Android System Development Day-3

Team Emertxe

Table of Content

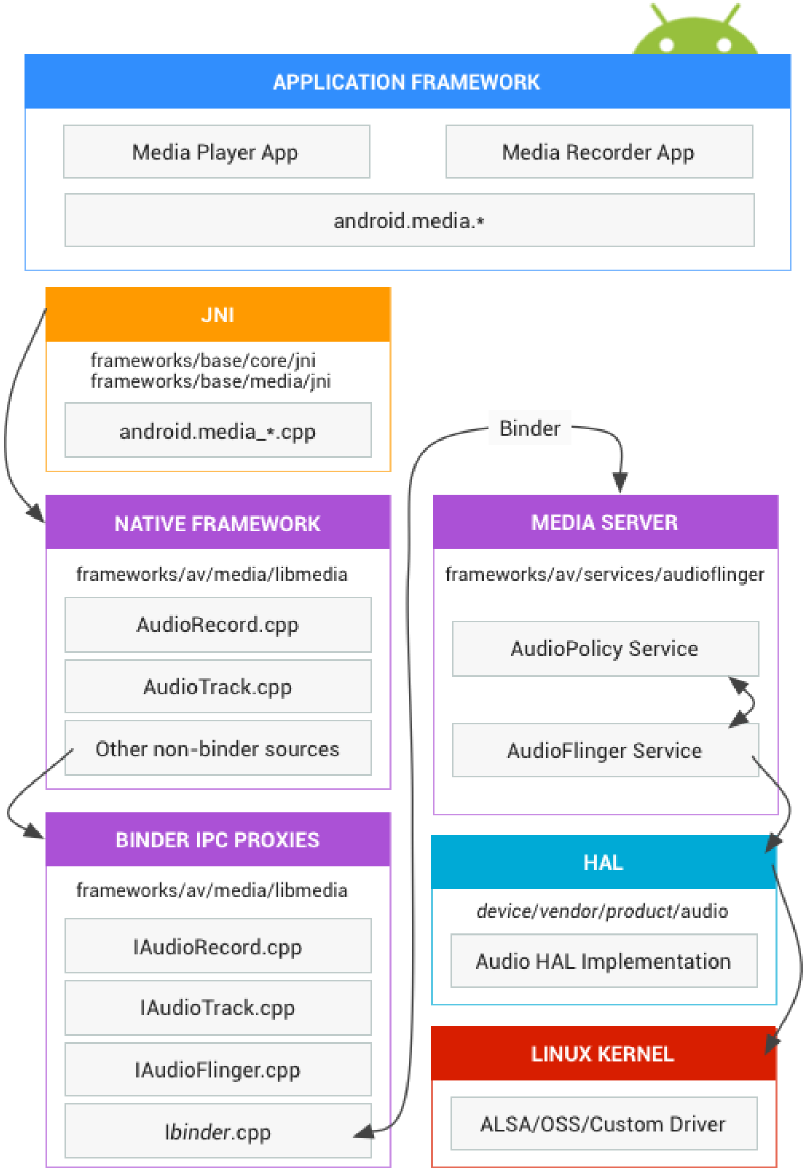
- Android Audio HAL
  - Audio Architecture
  - Audio HAL interface
  - Audio Policy
  - Audio HAL compilation & verification
  - Overview of Tinyalsa
- Android Video HAL
  - Camera Architecture
  - Overview of camera HAL interface
  - Overview of V4L2
  - Enabling V4l2 in kernel
  - Camera HAL compilation and verification

Audio HAL

Audio HAL
(Overview)

- Connects higher-level, audio-specific framework APIs in android.media to the underlying audio driver and hardware
- Defines standard interface that audio services call into and that you must implement for your audio hardware to function correctly
- Interfaces are located in hardware/libhardware/include/hardware

Audio Architecture


APPLICATION FRAMEWORK

Media Player App

Media Recorder App

android.media.*

JNI

frameworks/base/core/jni
frameworks/base/media/jni

android.media_*.cpp

NATIVE FRAMEWORK

frameworks/av/media/libmedia

AudioRecord.cpp

AudioTrack.cpp

Other non-binder sources

BINDER IPC PROXIES

frameworks/av/media/libmedia

IAudioRecord.cpp

IAudioTrack.cpp

IAudioFlinger.cpp

Ibinder.cpp

Binder

MEDIA SERVER

frameworks/av/services/audioflinger

AudioPolicy Service

AudioFlinger Service

HAL

device/vendor/product/audio

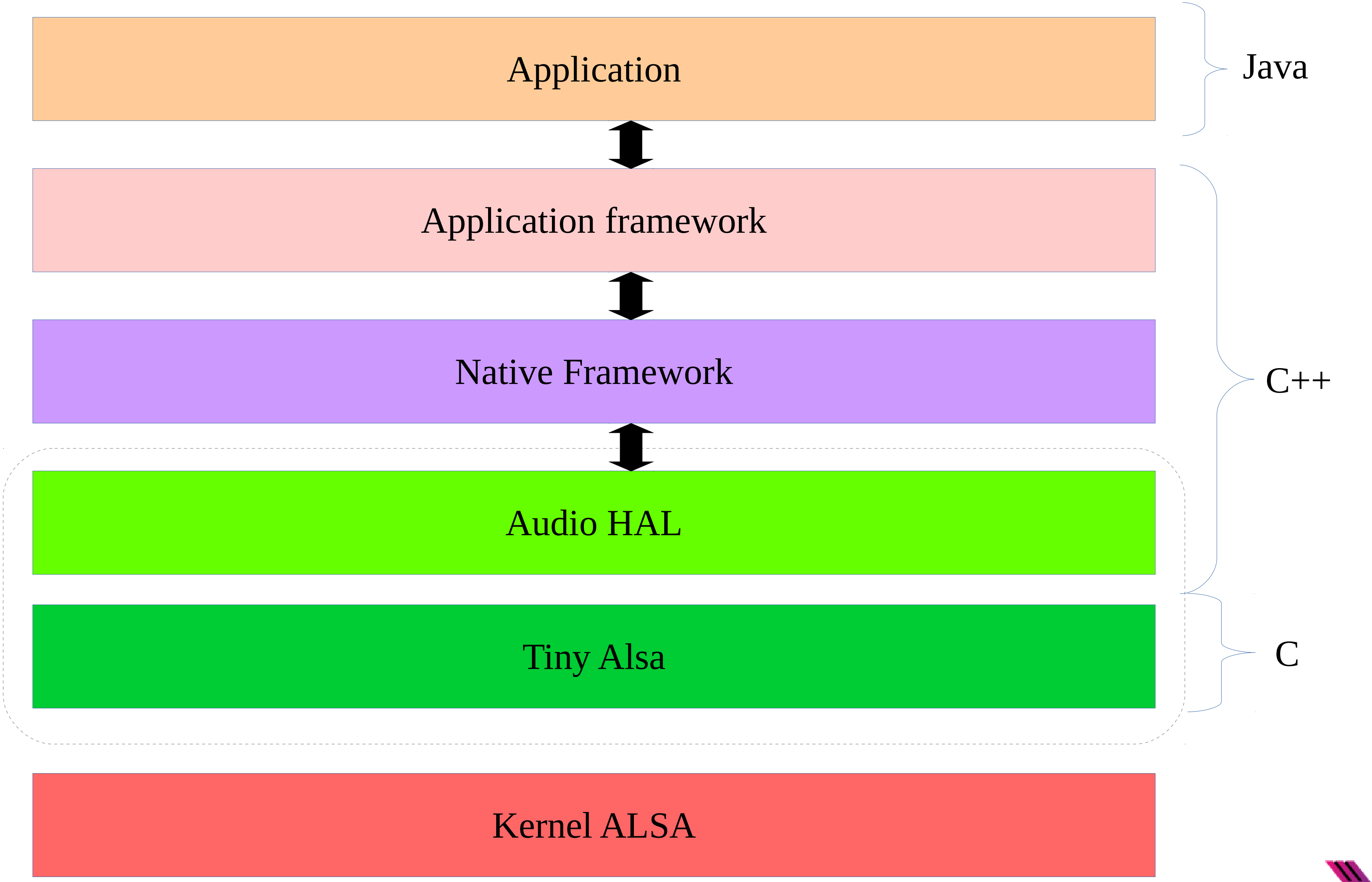
Audio HAL Implementation

LINUX KERNEL

ALSA/OSS/Custom Driver

*source : Android documentation

Audio Architecture



Application

Application framework

Native Framework

Audio HAL

Tiny Alsa

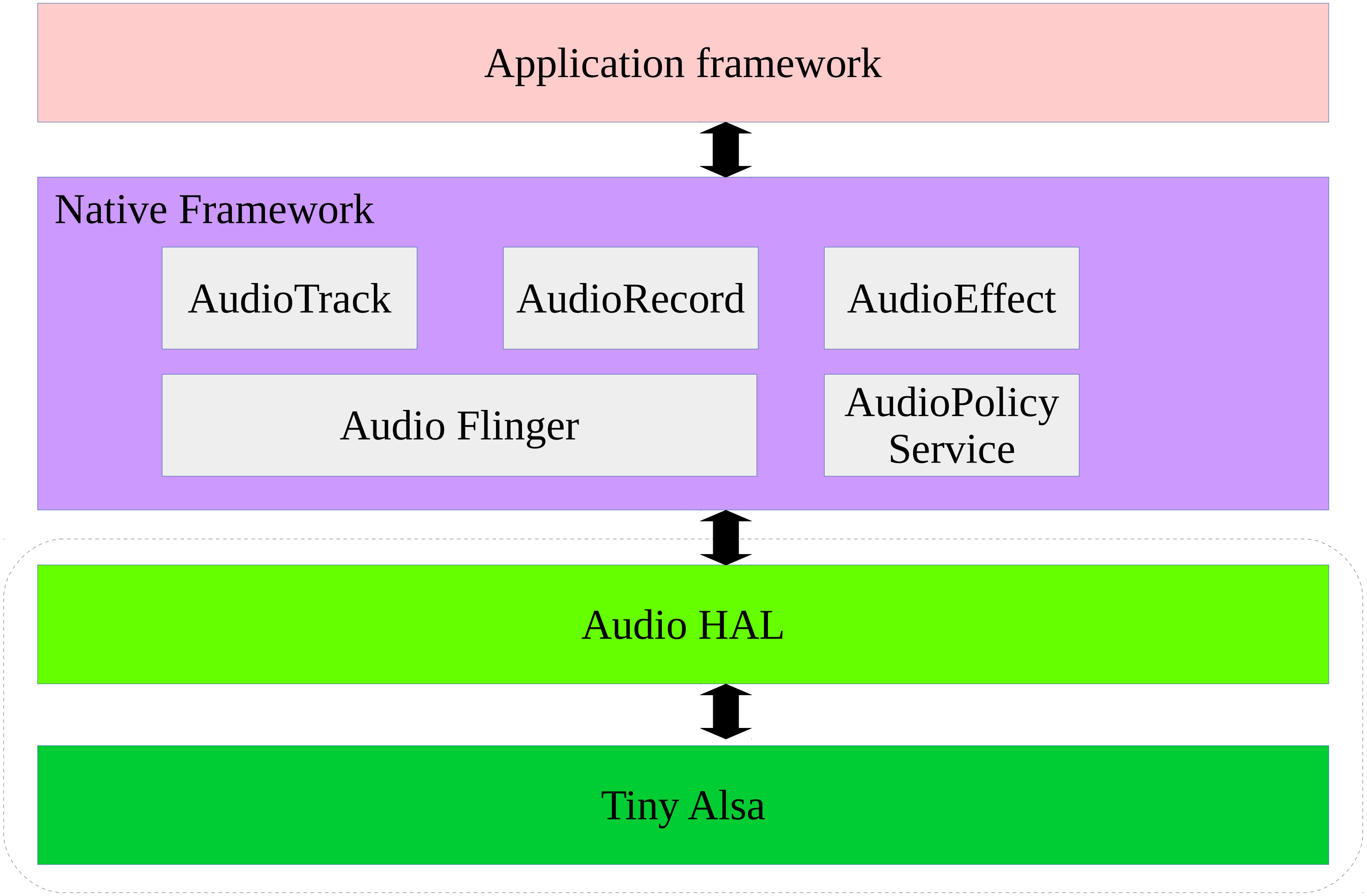
Kernel ALSA

Java

C++

C

Audio Architecture
(Native Framework)



Audio HAL
(Implementation steps)

- Implement interfaces described in audio.h, audio_effect.h files
- Create an audio policy configuration file
  - Describe audio topology and package the HAL implementation into a shared library
- Configure pre-processing effects such as automatic gain control and noise suppression

Audio HAL
(Interfaces)

- Main functions -
  - hardware/libhardware/include/hardware/audio.h
  - audio_stream_t
  - stream_callback_t
  - audio_stream_out_t
  - audio_stream_in_t
  - audio_hw_device_t
- Effects (downmixing, echo, or noise suppression)
  - hardware/libhardware/include/hardware/audio_effect.h
  - audio_effect_library_t
  - struct effect_interface_s

Audio HAL
(Interfaces)

- Reference implementation -
  - system/media/audio/include/system/audio.h
  - device/samsung/tuna/audio

Audio HAL

(Audio policy configuration)

- Naugat introduced new XML based audio policy
  - New audio policy configuration file audio_policy_configuration.xml
  - Location : /system/etc
  - Include build option USE_XML_AUDIO_POLICY_CONF := 1 in device makefile
- Old policy (audio_policy.conf) is deprecated
  - Location : device/<company>/<device>/audio/audio_policy.conf
  - Example : device/samsung/tuna/audio/audio_policy.conf

Audio HAL

(Audio policy – non-XML sample)

```
global_configuration {
    attached_output_devices AUDIO_DEVICE_OUT_SPEAKER
    default_output_device AUDIO_DEVICE_OUT_SPEAKER
}

audio_hw_modules {
    primary {
        outputs {
            primary {
                sampling_rates 44100
                channel_masks AUDIO_CHANNEL_OUT_STEREO
                formats AUDIO_FORMAT_PCM_16_BIT
                devices AUDIO_DEVICE_OUT_SPEAKER
                flags AUDIO_OUTPUT_FLAG_PRIMARY
            }
        }
    }
 }
```

*frameworks/av/services/audiopolicy/audio_policy.conf

Audio HAL
(Audio policy configuration)

- XML files from other folders can be included using Xinclude element
  - Example : <xi:include href=”a2dp_audio_policy_configuration.xml”/>
- Audio policy files
  - A2DP: a2dp_audio_policy_configuration.xml
  - Reroute submix: rsubmix_audio_policy_configuration.xml
  - USB: usb_audio_policy_configuration.xml

Audio HAL

(Audio policy – XML sample)

```
<audioPolicyConfiguration version="1.0">
    <globalConfiguration speaker_drc_enabled="true"/>
<module name="primary" halVersion="3.0">
    <attachedDevices>
        <item>Speaker</item>
        <item>Built-In Mic</item>
    </attachedDevices>
    <defaultOutputDevice>Speaker</defaultOutputDevice>
</module>
    <xi:include href="audio_policy_volumes.xml"/>
    <xi:include href="default_volume_tables.xml"/>
</audioPolicyConfiguration>
```

*frameworks/av/services/audiopolicy/config/audio_policy_configuration.xml

Audio HAL

(Advantages of XML based policy)

- Audio profiles are now structured similar to HDMI Simple Audio Descriptors and enable a different set of sampling rates/channel masks for each audio format
- Explicit definitions of all possible connections between devices and streams
  - Previously, an implicit rule made it possible to interconnect all devices attached to the same HAL module, preventing audio policy from controlling connections requested with audio patch APIs
  - In XML format, topology description now defines connection limitations
- Support for includes avoids repeating standard A2DP, USB, or reroute submit definitions
- Customizable volume curves
  - Previously, volume tables were hard-coded
  - In XML format, volume tables are described and can be customized

Audio HAL
(shared library)

- Create a device/<company>/<device>/audio directory to contain your library's source files.
- Create an Android.mk file to build the shared library
- Make file shall contain
  - LOCAL_MODULE := audio.primary.<device>
- Create device.mk (see rpi3.mk in device/brcm/rpi/ directory)

Audio HAL
(Preprocessing Effects)

- Default pre-processing effects applied for each AudioSource are specified in the /system/etc/audio_effects.conf file
- To apply custom effects for every AudioSource, create a /system/vendor/etc/audio_effects.conf file and specify the pre-processing effects to turn on

Audio HAL
(Module)

- Audio module shall be initialized as mentioned below
- Name shall be HAL_MODULE_INFO_SYM
- The hal_module_methods points to open function

```
struct audio_module HAL_MODULE_INFO_SYM = {
    .common = {
        .tag = HARDWARE_MODULE_TAG,
        .module_api_version = AUDIO_MODULE_API_VERSION_0_1,
        .hal_api_version = HARDWARE_HAL_API_VERSION,
        .id = AUDIO_HARDWARE_MODULE_ID,
        .name = "Raspberry Pi Audio HW HAL",
        .author = "The Android Open Source Project",
        .methods = &hal_module_methods,
    },
};
```

Audio HAL
(hw_module_methods_t)

```
static struct hw_module_methods_t hal_module_methods = {
    .open = adev_open,
};
```

TinyALSA

Audio HAL
(tinyalsa)

- ALSA is an open source sound architecture for Linux
- TinyALSA is a small ALSA library aimed to be used in Android
- Source code is available in “external/tinyalsa” directory
- Utility executables
  - tinyplay
  - tinymix
  - tinycap
  - Tinypcminfo
- Shared library - libtinyalsa

TinyALSA vs ALSA

#	Feature	TinyALSA	ALSA
1	*Control Interface*	Partly support	Full Support
2	*Mixer Interface*	Partly support	Full Support
3	*PCM Interface*	Partly support	Full Support
4	*Raw MIDI Interface*	No	Full Support
5	*Sequencer Interface*	No	Full Support
6	*Timer Interface*	No	Full Support

Audio HAL
(tinyalsa PCM APIs)

#	API	Description
1	pcm_open ()	Open PCM device
2	pcm_read ()	Read PCM audio data from PCM
3	pcm_write ()	Write data. This is start playback at first write.
4	pcm_close ()	Close driver
5	pcm_set_config ()	Set configuration parameters
6	pcm_get_config ()	Get configuration parameters
7	pcm_is_ready ()	Check if PCM driver is ready

*Path: external/tinyalsa/include/tinyalsa/asoundlib.h

Audio HAL
(tinyalsa Parameter APIs)

#	API	Description
1	pcm_params_get ()	Get parameters
2	pcm_params_get_mask ()	Get mask
3	pcm_params_get_min ()	Get min
4	pcm_params_set_min ()	Set Min
5	pcm_params_get_max ()	Get Max
6	pcm_params_set_max ()	Set Max
7	pcm_params_free ()	Free memory allocated for parameters
8	pcm_params_to_string ()	Human readable parameter string
9	pcm_params_format_test ()	

*Path: external/tinyalsa/include/tinyalsa/asoundlib.h

Audio HAL

(tinyalsa MMAP APIs)

#	API	Description
1	pcm_mmap_write ()	Write to shared memory
2	pcm_mmap_read ()	Read from shared memory
3	pcm_mmap_begin ()	Request to access a portion of direct (mmap) area
4	pcm_mmap_commit ()	Completed the access to area requested (using begin)
5	pcm_mmap_avail ()	Number of frames ready to be read (capture) / written (playback)
8	pcm_prepare ()	Prepare the PCM substream to be triggerable
9	pcm_start ()	Start a PCM channel that doesn't transfer data
10	pcm_stop ()	Stop a PCM channel that doesn't transfer data
11	pcm_ioctl ()	ioctl function for PCM driver
12	pcm_wait ()	Interrupt driven API
13	pcm_get_poll_fd ()	Get poll file descriptor

*Path: external/tinyalsa/include/tinyalsa/asoundlib.h

Audio HAL
(tinyalsa Other APIs)

#	API	Description
1	pcm_get_error ()	Returns human readable reason (string) for the last error
2	pcm_format_to_bits ()	Returns the sample size in bits for a PCM format
3	pcm_get_buffer_size ()	buffer size that should be used for pcm_write
4	pcm_frames_to_bytes ()	buffer size that should be used for pcm_write
5	pcm_bytes_to_frames ()	buffer size that should be used for pcm_write
6	pcm_get_latency ()	Get the PCM latency in milliseconds
7	pcm_get_htimestamp ()	Returns available frames in pcm buffer to read (in stream) or write (out stream)and corresponding time stamp

*Path: external/tinyalsa/include/tinyalsa/asoundlib.h

Audio HAL

(Hands-on : configure audio)

- $ tinymix [Press Enter Key]

```
Mixer name: 'bcm2835 ALSA'
Number of controls: 6
ctl type         num      name                          value
0   INT 1        PCM      Playback Volume               0
1   BOOL 1       PCM      Playback Switch               On
2   INT 1        PCM      Playback Route                0
3   IEC958  1 IEC958   Playback Default              unknown
4   IEC958  1 IEC958   Playback Con Mask             unknown
5   IEC958  1 IEC958   Playback PCM Stream           unknown
```

Audio HAL
(Hands-on : configure audio)

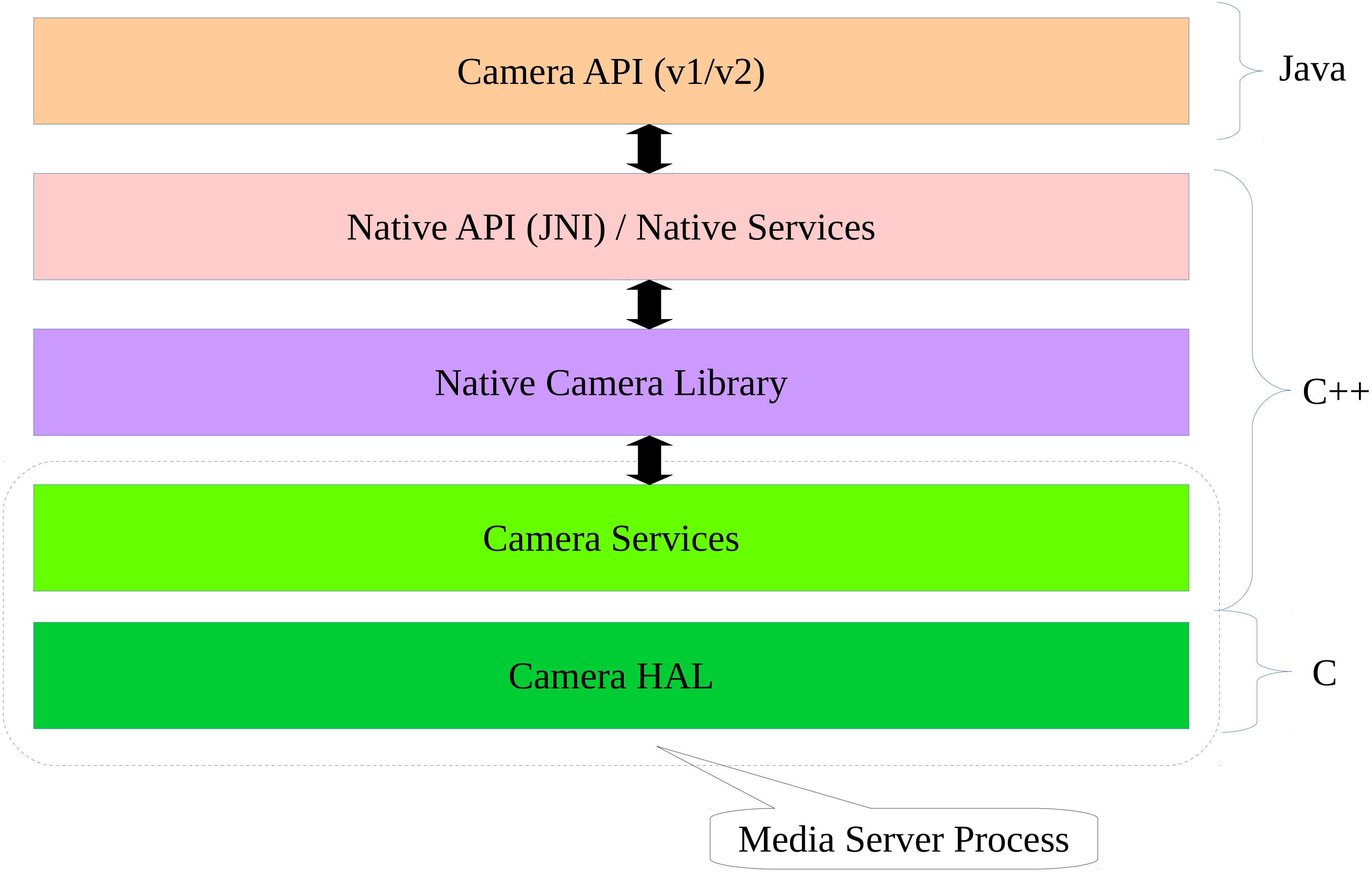
- Set volume
  - $ tinymix 0 50
- Set route
  - $ tinymix 2 1
- $ adb push <media-file-name> /data/
- Play media file
  - $ tinyplay <media-file-name>.wav

Camera HAL

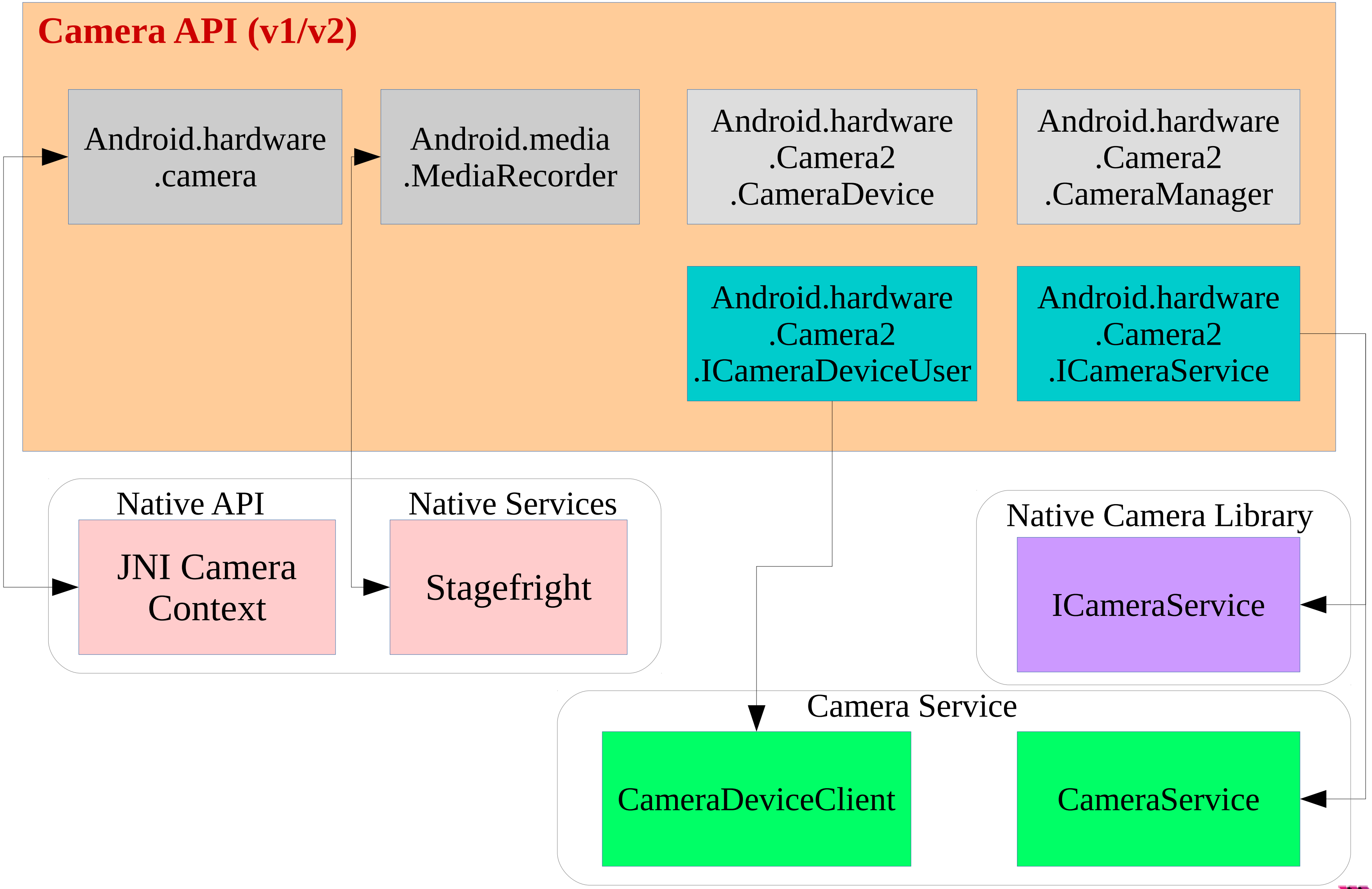
Camera HAL
(Overview)

- Connects the higher level camera framework APIs in android.hardware to underlying camera driver and hardware
- The camera HAL provides interfaces for use in implementing camera pipeline components
- Refer to following files
  - camera.h source file
  - camera3.h source file
  - camera_common.h source file

Camera Architecture



Camera Architecture
(APIs)



Camera API (v1/v2)

Android.hardware .camera

Android.media .MediaRecorder

Android.hardware .Camera2 .CameraDevice

Android.hardware .Camera2 .CameraManager

Android.hardware .Camera2 .ICameraDeviceUser

Android.hardware .Camera2 .ICameraService

Native API

JNI Camera Context

Native Services

Stagefright

Native Camera Library

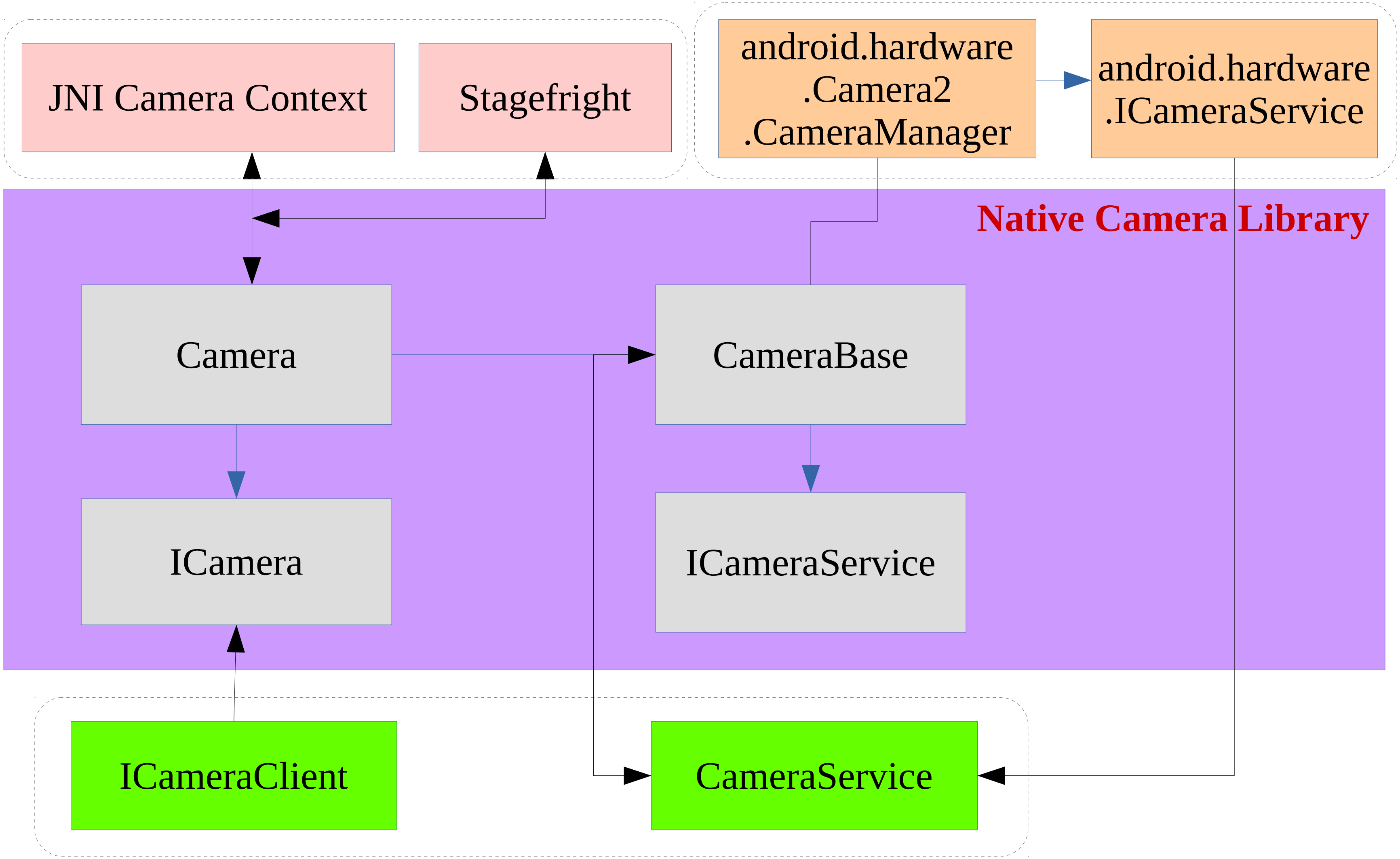
ICameraService

Camera Service

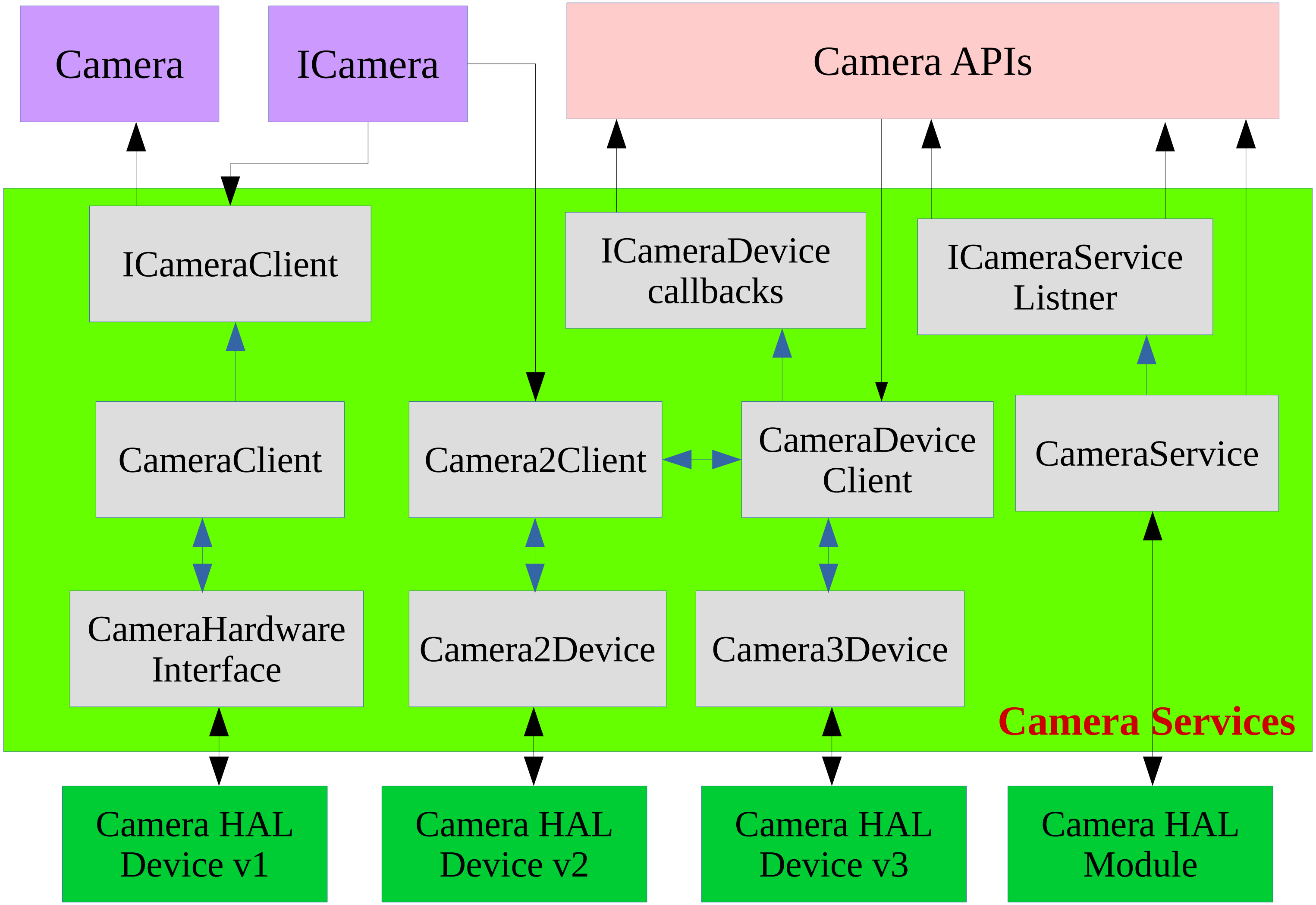
CameraDeviceClient

CameraService

Camera Architecture
(Native Camera Library)

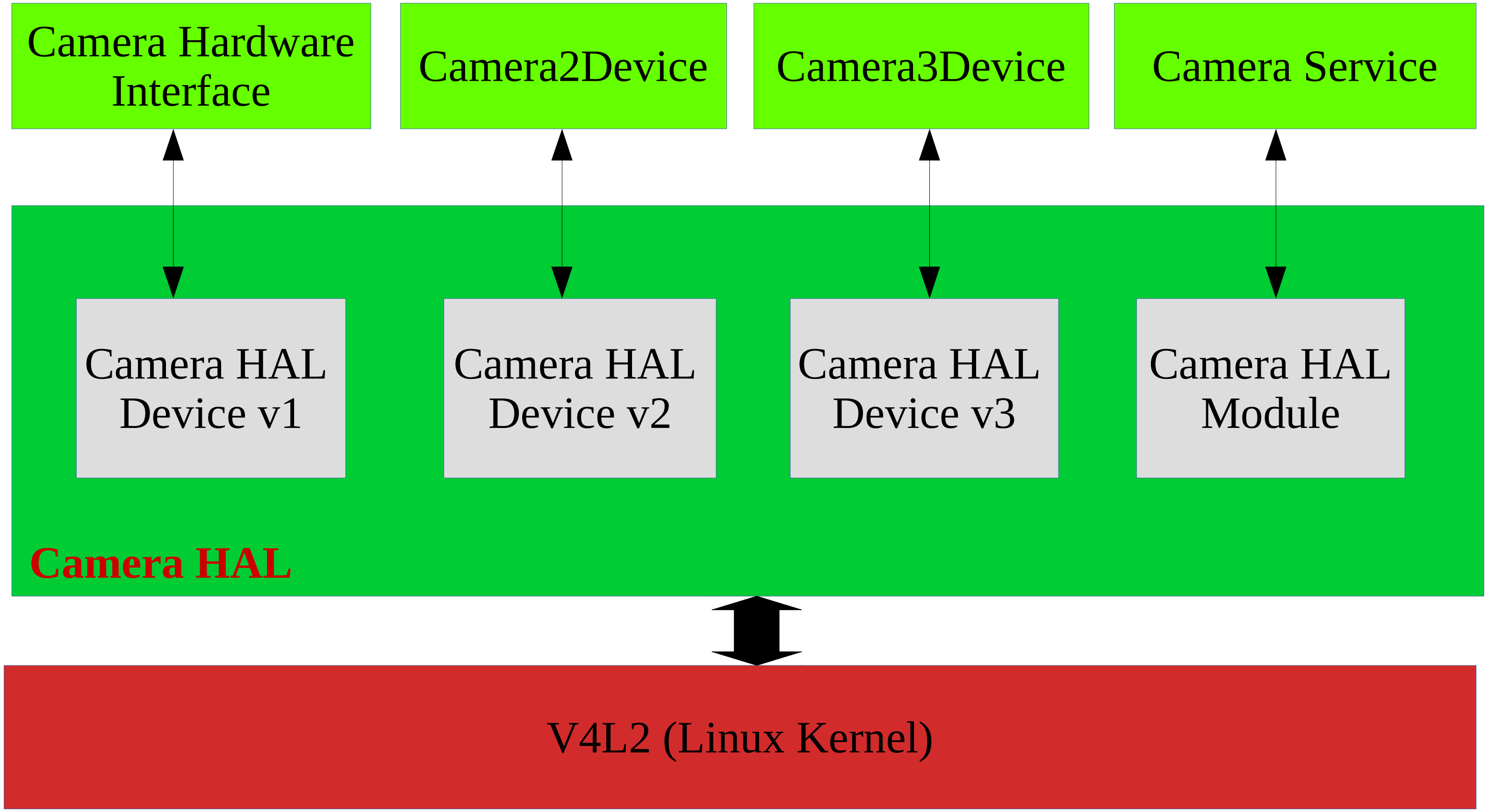


Camera Architecture
(Camera Service)



Camera Architecture

(Camera Services)



Camera HAL
(Implementation)

- HAL sits between camera driver and Android framework
- HAL interface is defined in :
  - hardware/libhardware/include/hardware/camera.h
  - hardware/libhardware/include/hardware/camera_common.h
- Camera_common.h defines camera_module, a standard structure to obtain general information about the camera
- Declares a camera_device struct that in turn contains a camera_device_ops struct with pointers to functions that implement the HAL interface
- Camera parameters are defined in frameworks/av/include/camera/CameraParameters.h

Camera HAL
(Creating shared library)

- Create a device/<company_name>/<device_name>/camera directory to contain library's source files
- Create an Android.mk file to build the shared library
- Make file shall contain
  - LOCAL_MODULE := camera.<device_name>
  - LOCAL_MODULE_RELATIVE_PATH := hw
- Specify camera features by copying the necessary feature XML files in frameworks/native/data/etc directory with device.mk
- See reference file "device/samsung/tuna/device.mk"

Camera HAL
(Creating shared library)

- Declare camera’s media codec, format, and resolution capabilities in
  - device/<company_name>/<device_name>/media_profiles.xml
  - device/<company_name>/<device_name>/media_codecs.xml XML files
- Add media_profiles.xml and media_codecs.xml in device.mk
- Add Camera app in PRODUCT_PACKAGES variable in device.mk to be part of system image

Camera HAL
(v1, V2, V3)

- Supported camera HAL1 as many devices still rely on it
- Android camera service supports implementing both HALs (1 & 3)
- Useful when to support a less-capable front-facing camera with camera HAL1
- Camera HAL2 is not supported as it was a temporary step on the way to camera HAL3
- Single camera HAL module which lists multiple independent camera devices (each may have own version number)
- Camera module 2 or newer required to support devices 2 or newer

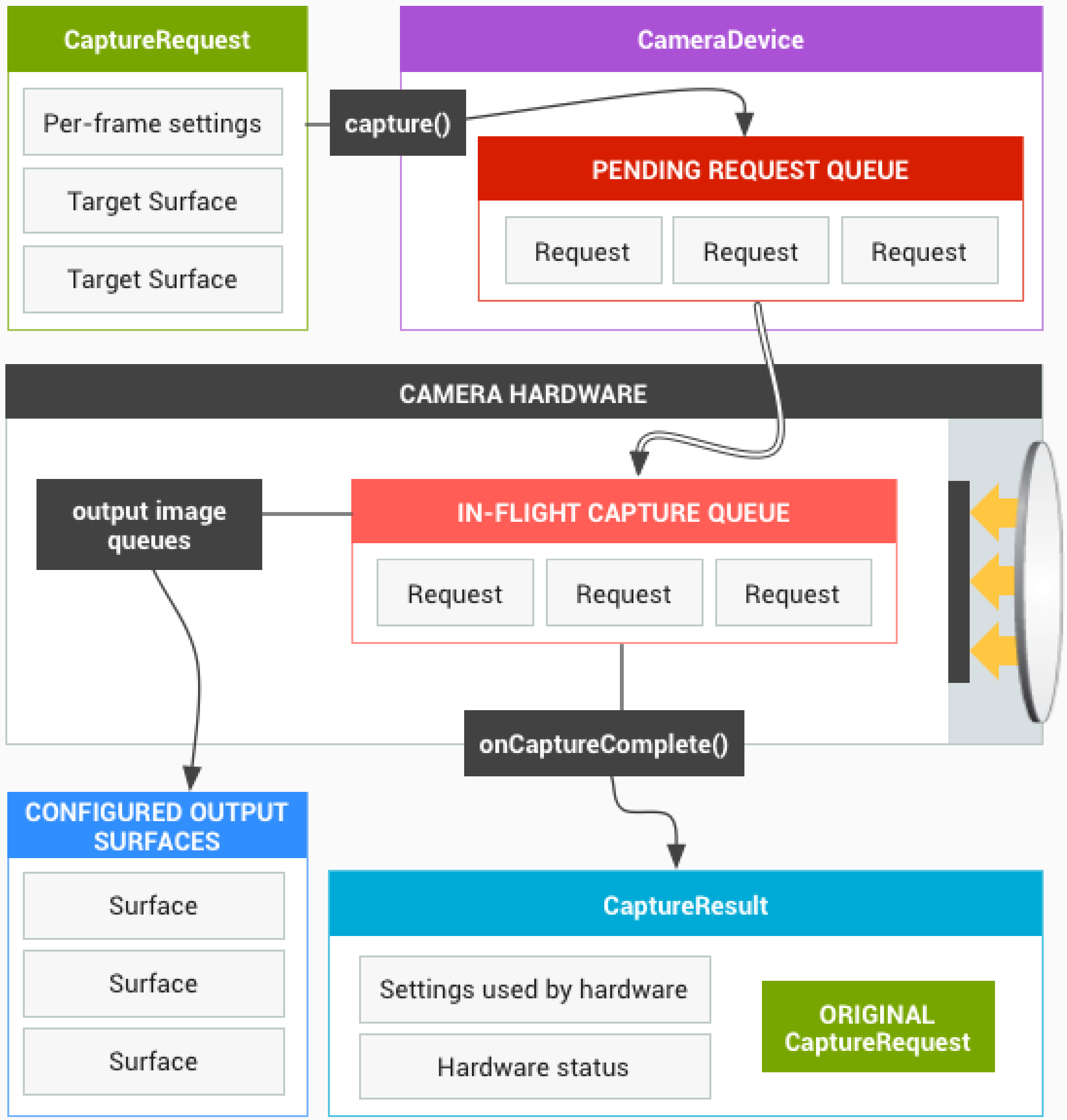
Camera3

- Re-designed to
  - Increase the ability of app to control camera
  - Be more efficient and maintainable
- Additional camera control enables
  - Develop high-quality camera app
  - Operate reliably across multiple products
  - Use device-specific algorithms whenever possible & maximize quality & performance
- Structures the operation modes into a single unified view

Camera3

- Single unified view results in better user control for
  - Focus and exposure
  - Post-processing, such as noise reduction, contrast and sharpening
- This simplified view makes it easier for app developers to use the camera's various functions

Camera3



CaptureRequest

Per-frame settings

Target Surface

Target Surface

capture()

CameraDevice

PENDING REQUEST QUEUE

Request

Request

Request

CAMERA HARDWARE

output image queues

IN-FLIGHT CAPTURE QUEUE

Request

Request

Request

onCaptureComplete()

CONFIGURED OUTPUT SURFACES

Surface

Surface

Surface

CaptureResult

Settings used by hardware

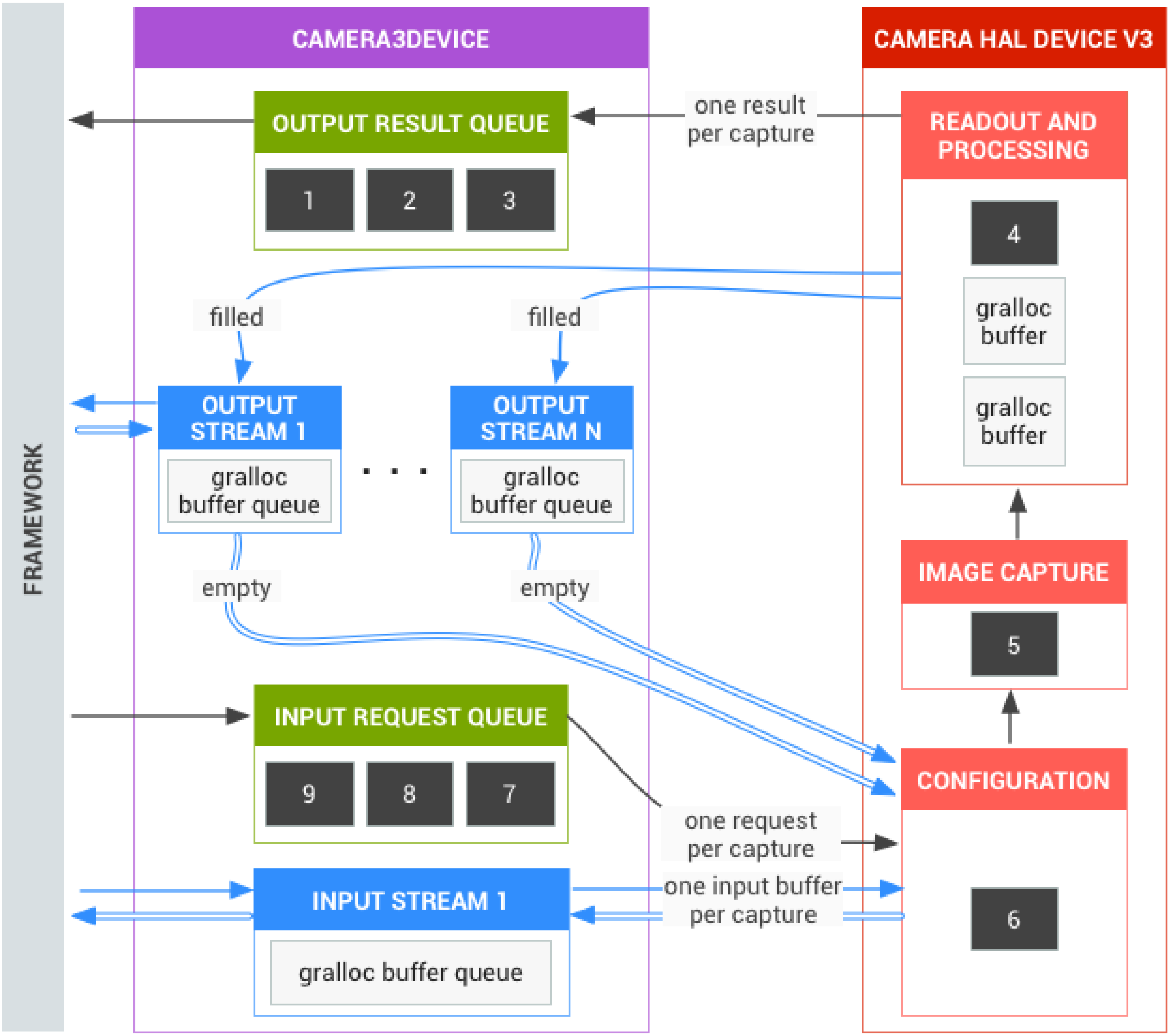
Hardware status

ORIGINAL CaptureRequest

HAL operation

- Asynchronous requests for captures come from the framework
- HAL device must process requests in order
- For each request, produce output result metadata, & one or more output image buffers
- First-in, first-out for requests & results, & for streams referenced by subsequent requests
- Timestamps must be identical for all outputs from a given request, so that the framework can match them together if needed
- All capture configuration and state [except for 3A (auto-exposure, auto-white-balance, auto-focus) control routines] is encapsulated in the requests and results

Camera3 operation



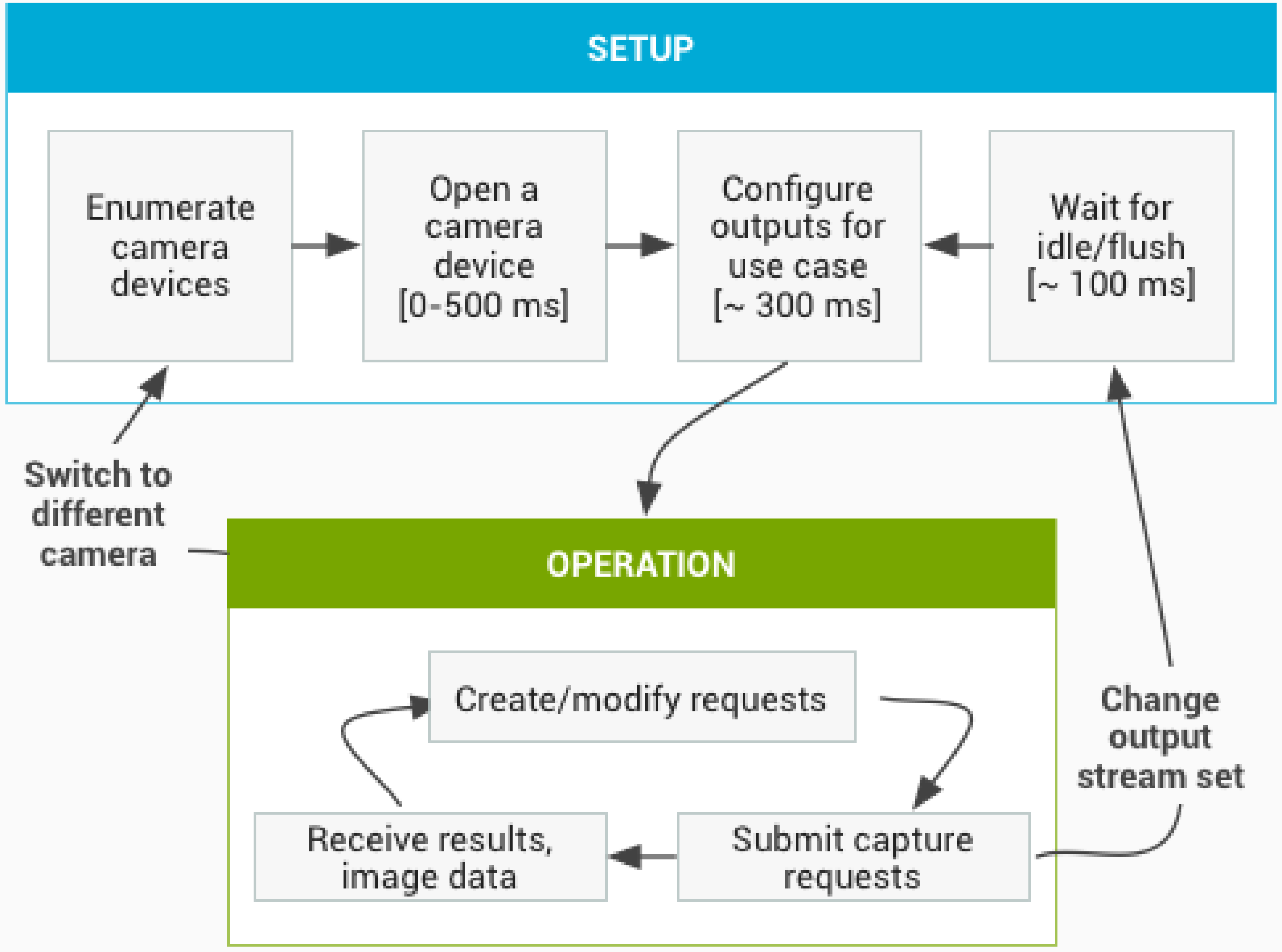
Framework operation

- Framework opens device
  - camera_module_t→common.open()
- Framework checks version field & instantiates appropriate handler for that version of camera hardware device
  - camera3_device_t→ops→initialize()
- Framework configure streams
  - camera3_device_t→ops→configure_streams()
- Framework allocate stream buffers
  - camera3_device_t→ops→register_stream_buffers()

Framework operation

- Framework requests default settings for some number of use cases with calls to camera3_device_t→ops→construct_default_request_settings()
- Framework construct a capture request and send it to HAL
  - camera3_device_t→ops→process_capture_request()
- HAL notifies framework of a started capture request
  - camera3_callback_ops→notify()
- HAL notifies framework of a finished capture request
  - camera3_callback_ops→process_capture_result()
- Framework calls close the device
  - camera3_device_t→common→close()

Camera operation flow


SETUP
Enumerate camera devices
Open a camera device [0-500 ms]
Configure outputs for use case [~ 300 ms]
Wait for idle/flush [~ 100 ms]
Switch to different camera
OPERATION
Create/modify requests
Receive results, image data
Submit capture requests
Change output stream set

Camera Devices

- **LEGACY :** These devices expose capabilities to apps through the Camera API2 interfaces that are approximately the same capabilities as those exposed to apps through the Camera API1 interfaces. The legacy frameworks code conceptually translates Camera API2 calls into Camera API1 calls; legacy devices do not support Camera API2 features such as per-frame controls
- **FULL :** These devices support all of major capabilities of Camera API2 and must use Camera HAL 3.2 or later and Android 5.0 or later
- **LIMITED :** These devices support some Camera API2 capabilities (but not all) and must use Camera HAL 3.2 or later

Camera modes

- The camera 3 HAL device can implement one of two possible operational modes
  - Limited
  - Full
- Full support is expected from new higher-end devices
- Limited mode has hardware requirements roughly in line with those for a camera HAL device v1 implementation, and is expected from older or inexpensive devices
- HAL must indicate its level of support with the android.info.supportedHardwareLevel static metadata entry, with 0 indicating limited mode, and 1 indicating full mode support.

Camera HAL
(versions)

- Camera API1
  - The app-level camera framework on Android 4.4 and earlier devices, exposed through the android.hardware.Camera class
- Camera API2
  - The app-level camera framework on Android 5.0 and later devices, exposed through the android.hardware.camera2 package
- Camera HAL
  - The camera module layer implemented by SoC vendors. The app-level public frameworks are built on top of the camera HAL
- Camera HAL3.1
  - Version of the camera device HAL released with Android 4.4
- Camera HAL3.2
  - Version of the camera device HAL released with Android 5.0

Camera HAL

(Hands-on : enabling v4l2)

- $ make ARCH=arm menuconfig
- Select “Device Drivers”
- Select “Multimedia Support”
- Select “V4L platform devices”
- Select “Broadcom BM2835 MMAL camera interface driver”
- Save configuration and exit

Camera HAL

(Hands-on : enabling v4l2)

- $ ARCH=arm CROSS_COMPILE=arm-linux-androideabi- make zImage -j4
- $ ARCH=arm CROSS_COMPILE=arm-linux-androideabi- make dtbs -j4
- Copy zImage and bcm2710-rpi-3-b.dtb in /boot folder (SD card)
- Copy vc4-kms-v3d.dtbo in /boot/overlay folder (SD card)

Enabling Camera

- Add following parameters in /boot/config.txt
  - start_x=1
  - gpu_mem=256 (128 or more)
- Ensure start_x.elf, fixup_x.dat files are present in /boot folder in SD card

Camera Interface
(camera_device_ops_t)

- set_preview_window
- set_callbacks (notification and data)
- enable_msg_type
- disable_msg_type
- msg_type_enabled
- start_preview
- preview_enabled
- store_meta_data_in_buffers

Camera Interface

(camera_device_ops_t)

- start_recording
- stop_recording
- recording_enabled
- release_recording_frame
- auto_focus
- cancel_auto_focus
- take_picture
- cancel_picture

Camera Interface
(camera_device_ops_t)

- set_parameters
- get_parameters
- put_parameters (to return memory to HAL)
- send_command (to driver)
- release (hardware resources)
- dump (state of camera)

Camera Interface
(preview_stream_ops_t)

- deque_buffer
- enqueue_buffer
- cancel_buffer
- set_buffer_count
- set_buffers_geometry
- set_crop
- set_usage
- set_swap
- get_min_undequed_buffer_count
- lock_buffer
- set_timestamp

Video4Linux (V4L2)

What is V4L2?

- An open source standard to capture real time video for linux systems
- Second version of video for linux
- V4L2 is two layer driver system
- Top layer is videodev module
- Lower layer is collection of several driver modules
- V4L2 drivers are clients of videodev

V4L2

(Devices)

Device Name		Minor Number	Description
From	To		
/dev/video0	/dev/video63	0-63	Video Capturer Devices
/dev/radio0	/dev/radio63	64-127	AM/FM Radio Devices
/dev/vtx0	/dev/vtx31	192-223	Teletext Devices
/dev/vbi0	/dev/vbi15	224-239	VBI Devices

V4L2

(Video Device)

#	Members	Description
char	name[32]	Name of the device
int	type	Type of V4L2 device
int	minor	Device minor number
Pointer	*fops	File operations used
Function pointer	void (*release)(struct video_device *vfd)	Release function used by the driver
Void pointer	priv	The private data

V4L2

(File operations)

#	Funcrion	Description
1	int (*open)()	Called when a file descriptor is opened on the device file (/dev/videoX)
2	int (*close)()	Called on last close (release) of a file descriptor
3	int (*read)()	Called to data (buffer) of size count (the number of bytes of data requested)
4	int (*write)()	Called to write data of size count (the number of bytes of data to write)
5	int (*ioctl)()	Called when the application calls ioctl()
6	int (*mmap)()	Called when the application calls mmap()
7	int (*poll)()	Called when the application calls select()

V4L2

(Property Negotiation)

- First the application asks for the possibles values of some property
- Next chooses one of the possible values
- Next configures that value
- Finally checks right configuration of the value
- Important properties - Video Input, Video Output, Norm (only for input devices), Modulator (only for output devices), Input Channel, Window Size

V4L2

(Pixel format negotiation)

- Pixel format is how every pixel is stored in memory
- Application need to know this format to allow the properly interpretation of that pixel
- There are two "families" of pixel formats RGB and YUV
- Mostly, devices capture natively in YUV formats
- Video is converted to RGB formats for displaying in the viewer

V4l2
(Buffers)

- V4L2_MMAP
  - Memory mapping (allocated by the kernel)
- V4L2_USERPTR
  - User memory (allocated by user app)
- DMABUF, read/write
  - Direct Memory Access (GPU, OpenGL)
  - Can handle fullHD (1080p) at 60FPS

V4L2
(Driver Compilation)

- Step 1 : Copy driver source files in following folder
  - kernels/rpi/deriver/media/platform/<device>
- Step 2 : Edit Makefile to add following lines
  - <driver_name>-obj := (add all file which required to build mofules with “.o” extantion)
  - obj-$(CONFIG_<unique word which easly identify and give information about driver>) +=<driver_name>.o
- Step 3 : Add following line in Kconfig
  - connfig <driver_name which show on menuconfig>
  - tristate “<write the purpose of module>”
  - default n (Must specify; other options y and m)
  - --help--

    <write brief description about module to help user selection while building the kernel module>

V4L2

(Driver Compilation and loading)

- Step 4 : make ARCH=arm menuconfig
  - Select driver with m option; save and exit
- Step 5 : Compile module from /kernel/rpi directory
  - make ARCH=arm CROSSCOMPILE=<absolute path of compiler> -j4 modules
- Step 6 : copy <driver>.ko file in target board
  - system/lib/modules/<kernel version>/kernel/media/v4l2-core
- Step 7 : Run following command
  - insmod <driver_name>.ko (to load driver)
  - rmmod <driver_name> (to remove driver)

Stay connected

About us: Emertxe is India's one of the top IT finishing schools & self learning kits provider. Our primary focus is on Embedded with diversification focus on Java, Oracle and Android areas

Emertxe Information Technologies Pvt Ltd
No. 83, 1st Floor, Farah Towers,
M.G Road, Bangalore
Karnataka - 560001
T: +91 809 555 7 333
T: +91 80 4128 9576
E: training@emertxe.com

https://www.facebook.com/Emertxe

https://twitter.com/EmertxeTweet

https://www.slideshare.net/EmertxeSlides

ΣMERTXE

Thank You