MTK Camera HALv3 Overall Architecture and Flow

HALv3 Overall Architecture and Flow

Documentation Support

Common Platform

Doc No: DS6000-AK37A-DMT-V1.0ZH
Version: V1.0
Release date: 2018-06-12
Classification: Internal

Keywords
Template, Common Part

MediaTek Inc.

Postal address
No. 1, Dusing 1st Rd. , Hsinchu Science Park, Hsinchu City, Taiwan 30078

MTK support office address
No. 1, Dusing 1st Rd. , Hsinchu Science Park, Hsinchu City, Taiwan 30078

Internet
http://www.mediatek.com/

Document Revision History

Revision	Date	Author	Description
V0.1	2018-05-31	David Cheng	Initial Release

Table of Contents

Lists of Figures

1 Introduction

本文件介紹基於 Android P MTK Camera HAL3 的架構設計，下圖為 Android Camera architecture，文件會著重於 binderized HAL layer，描述 MTK 平台如何實做 Android 所定義的 ICameraProvider、ICameraDevice 與 ICameraDeviceSession HIDL interfaces 並實現 Camera HAL3。

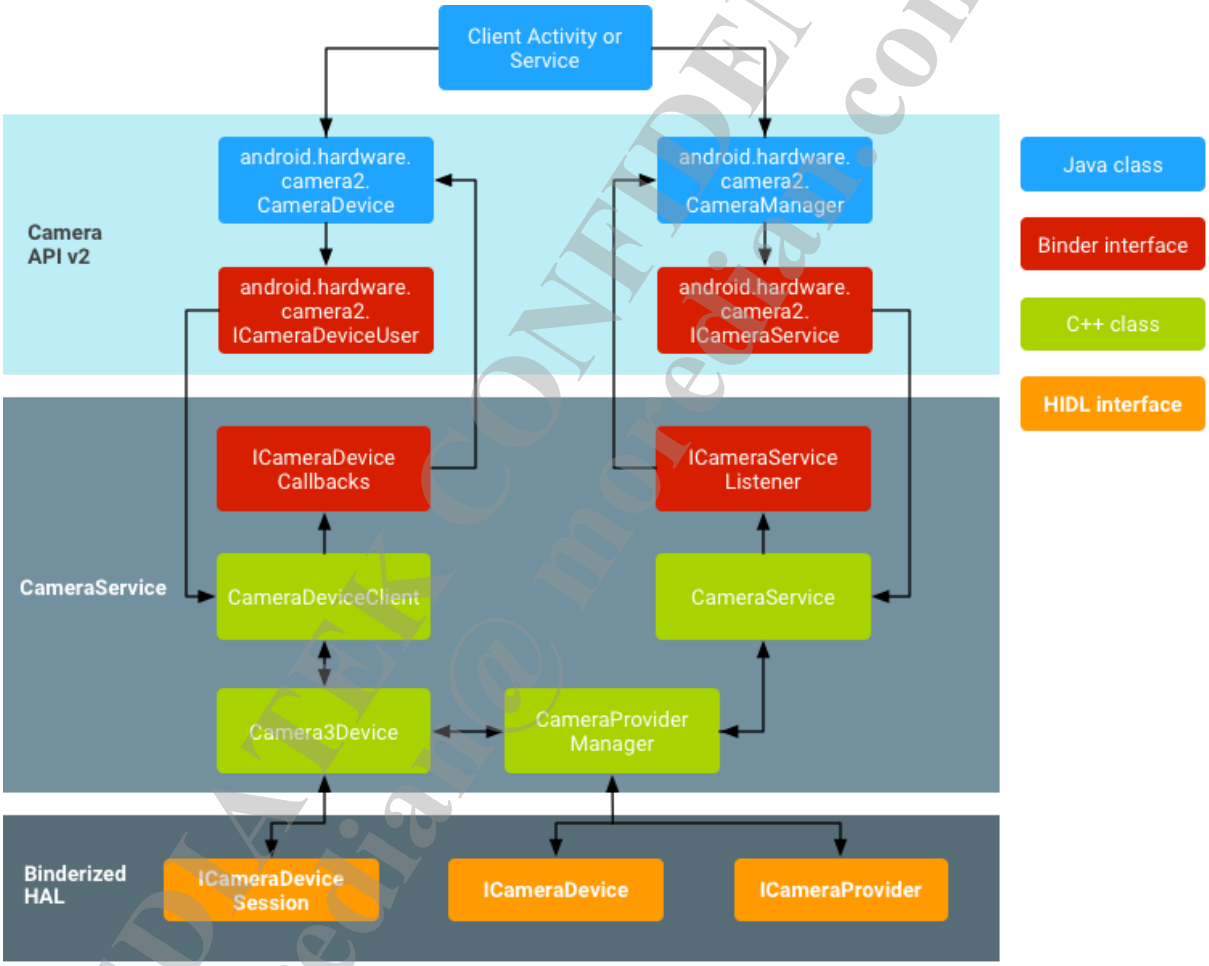


圖 1Android Camera architecture

1.1 Overview

本文件包含以下幾個部份

1. API1/API2 + HAL1/HAL3 Supporting on Android P
2. Camera HAL Service in Android P
3. HAL3 High Level Architecture

4. HAL3 Pipeline Architecture
5. HAL3 Basic operation

1.2 Scope

本文件所包含模塊及支援範圍:

a) 支援 Android 版本: Android P + HAL3

b) 支援平台: ISP3+ (MT6739, MT6761, MT6763, MT6765/6762, MT6771).

1.3 Target Audience

本文件主要提供給需要了解 MTK Camera HAL3 架構的客戶/ACS。

1.4 Definitions, acronyms, and abbreviations

HAL: Hardware Abstract Layer
TPI: Third Party Interface
API1: android.hardware.camera API
API2: android.hardware.camera2 API

2 SW Architecture

2.1 Overview

Android Camera 架構如下圖，AOSP layer 依序為 Camera apk 、Java SDK(use Camera2 API)、Camera Service and HIDL interface，往下藍色區塊為 Camera HAL，分為 Middleware、Pipeline(含 HWNode)、與 FeaturePipe。

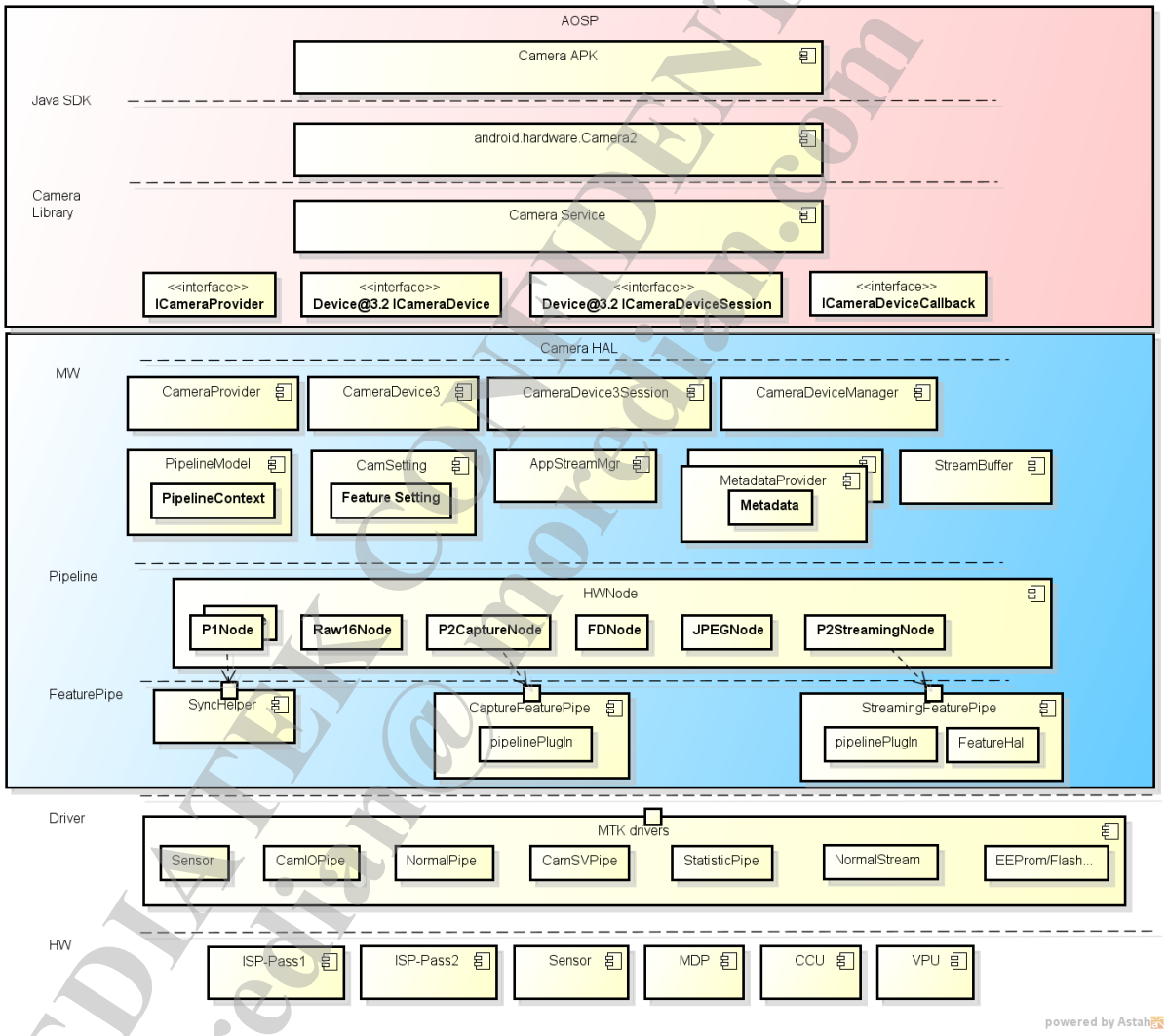


圖 2 Overall SW stack

以下會依序介紹:

- API1/API2 與 HAL1/HAL3 的關係
- 簡介 Camera HAL interface
- 簡介 Logical device 與 Physical device 的差別
- 簡介 Pipeline architecture

2.2 API1/API2 + HAL1/HAL3 Supporting on Android P

本節說明在 Android P 上 MTK 是如何分別支持 Camera HAL1 與 HAL3，Camera apk 使用 API1 或 API2 會如何對應到 Camera HAL 版本。

如下圖所示:

- API1 + HAL1 只支持舊案升級(如 Android O -> Android P)，有完整的 HAL1 MTK TK features。
- 新開案只支持 HAL3，建議使用 API2，有完整的 HAL3 MTK TK features。
- API1 + HAL3 支持 AOSP 基本流程，不支持 MTK TK features。

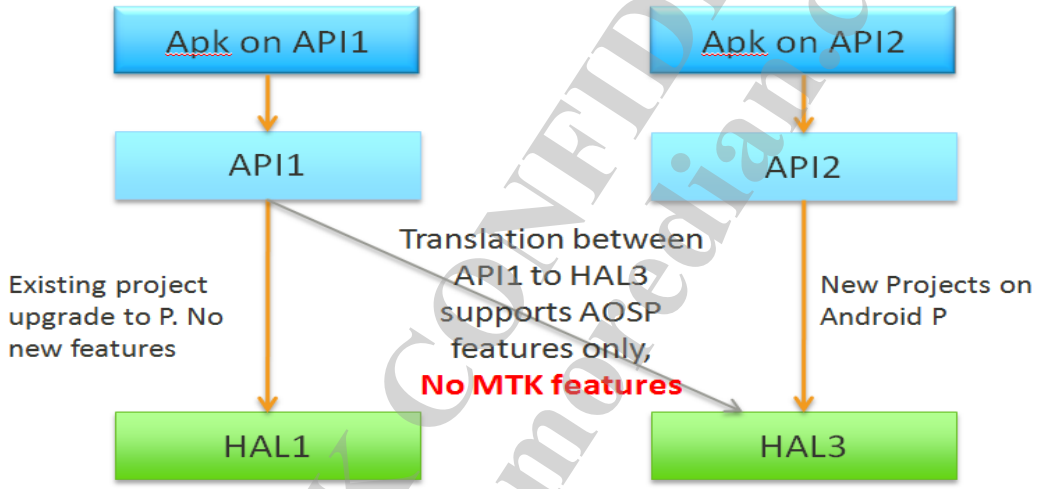


圖 3 API1/API2 與 HAL1/HAL3

2.3 Camera HAL interface

本節說明 Camera HAL interface: ICameraProvider、ICameraDevice 與 ICameraDeviceSession 間的互動關係。

Camera Provider HAL Interface

- Camera provider HAL which enumerates the available individual camera devices known to the provider,
  and provides updates about changes to device status, such as connection, disconnection, or torch mode enable/disable.
- The provider is responsible for generating a list of camera device service names that can then be opened later.

Camera Device HAL Interface

- The camera device HAL interface is used by the Android camera service to operate individual camera devices. Instances of camera device HAL interface can be obtained via one of the

ICameraProvider::getCameraDeviceInterface_VN_x() methods, where N is the major version of the camera device interface.

Camera Device Session HAL Interface

- The camera device session HAL interface is created by Camera Device HAL3.

對比 Android N 的 module hal，Camera Provider HAL 並沒有 open API，取而代之的是 get API，此 API 的目的是為了取得 Camera Device HAL1 or Camera Device HAL3 的 instance，且 get API 本身不會進行 power on 動作，當上層 App 取得此 Device instance 後，才會個別透過 Camera Device 所提供的 Open API 來做 power on 動作。

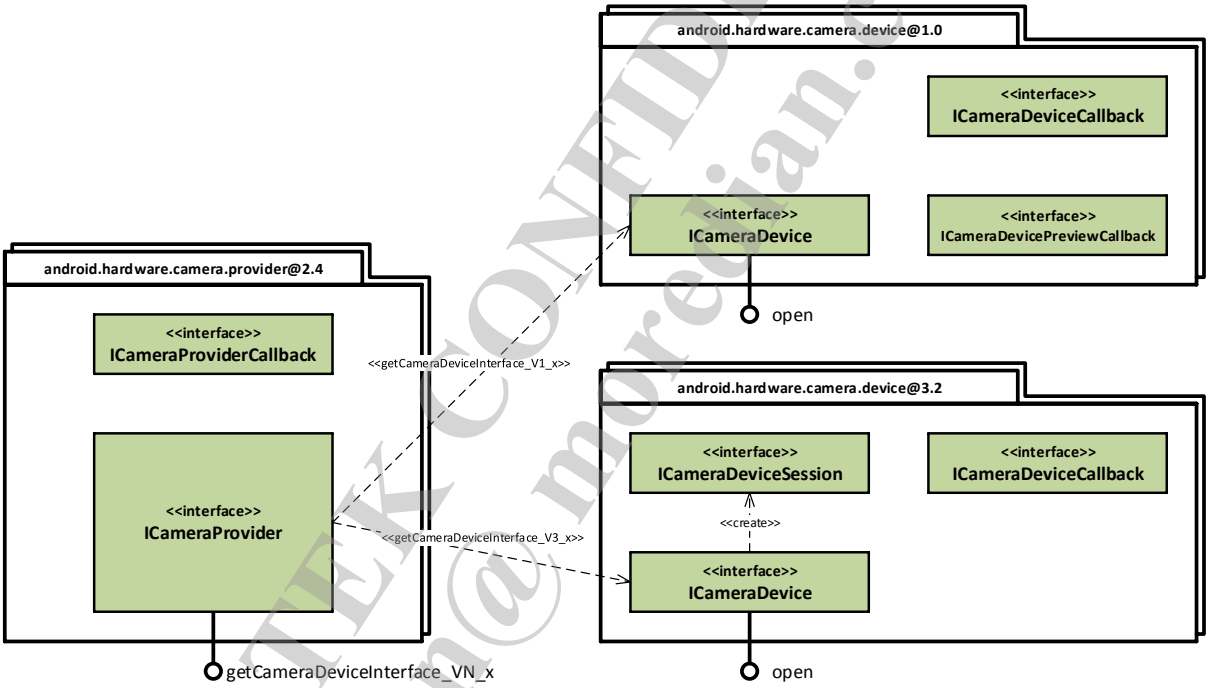


圖 4 Camera HAL interface

2.4 Physical Device and Logical Device

- 在 Android N 過去版本，Camera HAL 會回傳 physical device，也就是說有幾個 physical sensor 就會回傳幾個 device。
- 從 Android O 開始導入了 Logical device 的概念，Logical Device 顧名思義和 physical sensor 並非一對一 mapping，目前有兩種使用方式。
  - 同一個 physical sensor 有不同的 HAL 版本可回傳不同的 Logical Device。舉例，若後鏡頭可支持 HAL1 與 HAL3 則會回傳兩個 Device 給 framework。
  - 多個 physical sensor 對應到一個 Logical Device。舉例，如 dual camera 的應用。(Logical multi-device)

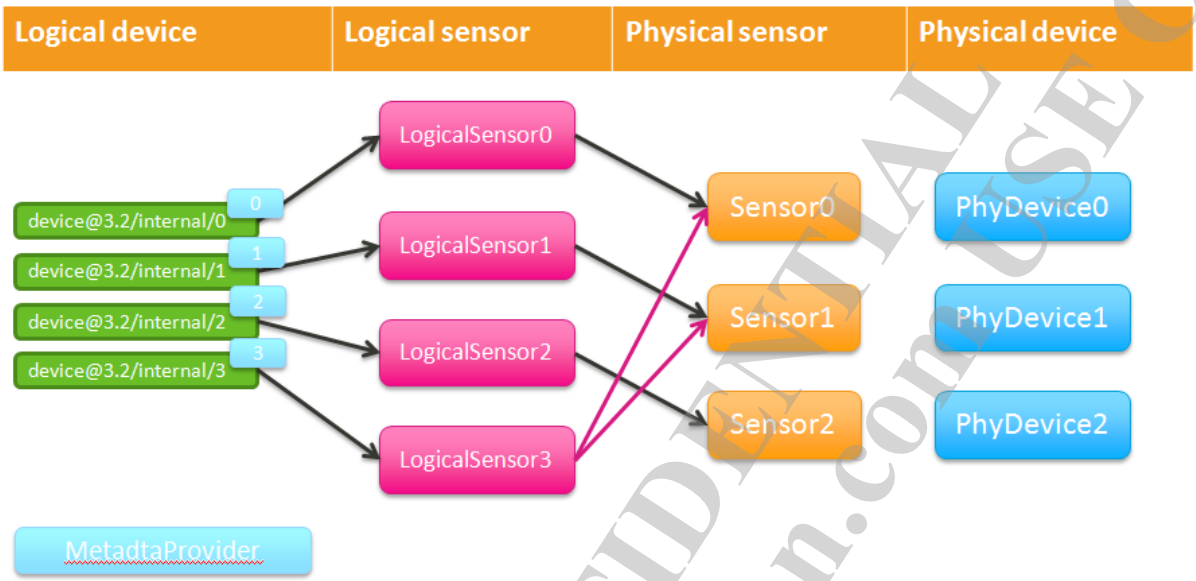


圖 5 Logical device 與 Physical sensor

Module Description

- Physical sensor : HW Camera sensor
- Physical device : Physical Camera device, one-one mapping to Physical sensor
- Logical sensor : “Logical” means that one sensor mapping to 2 physical sensor or one sensor may support different capability (e.g. low power version)
    - one-one mapping to MetadataProvider
- Logical device: The devices of currently connected camera devices by identifier, including cameras that may be in use by other camera API clients.
    - Number: Logical device >= Logical sensor

2.5 Pipeline architecture

本節介紹 camera pipeline pipeline 架構，pipeline high level component 由 pipelineModel, HWNode 與 FeaturePipe 所組成，以 SW stack 來看可參考下圖紅色虛線框部分，其中:

pipelineModel 會針對不同 user scenario 來創建對應的 pipeline 與 HWNode，而 request status 亦針對各 request 需求決定 pipeline path, HWNode input/output buffer 與 metadata，PipelineModel 細部介紹會在後續文件說明 (document name: TBD)。

- input : App request
- output: Image streams and metadata result

HWNode: 負責整合 MTK hardware functionality, software algorithm, 3rd party algorithm，各 HWNode/FeaturePipe 細部介紹可參考”MTK camera P2CaptureNode introduction”與”MTK camera P2StreamingNode introduction “…等 HWNode 介紹文件。

- P1Node:
  - The root node of the pipeline
  - The input data is from upper layer
  - The output data is to the next P2CaptureNode and P2StreamingNode
  - Uses ISP-P1
  - Get / Set Hal3A module
- P2CaptureNode
  - Process raw data into yuvs
  - Support scale/crop
  - Uses ISP-P2 & MDP
  - Handle all MTK TK and 3rd party capture features (with CaptureFeaturePipe)，3rd party 整合細部介紹可參考” MTK camera plugin interface user guide”
- P2StreamingNode
  - Process raw data into yuvs
  - Support scale/crop
  - Uses ISP-P2 & MDP
  - Handle all MTK TK and 3rd party streaming features(with StreamingFeaturePipe) ，3rd party 整合細部介紹可參考” MTK camera plugin interface user guide”
- JPEGNode
  - Convert YUV to Jpeg
  - Input data from P2StreamingNode or P2CaptureNode
  - Output data to application
- FDNode
  - Generate the FD information

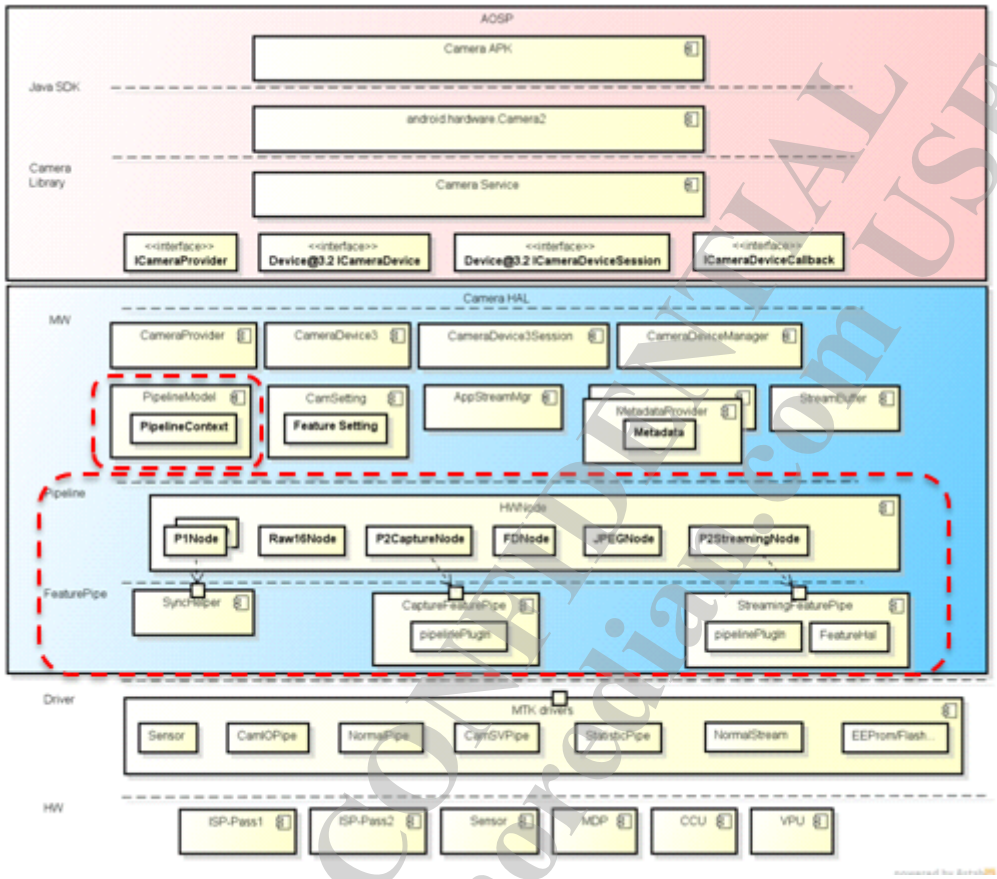


圖 6 pipeline components of overall sw stack

以 dual camera scenario 為例，一個 physical sensor 會對應一個 P1Node，所以 pipeline 會由兩個 P1Node 所組成，其餘的 HWNode 與 single camera scenario 一致。

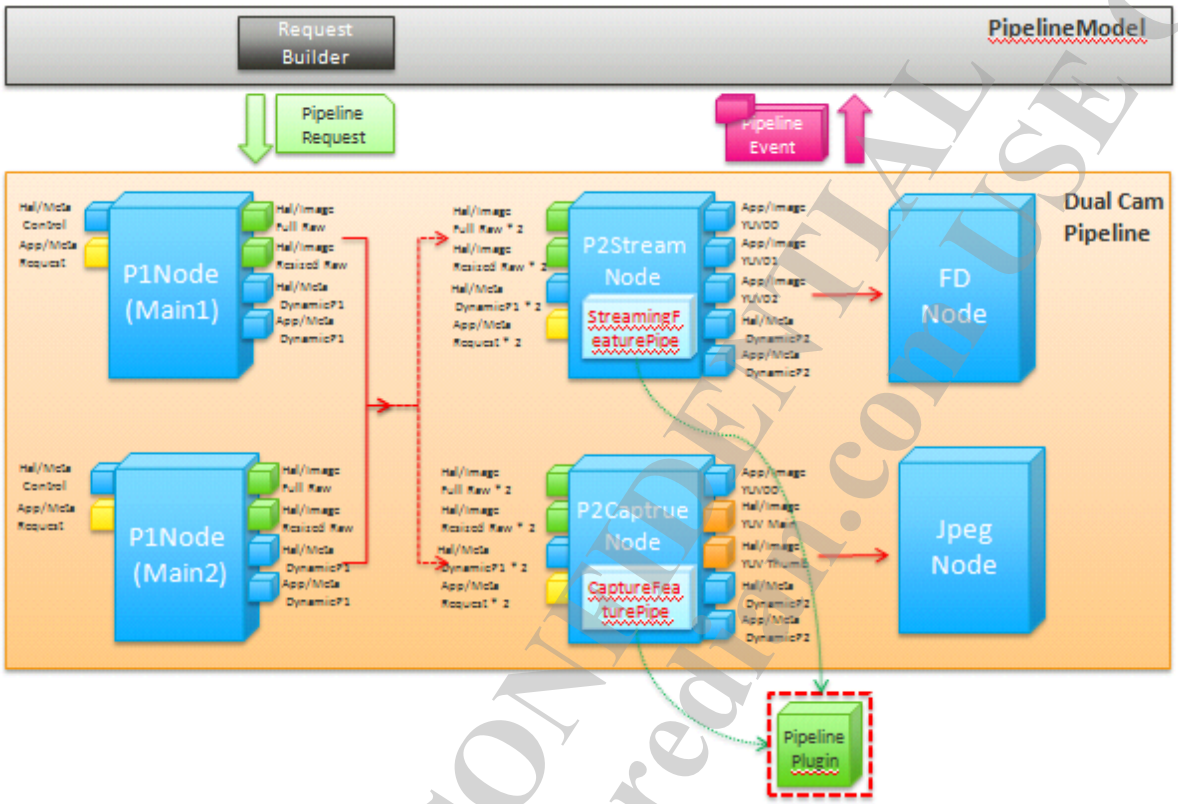


圖 7 dual camera pipeline

2.6 Control Path

本節介紹 Camera 基礎行為，包含 get camera devices、open/close camera device 與 flash 開啟，首先簡介相關模組:

- Camera Device Manager
  - Camera Device Manager is a core implementation, used to manage camera devices, including devices enumeration, opening, and closing.
- Camera Device
  - Camera Device exposes ICameraDevice interface for Android Camera Service to operate each camera.
  - Camera Device exposes IVirtualDevice interface for Camera Device Manager to interact with.
- Camera Provider
  - ICameraDevice interface is obtained by Android Camera Service via ICameraProvider interface.
  - The implementation of Camera Provider is just an adapter which wraps Camera Device Manager.
- Camera HAL Service
  - A standalone process used in Binderized mode, where ICameraProvider is registered.
- Camera Service

– A client of Camera HAL (Service).

2.6.1 Enumerating devices while registering Camera HAL Service

本節描述當註冊 Camera HAL service 時所觸發的 search sensor 的流程:

1. 當 Camera HAL service lunch 時會透過 registerPassthroughServiceImplementation 註冊 Camera Provider
   a. using android::hardware::camera::provider::V2_4::ICameraProvider; registerPassthroughServiceImplementation<ICameraProvider>("internal/0" /*"internal" for binderized mode*/);
2. registerPassthroughServiceImplementation 會呼叫 ICameraProvider::getService API 來取得 ICameraProvder instance。
   a. sp<ICameraProvider> getService(const std::string &serviceName="default", bool getStub=false);
3. 再透過 get API 呼叫到 HIDL_FETCH_ICameraProvider，此為 AOSP 定義的 interface 而 MTK 所實做，目前實做會 create Camera Provider 也同時透過 getCameraDeviceManager 來 create CameraDeviceManager。
4. CameraDeviceManager 在 init 過程就會透過 enumerateDevicesLocked 來進行 search sensor 動作。而 2.4 章所提到的 Logical sensor 也是在這個時間決定好的。

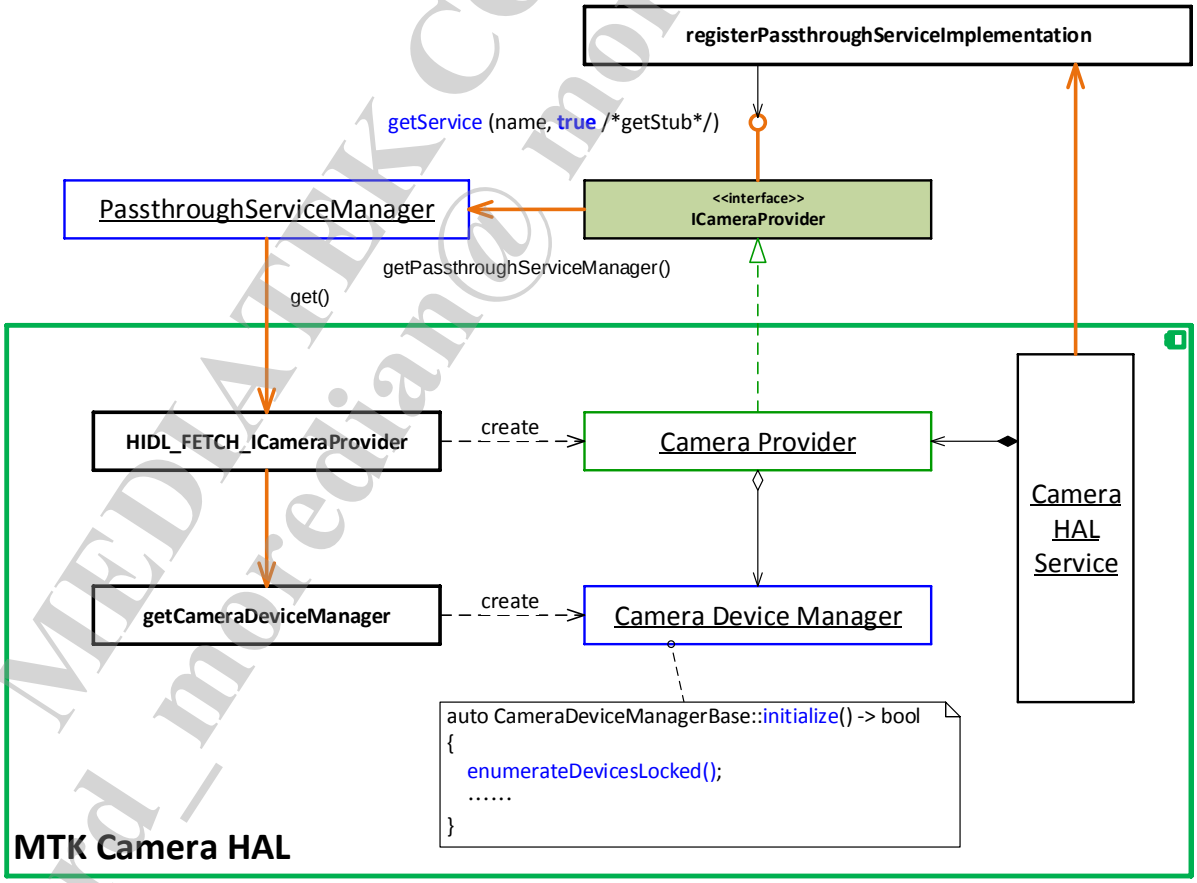


圖 8 Enumerating devices while registering Camera HAL Service

2.6.2 CameraDevice::open

本節介紹 CameraDevice open 行為，CameraDevice 會先透過 CameraProvder::getCameraIdList、getCameraDeviceInterface_V1_x、getCameraDeviceInterface_V3_x 取得 device id 與對應的 Device instance，CameraService 可決定要使用哪一個 CameraDevice。

1. 呼叫 ICameraDevice 所提供的 open API
2. CameraDevice 內部實做:
   2.1. CameraDevice 要開始進行 open 相關行為前會先透過 startOpenDevice function 和 CameraDeviceManager 註冊，CameraDeviceManager 會確認該 Device 是否有重覆 open 的行為。
      i. CameraDeviceManager attach the device to open map
      ii. Lock flash and turn off torch when 1st camera opened
   2.2. 進行這個 device 相關初始設定(e.g. sensor power on, 3A init)。
   2.3. 通知 CameraDeviceManager open 結束(by finishOpenDevice)。
      i. Device open 結束後，會透過 ICameraProviderCallback:: torchModeStatusChange API 來通知 CameraService torchMode 狀態改變，以此例來說要通知 CameraService torch mode 已經變成不可利用的狀態。

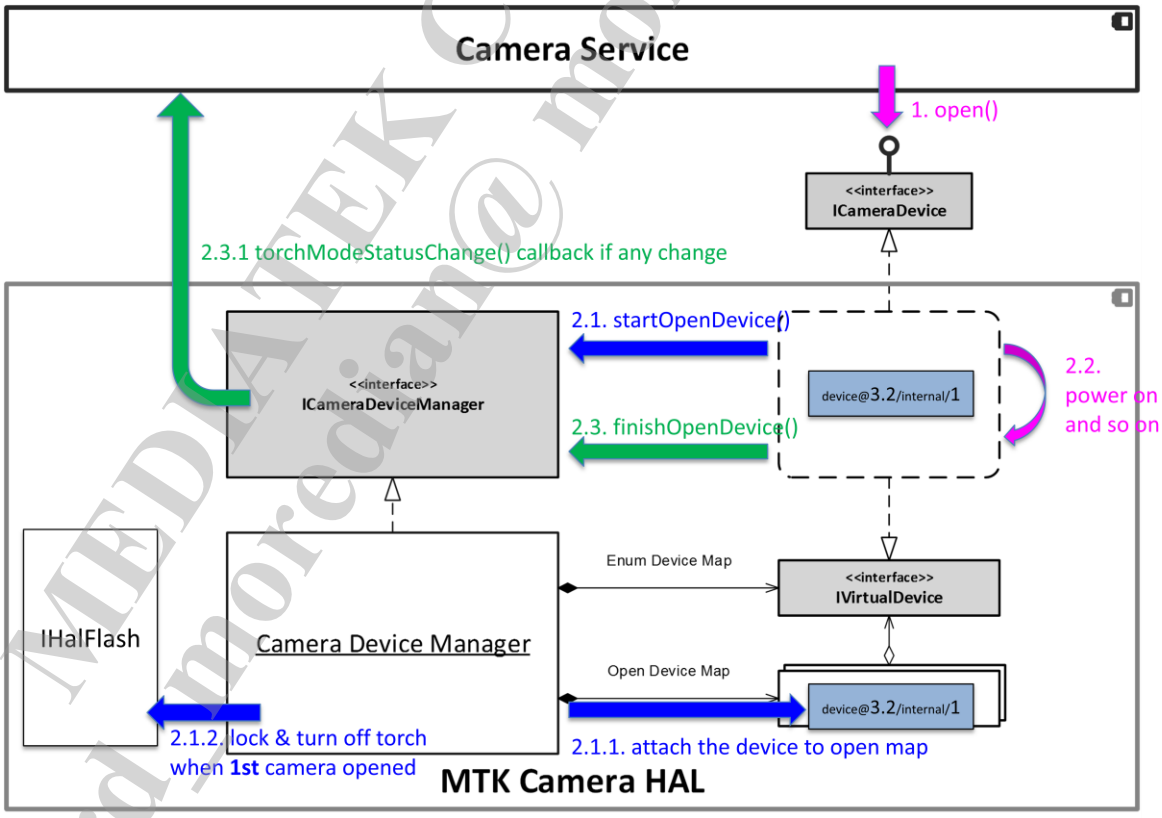


圖 9 Device open

2.6.3 CameraDevice::close

類似 Device open，CameraService 可針對某個 CameraDevice 進行 close 動作:

1. 呼叫 ICameraDevice 所提供的 close API
2. CameraDevice 內部實做:
   2.1. CameraDevice 要開始進行 close 相關行為前會先透過 startCloseDevice function 和 CameraDeviceManager 註冊。
      i. CameraDeviceManager detach the device from open map
      ii. Unlock torch when all cameras closed
   2.2. 進行這個 device 相關 uninit 設定(e.g. sensor power off, 3A uninit)。
   2.3. 通知 CameraDeviceManager open 結束 (by finishCloseDevice)。
      i. Device close 結束後，若 torchMode 有改變仍會透過 ICameraProviderCallback:: torchModeStatusChange API 來通知 CameraService torchMode 狀態改變，以上例而言若所有 CameraDevice 都 close 了，會通知 CameraService torch 已變成可利用狀態。

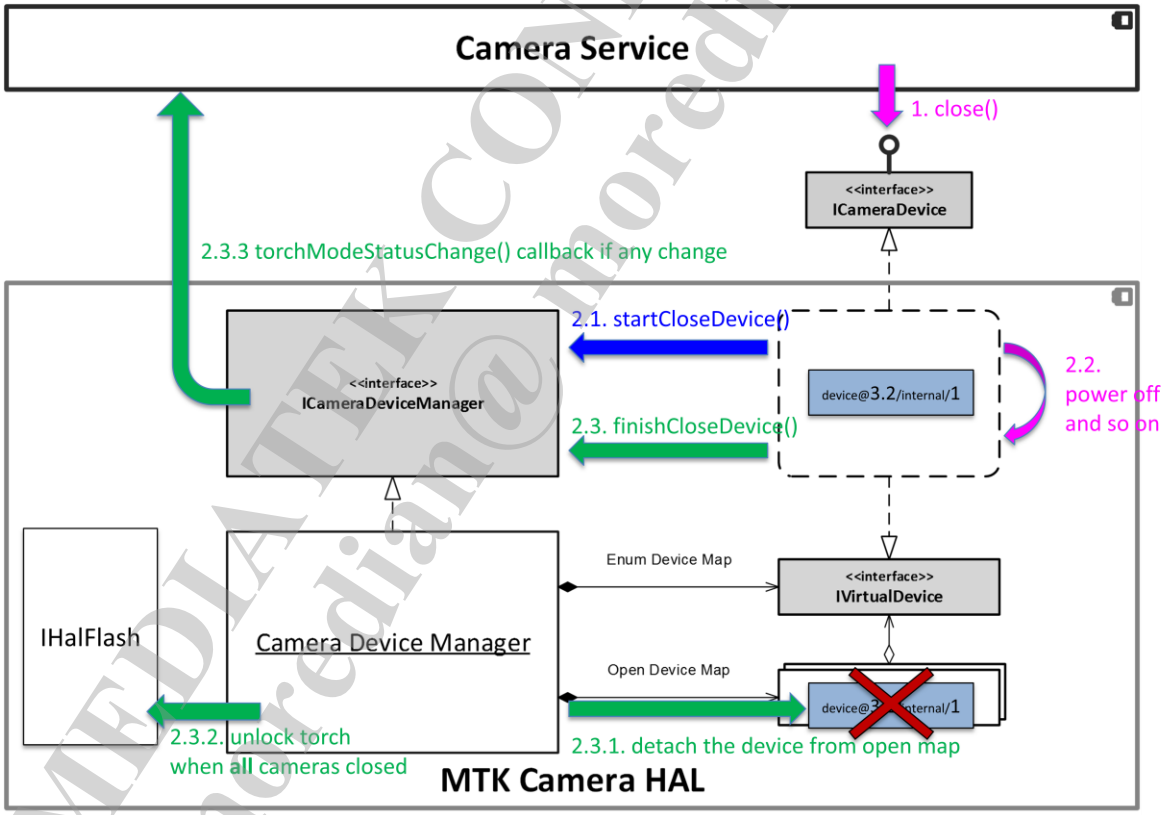


圖 10 Device close

2.6.4 CameraDevice::setTorchMode

TorchMode 也是要透過 ICameraDevice 所提供的 Interface 來控制，相關流程如下圖所示:

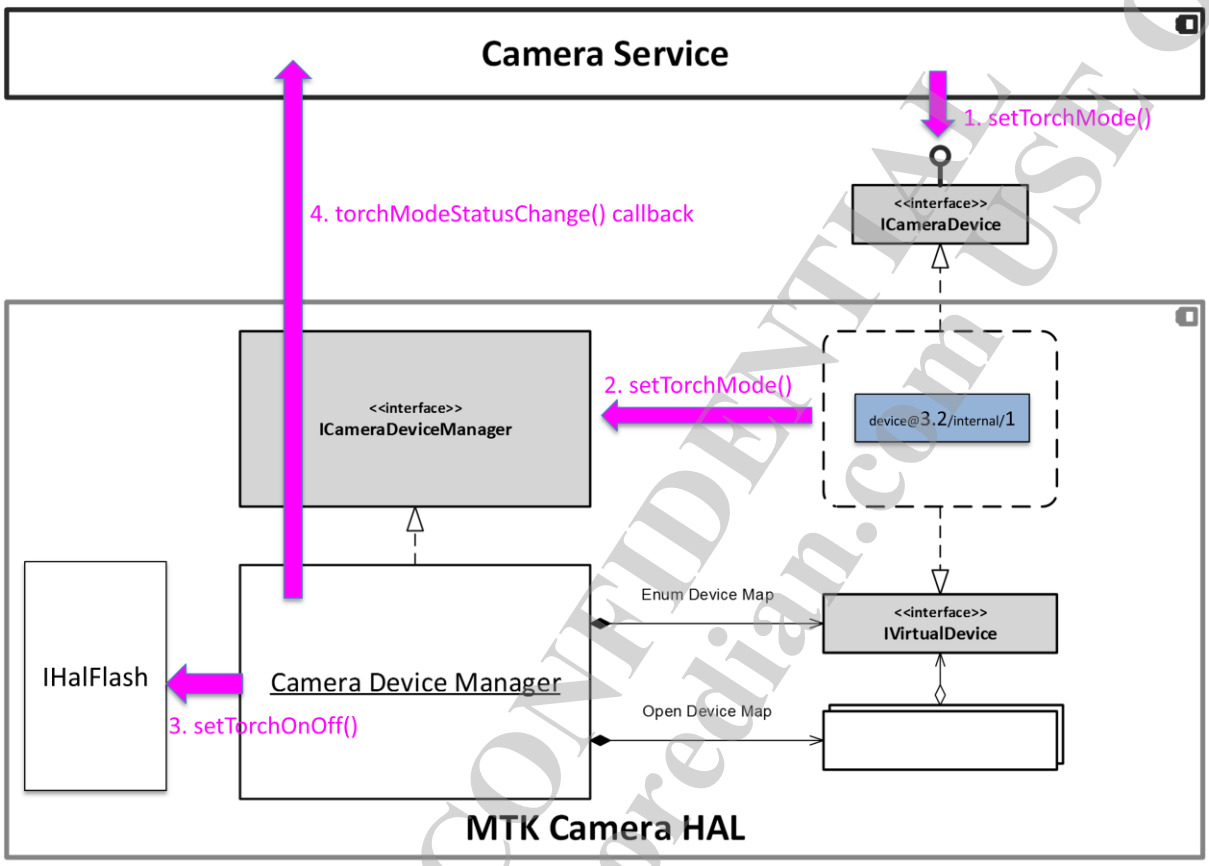


圖 11 setTorchMode

2.7 Functionality

2.7.1 HAL3 HAL APIs

下表對比 Andrid N 以前的 Module HAL / Device HAL，與 Android O/P 的 Provider HAL / Device HAL 3.4 的 API level 差別:

Conventional HAL (Android N & before)	HIDL HAL (link)
Module HAL	**Provider HAL**
init	
set_callbacks	setCallback
get_vendor_tag_ops	getVendorTags
get_number_of_cameras	getCameraIdList
	isSetTorchModeSupported
	getCameraDeviceInterface_V1_x getCameraDeviceInterface_V3_x

	Device HAL 3.4 (ICameraDevice.hal)
get_camera_info	getResourceCost getCameraCharacteristics
set_torch_mode	setTorchMode
hw_module_t::hw_module_methods_t::open	open
open_legacy	
Device HAL3 (camera3_device_ops)	
dump	dumpState
	Device HAL 3.4 (ICameraDeviceSession.hal)
construct_default_request_settings	constructDefaultRequestSettings
configure_streams	configureStreams
process_capture_request	processCaptureRequest
	getCaptureRequestMetadataQueue
	getCaptureResultMetadataQueue
flush	flush
hw_device_t::close	close
initialize	
register_stream_buffers get_metadata_vendor_tag_ops	
Device HAL 3 (camera3_callback_ops)	**Device HAL 3.4 (ICameraDeviceCallback.hal)**
process_capture_result	processCaptureResult
notify	notify

3 Detail Design

3.1 Folder and file structure

TBD.

3.2 SW block diagram & Class diagram

TBD.

3.3 Sequence Diagram

3.3.1 Camera HAL3 operation

本節描述從 camera device open 到 config/request stage 時所使用到的 HAL API 與 HAL modules，以下為 camera service 所使用到的 API。

- CameraDevice::open
- CameraDeviceSession::
  - configureStreams()
  - constructDefaultRequestSettings()
  - getCaptureRequestMetadataQueue()、getCaptureResultMetadataQueue()
  - processCaptureRequest()

相關模組簡介如下:

- Device HALv3: include Camera Device and Camera Device Session.
  - Camera Device
    - Camera Device exposes ICameraDevice interface for Android Camera Service to operate each camera.
    - Camera Device exposes IVirtualDevice interface for Camera Device Manager to interact with.
  - Camera Device Session.
    - Camera Device exposes ICameraDeviceSession interface for Android Camera Service to operate each camera.
- App Stream Manager
  - Callback result metadata and filled image streams by IcameraDeviceCallback interface to Android Camera Service.
  - Used to manage App streams and their associated stream buffers.
- Pipeline Model Manager
  - Used to manage pipeline model.

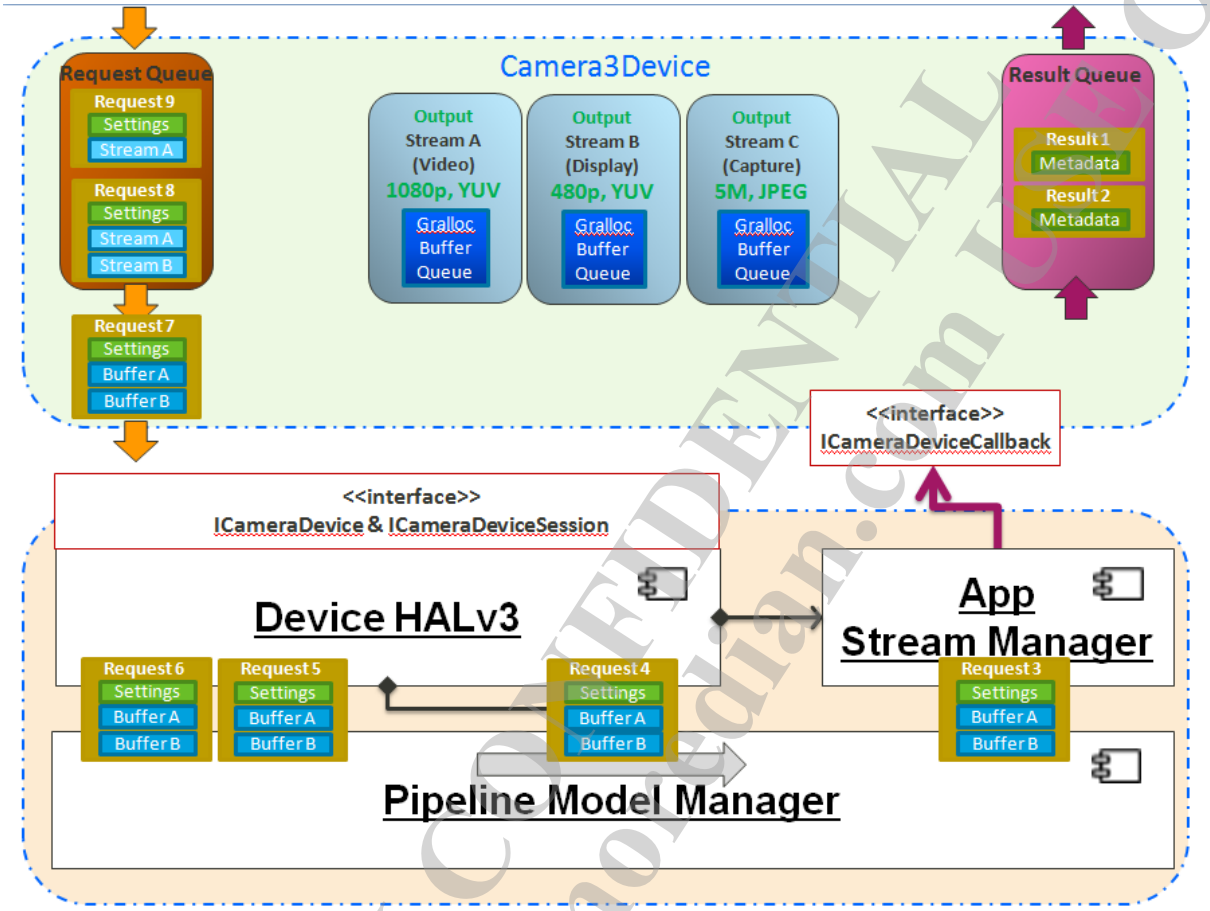


圖 12 Camera Device, AppStreamManger 與 PipelineModelManager 關係圖

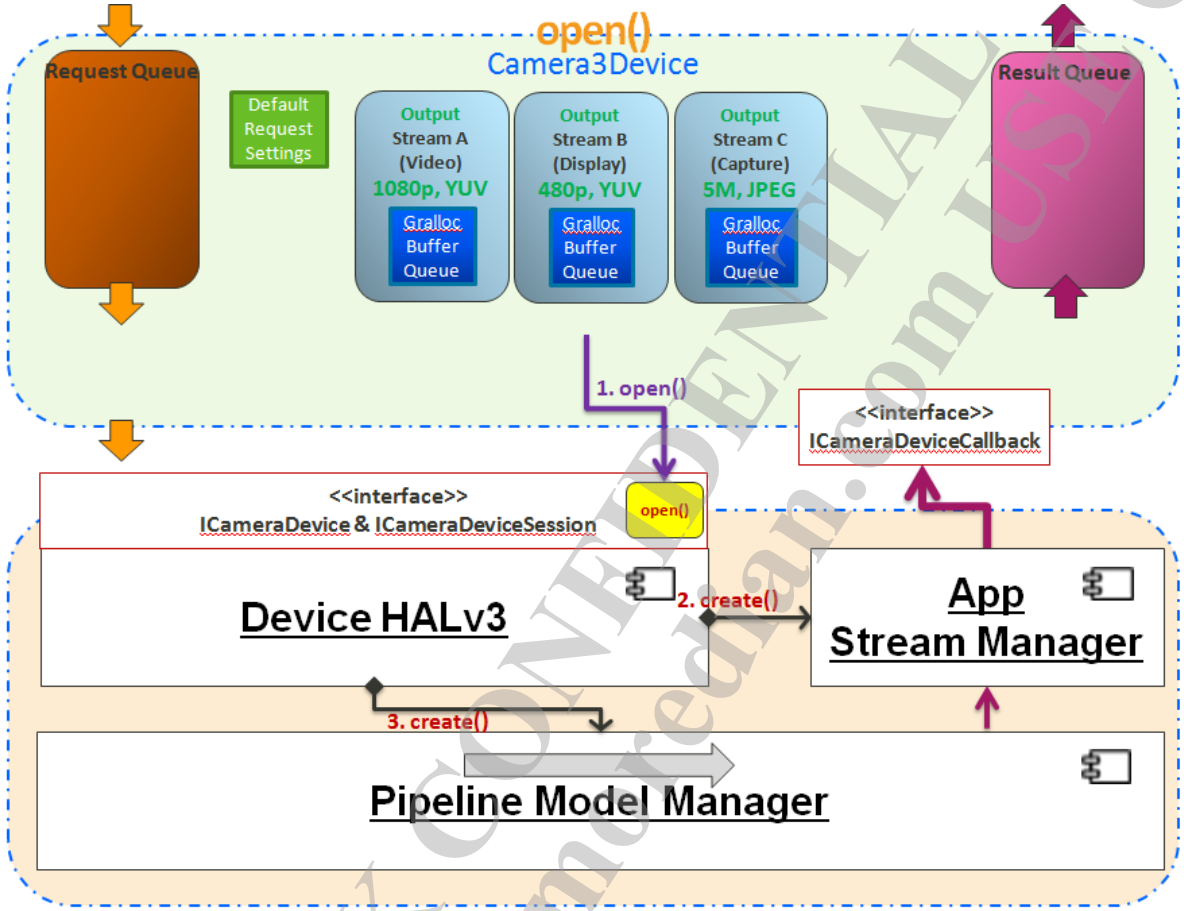


圖 13 open flow

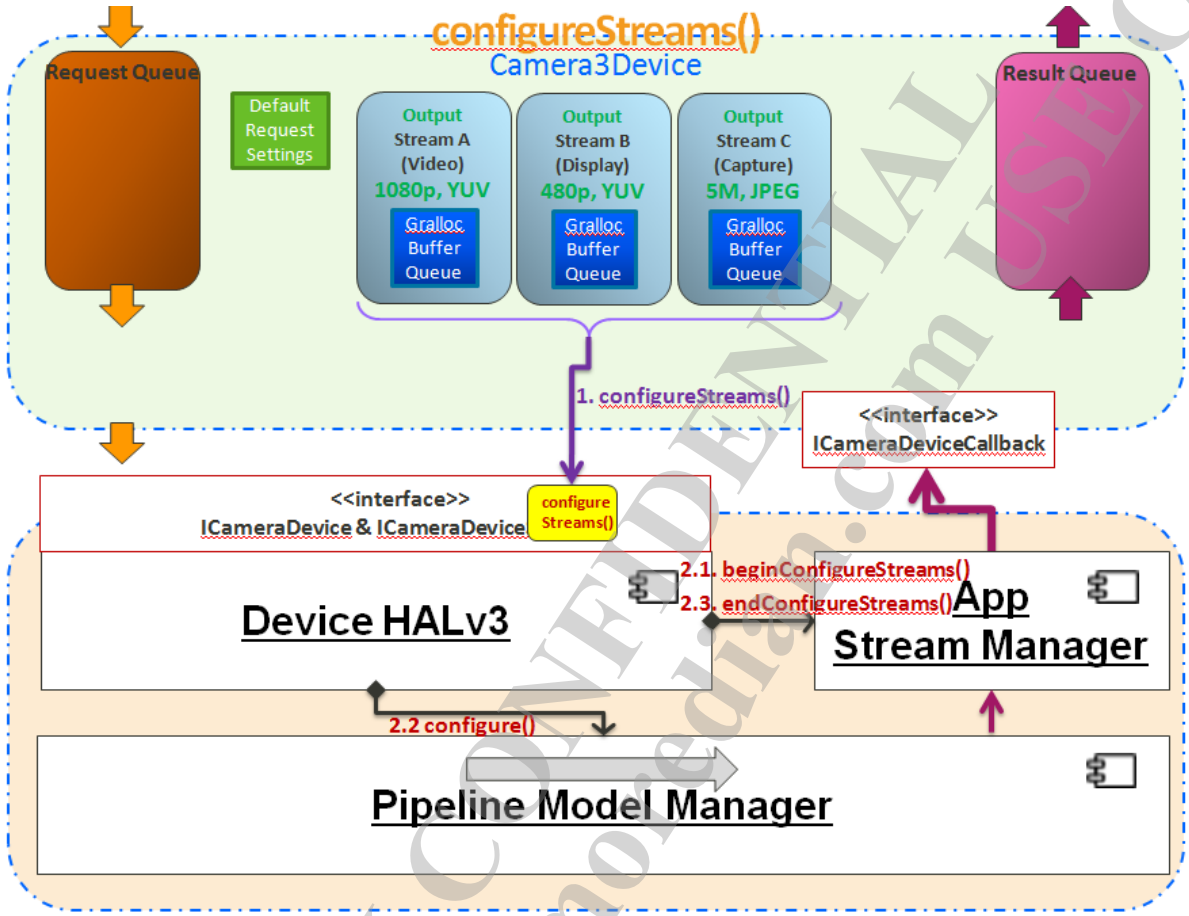


圖 14 configStreams flow

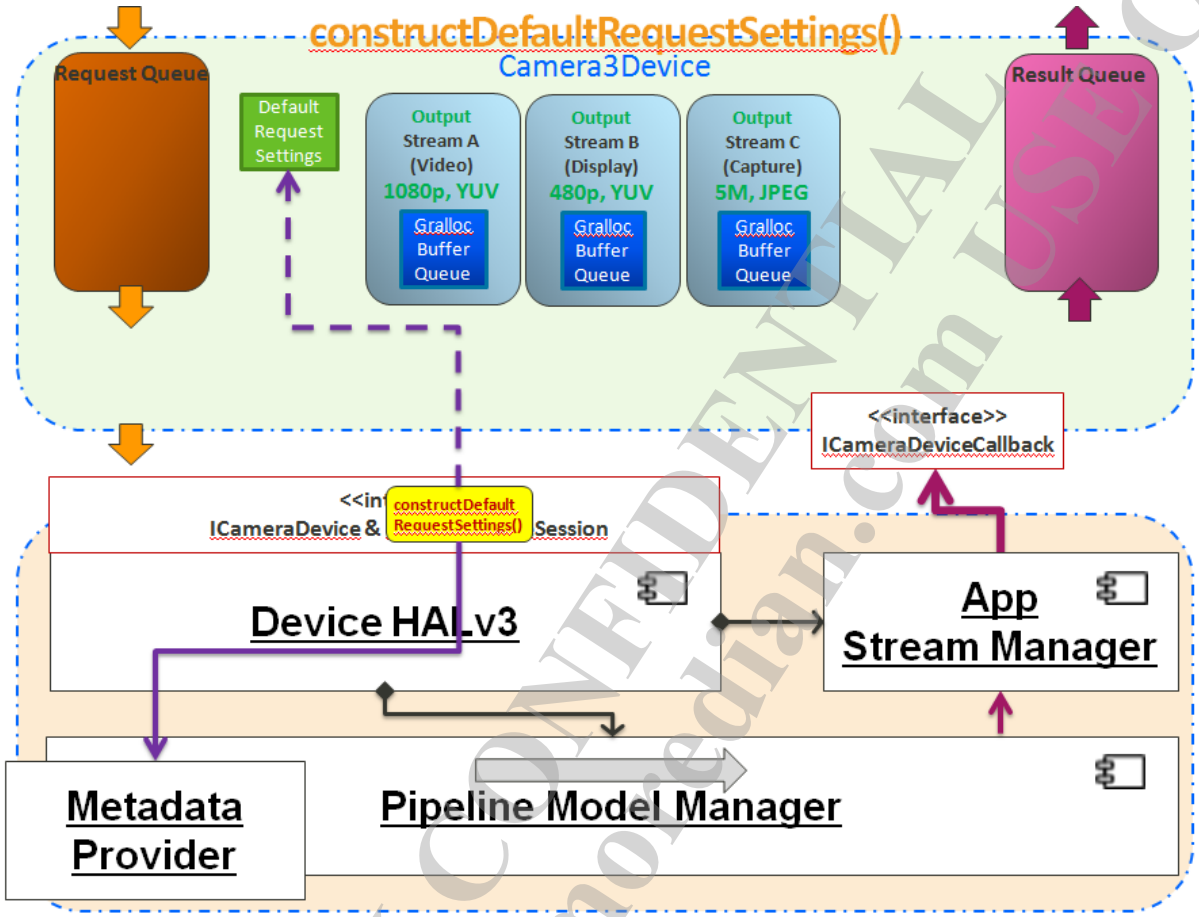


圖 15 constructDefaultRequestSettings flow

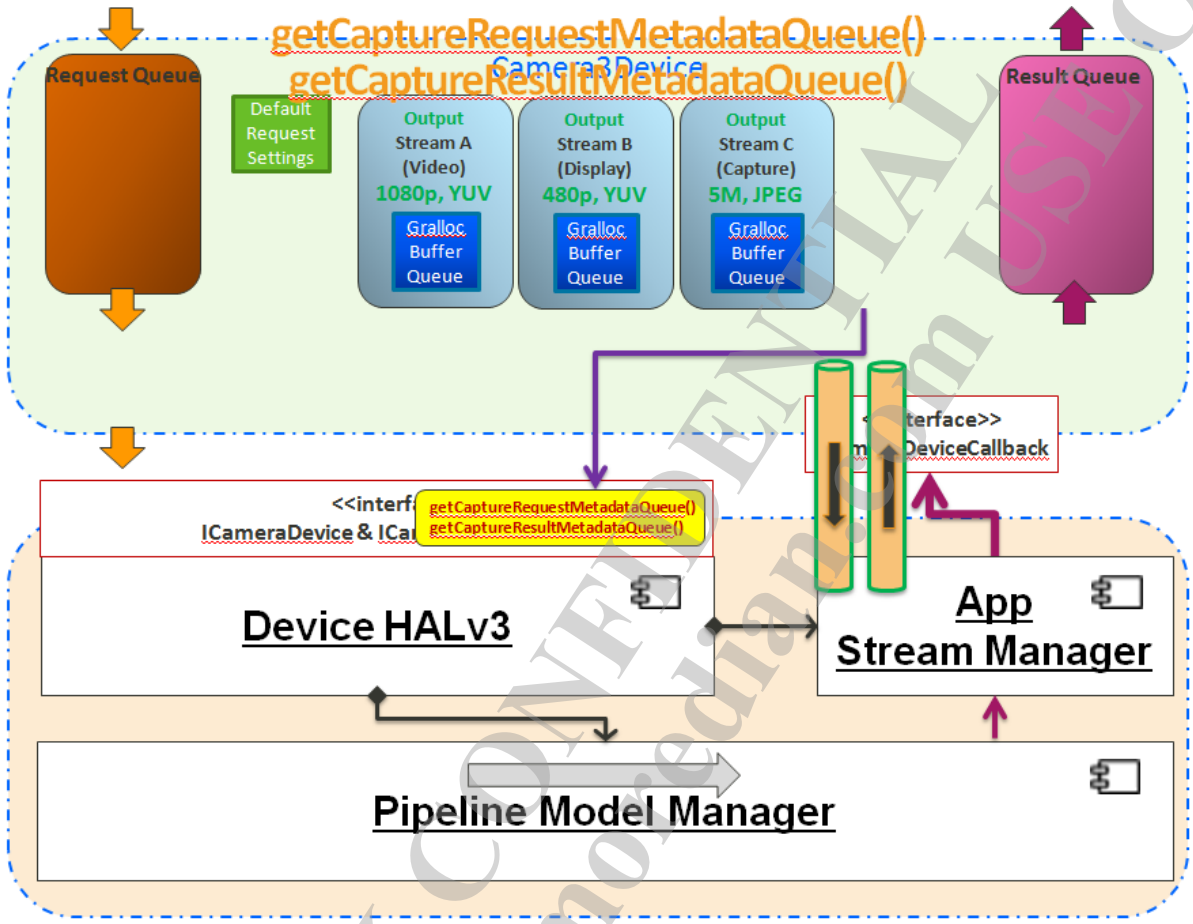
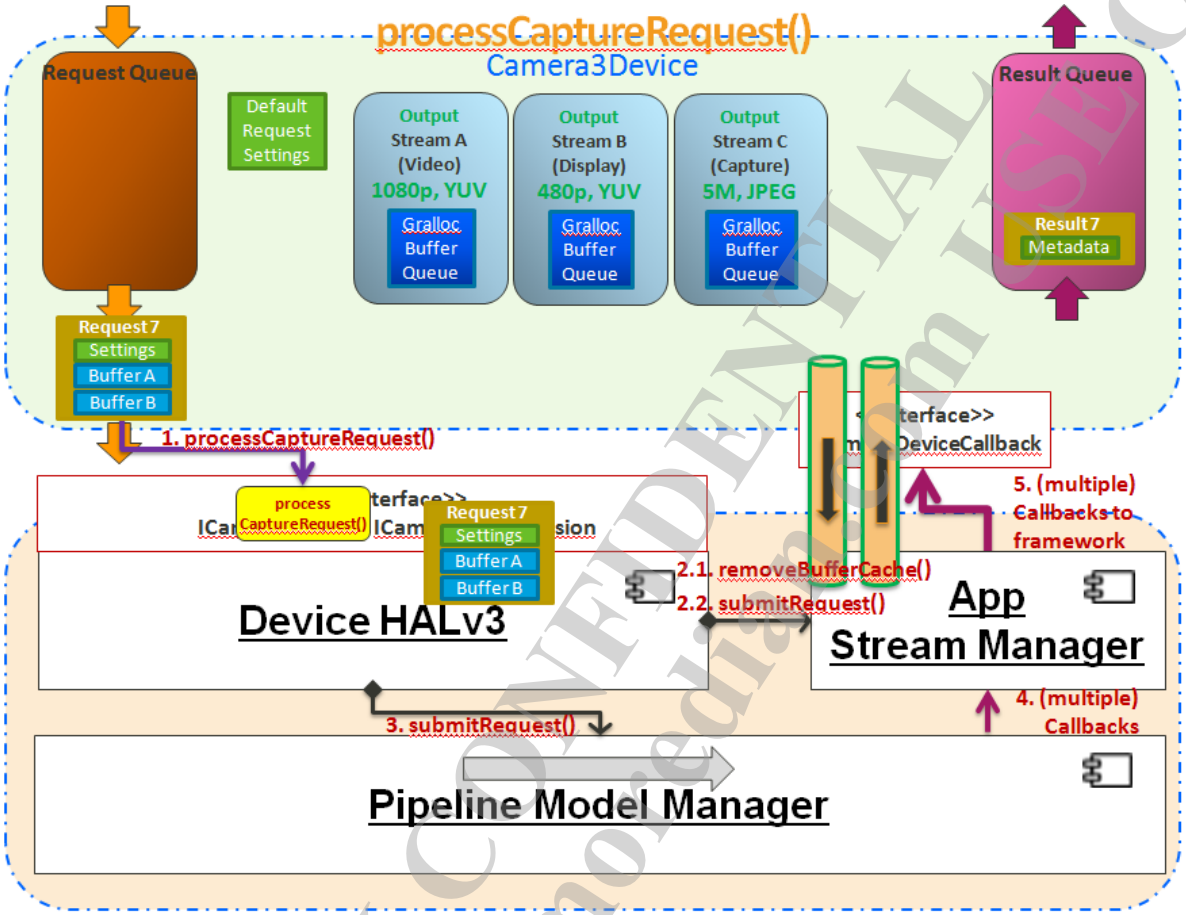


圖 16 getCaptureRequestMetadataQueue 與 getCaptureResultMetadataQueue



17 processCaptureRequest flow