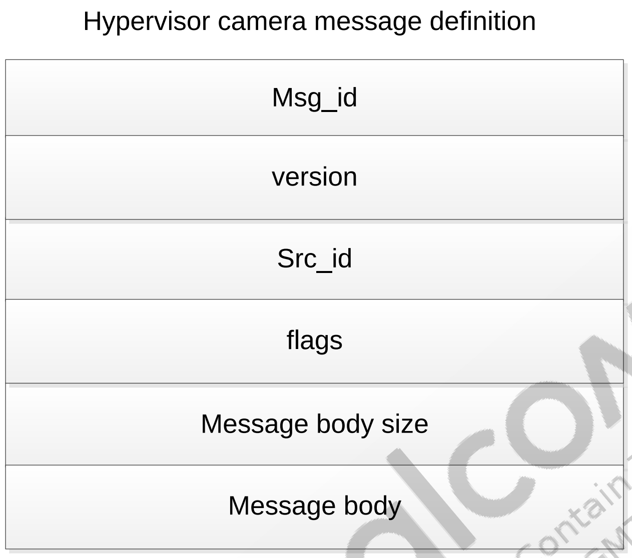


Figure 3-2 Hypervisor camera message

3.3.2 Hypervisor command message structure

Table 3-1 Hypervisor command message structure

hypcamera_msg_id_type	Description	Details

`CAM_MSG_OPEN`	Opens a camera or other input devices	
		```
		typedef struct
		{
		    /**< The input desc to open.  This was returned by query input call */
		    qcarcam_input_desc_t desc;
		}cam_vm_cmd_open_t;
		```

`CAM_MSG_QUERY_INPUTS`	Queries what inputs are available	
		```
		typedef struct
		{
		    /**< flag to indicate if p_inputs was set */
		    unsigned int is_p_inputs_set;

		    /**< Number of elements in input structure */
		    unsigned int size;
		}cam_vm_cmd_query_inputs_t;
		```

hypcamera_msg_id_type	Description	Details
`CAM_MSG_GET_PARM`	Gets a parameter	–
`CAM_MSG_SET_PARM`	Sets a parameter	–

`CAM_MSG_SET_BUFFERS`	Sets the buffer list to write camera data into	`typedef struct cam_vm_cmd_s_buffer_t`
		`{`
		`/**< handle returned during open*/`
		`qcarcam_hndl_t hndl;`
		
		`/**< color format*/`
		`qcarcam_color_fmt_t color_fmt;`
		
		`/**< The buffers to use*/`
		`qcarcam_buffer_t buffers[QCARCAM_MAX_NUM_BUFFERS];`
		
		`/**< The number of buffers*/`
		`unsigned int n_buffers;`
		`}cam_vm_cmd_s_buffer_t;`

`CAM_MSG_START`	–	`typedef struct`
		`{`
		`/**< handle returned during open*/`
		`qcarcam_hndl_t hndl;`
		`}cam_vm_cmd_start_t;`

`CAM_MSG_STOP`	–	See `CAM_MSG_START`
`CAM_MSG_PAUSE`	–	See `CAM_MSG_START`
`CAM_MSG_RESUME`	–	See `CAM_MSG_START`

`CAM_MSG_GET_FRAME`	Blocks until a frame is ready in a buffer	`typedef struct`
		`{`
		`/**< handle returned during open*/`
		`qcarcam_hndl_t hndl;`
		
		`/**< timeout*/`
		`unsigned long long int timeout;`
		
		`/**< flags*/`
		`unsigned int flags;`
		`}cam_vm_cmd_get_frame_t;`

hypcamera_msg_id_type	Description	Details

CAM_MSG_RELEASE_FRAME	Releases a buffer back to queue so that hardware can write to it again	typedef struct
		{
		    /**< handle returned during open*/
		    qcarcam_hndl_t hndl;
		
		    /**< buffer index to release back to ais*/
		    int idx;
		}cam_vm_cmd_release_frame_t;

CAM_MSG_CLOSE	Specifies the close device call. The data stores the corresponding handle.	typedef struct { /**< The opened service handle. */ qcarcam_hndl_t handle; }cam_vm_cmd_close_t ;

3.3.3 Hypervisor response message structure

Table 3-2 Hypervisor response message structure

hypcamera_msg_id_type	Description	Details
CAM_MSG_OPEN	Returns the status and a handle to the newly opened input	typedef struct { /**< The status of open. */ unsigned int status; /**< The opened service handle. */ qcarcam_hndl_t handle; }cam_vm_cmd_open_rsp_t

CAM_MSG_QUERY_INPUTS	Queries what inputs are available.
	For more details, see `qcarcam_input_t`.
		typedef struct
		{
		    /**< Available input structure */
		    qcarcam_input_t p_inputs[MAX_NUM_AIS_INPUTS];
		
		    /**< Filled size */
		    unsigned int ret_size;
		
		    /**< status */
		    qcarcam_ret_t status;
		}cam_vm_cmd_query_inputs_rsp_t;

CAM_MSG_GET_PARM	Gets a parameter	–

hypcamera_msg_id_type	Description	Details
`CAM_MSG_SET_PARM`	Sets a parameter	–
`CAM_MSG_SET_BUFFERS`	–	`typedef struct { /**< The status of call. */ unsigned int status; }cam_vm_cmd_s_buffer_rsp_t;`
`CAM_MSG_START`	–	`typedef struct { /**< The status of start. */ unsigned int status; }cam_vm_cmd_start_rsp_t;`
`CAM_MSG_STOP`	–	See `CAM_MSG_START`
`CAM_MSG_PAUSE`	–	See `CAM_MSG_START`
`CAM_MSG_RESUME`	–	See `CAM_MSG_START`
`CAM_MSG_GET_FRAME`	Blocks until a frame is ready in a buffer	`typedef struct { /**< The status of get frame. */ unsigned int status; /**< frame info*/ qcarcam_frame_info_t frame_info; }cam_vm_cmd_get_frame_rsp_t;`
`CAM_MSG_RELEASE_FRAME`	Releases a buffer back to queue so that hardware can write to it again	`typedef struct { /**< The status of release frame. */ unsigned int status; }cam_vm_cmd_release_frame_rsp_t;`
`CAM_MSG_CLOSE`	Specifies the close device call. The data stores the corresponding handle.	`typedef struct { /**< The status of close. */ unsigned int status; }cam_vm_cmd_close_rsp_t;`

3.4 Data flow

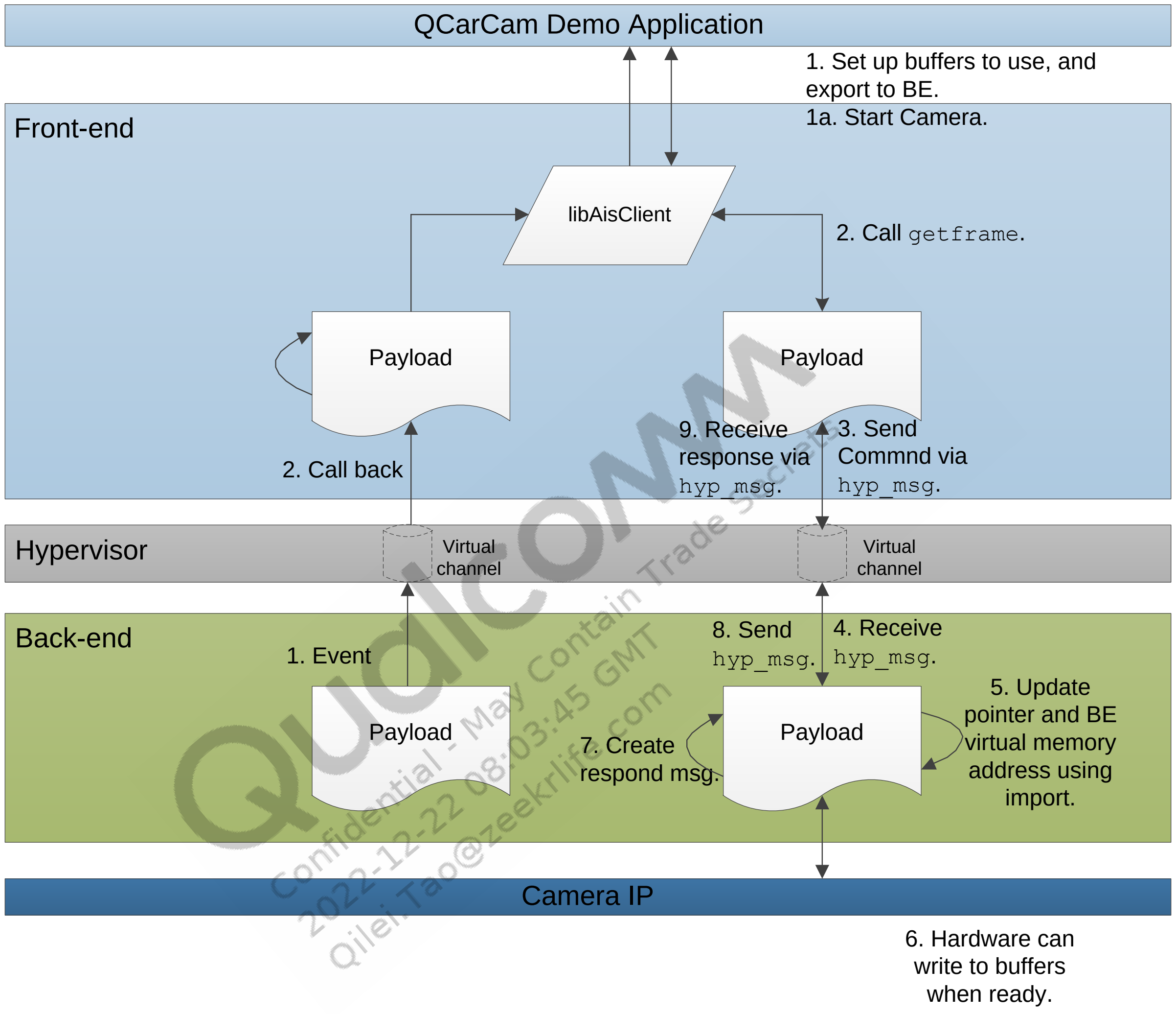


Figure 3-3 Data flow of a Hypervisor I/O Control call

Camera data flow is transparent to the message channel. Memory is mapped during `set_buffer()`. The `Getframe()` call returns which buffer contains data, allowing the application to use the data and then call `releaseFrame()` to give it back to the system to use again.

4 AIS client application

4.1 Typical client workflow

Figure 4-1 shows the recommended call flow when using the QCarCam API.

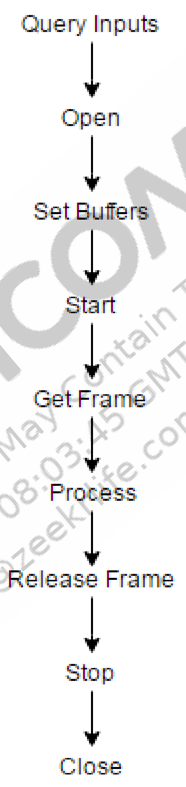


Figure 4-1 QCarCam call flow

4.2 Demo application

The `qcarcam_test` native demo applications are executable over the terminal to test the QCarCam API. This demo is available on GHS, Android O, QNX, and Automotive Grade Linux (AGL) with root permissions.

Table 4-1 Command-line arguments for each OS

Parameter	Description	Example
`config`	Specifies `qcarcam_config.xml` file location	`config=/bin/camera/qcarcam_test/qcarcam_config.xml`
`dumpFrame`	Enables frame dump every N frames	`dumpFame` = `50`
`startStop`	Starts/Stops every N frames	`startStop` = `50`

Parameter	Description	Example
`pauseResume`	Pauses/Resumes every N frames	`pauseResume` = `50`
`noDisplay`	Runs without displaying frames on the display	`noDisplay`
`singlethread`	Runs `qcarcam_test` on a single thread	`singlethread`
`printfps`	Prints average frames per second every N seconds	`printfps` = `10`

4.2.1 Android O

To run the demo application on Android O:

```
adb root
adb remount
adb shell
ais_server &
//one camera test for camera at channel 0
qcarcam_test -config=/system/bin/qcarcam_config_single.xml
//multi camera test can also be used for one camera
qcarcam_test -config=/system/bin/qcarcam_config.xml
```

4.2.2 AGL

To run the demo application on AGL:

```
ais_server &
// one camera test for camera at channel 0
qcarcam_test -config=/data/misc/camera/qcarcam_config_one.xml
```

4.2.3 QNX

To run the demo application on QNX:

```
ais_server &
cd bin/camera/qcarcam_test
// one camera test for camera at channel 0
./qcarcam_test -config=qcarcam_config_single.xml
//multi camera test can also be used for one camera
./qcarcam_test -config=/qcarcam_config.xml
```

4.2.4 QNX Hypervisor

To run the demo application on QNX, Hypervisor, and AGL (with AGL running as GVM for example):

```
//Make  sure to enable the camera demon on QNX side:

//push qcarcam_config_single_0.xml to AGL side /usr/bin

//Run test in AGL GVM:
qcarcam_test -config=/usr/bin/qcarcam_config_single_0.xml
//Result: Preview camera CH0 success, and the camera CH0 show on panel
```

4.2.5 GHS RTOS

To launch the demo application on GHS RTOS:

```
 rt qcarcam_test Initial
```

This command only launches the camera. Future versions of this service will include dynamic virtual address space (VAS).

5 Customization

5.1 Requirements

Most customization requirements change the camera type and bridge chips.

Figure 2-2 showed a submodule (sensor drivers) containing the following:

- TI960_lib
  - OV10635
  - OV10640
- BA_lib
  - LA (for both Android and AGL)
  - QNX,GHS

TI960_lib and BA_lib are bridge chip drivers. OV10635 and OV10640 are sensor drivers. Users can modify these source files to port to a new type of bridge chip and camera module.

NOTE: The file `camera_config.c` contains structures that define camera information, including ID and connection relationship.

5.2 File locations

Camera frameworks from user space is located in `camera/services/mm-camera/ais/`.

On the GHS platform, camera files are located at `ghs_apps_proc/qc_bsp/AMSS/multimedia/camera`.

Table 5-1 lists file locations (using AGL as an example) for each block in the user space.

Table 5-1 File locations

Block	Location
Sensor driver	`vendor/qcom/proprietary/ais/ImagingInputs/SensorLibs/`
`Camera_config.so`	`vendor/qcom/proprietary/ais/ais/CameraConfig/`
`AIS_client` and `AIS_server`	`vendor/qcom/proprietary/ais/CameraMulticlient/`
Camera platform lib	`vendor/qcom/proprietary/ais/CameraPlatform/`
Camera imaging engine, managers, and configurers	`vendor/qcom/proprietary/ais/Engine/`
Hardware driver API	`vendor/qcom/proprietary/ais/HWDrivers/`

Block	Location
Application tests	`vendor/qcom/proprietary/ais/test/`
Kernel level blocks	▪ `kernel/MSM-4.4/drivers/media/platform/MSM/ais/` ▪ `kernel/MSM-3.18/drivers/media/platform/MSM/ais/`

A References

A.1 Related documents

Title	Number
Qualcomm Technologies, Inc.	
S820A QNX Hypervisor Camera Architecture Overview	80-CF832-1
SA6155/SA8155 Automotive Camera AIS Customization Guide	80-PG469-93
SA8155/SA8155P HQX Automotive Camera Architecture Overview	80-PG469-195
SA6155/SA6155P HQX Automotive Camera Architecture Overview	80-PK753-195
QCarCam API Overview	80-P2310-25

A.2 Acronyms and terms

Acronym or term	Definition
AIS	Automotive imaging system
CSI	Camera Serial interface
CVBS	Composite Video Broadcast Signal
FPD-Link	Flat Panel Display-Link
GHS	Green Hills
GVM	Guest virtual machine
HAB	Hypervisor Abstract Communication driver
HDMI	High-Definition Multimedia Interface
MIPI	Mobile Industry Processor Interface
PVM	Physical virtual machine
RTOS	Real Time Operating System
VAS	Virtual address space