DVPP API Reference

Issue 01

Date 2020-05-30

HUAWEI TECHNOLOGIES CO., LTD.

Trademarks and Permissions


and other Huawei trademarks are trademarks of Huawei Technologies Co., Ltd.
All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Contents

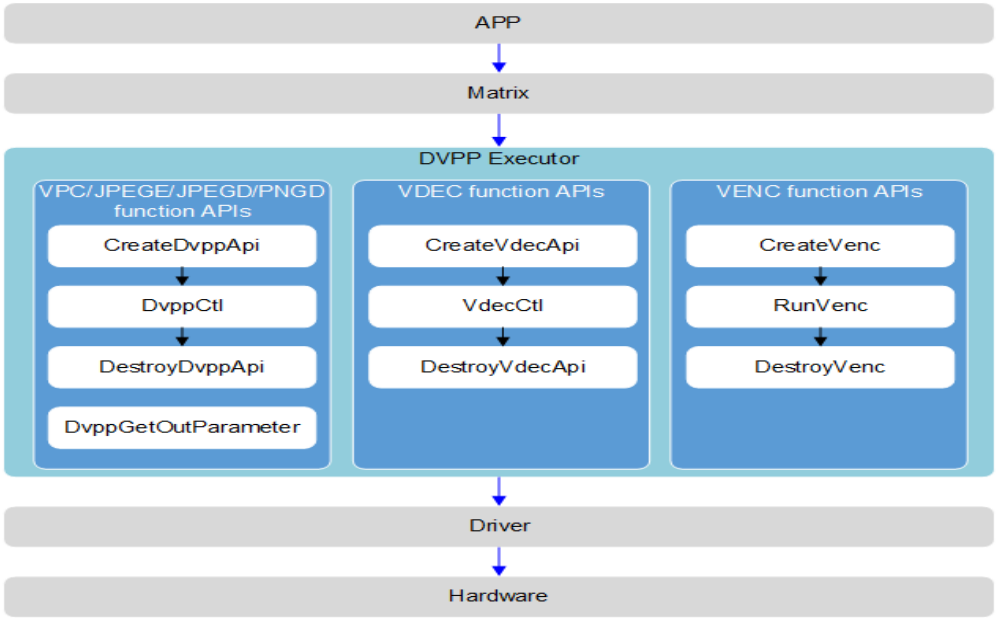
1 Overview

1.1 API Introduction

This document describes the external APIs of the digital vision pre-processing (DVPP) executor. The API functions, API calling guidance, and API usage examples are provided for developers and test engineers.

The process of calling the DVPP APIs during application development is as follows:

Figure 1-1 Process flowchart



- In the current version, the VPC function can be implemented by **CreateDvppApi**, **DvppCtl**, and **DestroyDvppApi** in either of the following methods:
  - Method 1: Pass input parameters to **DvppCtl**, that is, the **DVPP_CTL_VPC_PROC** command word and **VpcUserImageConfigure** structure parameters. **This method is recommended.**
  - Method 2: Pass input parameters to **DvppCtl**, that is, the **DVPP_CTL_VPC_PROC** command word and **resize_param_in_msg** structure parameters. In scenarios that have low requirements on latency and process a large number of low-resolution images, pass input parameters to **DvppCtl**, that is, the **DVPP_CTL_CMDLIST_PROC** command word and **IMAGE_CONFIG** structure parameters. **This method is not recommended.** It is used to be compatible with the functions in earlier versions and will be deleted in later versions.
- In the current version, the VENC function can be implemented in either of the following methods:
  - Method 1: Call **CreateVenc**, **RunVenc**, and **DestroyVenc** to implement the VENC function. **This method is recommended.**
  - Method 2: Call **CreateDvppApi**, **DvppCtl**, and **DestroyDvppApi** to implement the VENC function. Pass input parameters to **DvppCtl**, that is, the **DVPP_CTL_VENC_PROC** command word and **venc_in_msg** structure parameters. **This method is not recommended.** It is used to be compatible with the functions in earlier versions and will be deleted in later versions.

1.2 API List

You can view the header files of the APIs in the **ddk/include/inc/dvpp/** installation directory of the device development kit (DDK). If the APIs provided by the DVPP need to be called, the code can contain **idvppapi.h**, **Venc.h**, and **Vpc.h**.

For details about the header files that define data types, see **7.1 Structures in VpcUserImageConfigure**.

Table 1-1 API list

Category	API	Function	Header File
Implementing the VPC, JPEGE, JPEGD, and PNGD functions	**CreateDvppApi**	Creates a DVPP API instance, which is equivalent to creating the handle to the DVPP executor. The caller can use the applied DVPP API instance to call **DvppCtl** to process an image, either across functions or across threads.	idvppapi.h
	DvppCtl	Controls the execution of DVPP modules, such as the VPC, JPEGE, JPEGD and PNGD. The **DvppCtl** API is called by using the instance created by **CreateDvppApi**.	
	DestroyDvppApi	Destroys the DVPP API instance created by calling **CreateDvppApi** and closes the DVPP executor.	
	DvppGetOutParameter	Obtains the output buffer size of the JPEGD/JPEGE/PNGD module.	
Implementing the VDEC function	**CreateVdecApi**	Obtains a VDEC API instance, which is equivalent to the handle to the VDEC executor. The caller can use the obtained VDEC API instance to call **CreateVdecApi** for video decoding. Cross-function calling and cross-thread calling are supported.	idvppapi.h

Category	API	Function	Header File
	VdecCtl	Controls the DVPP executor to implement video decoding. The **VdecCtl** API is called by using the instance created by **CreateVdecApi**.	
	DestroyVdecApi	Releases the VDEC API instance created by **CreateVdecApi** and closes the VDEC executor.	
Implementing the VENC function	CreateVenc	Obtains the VENC instance, which is equivalent to obtaining the handle of the VENC executor. The caller can call **RunVenc** to encode images by using the obtained VENC instance.	Venc.h
	RunVenc	Controls the DVPP executor to implement video encoding. The **RunVenc** API is called by using the instance created by **CreateVenc**.	
	DestroyVenc	Releases the VENC instance created by calling **CreateVenc** and closes the VENC executor.	

1.3 VPC Function

Function Description

The VPC module provides the following functions:

- **Cropping**: Crops a required area out of the input image.
- **Resizing**
  - In the VPC, image resizing can be classified into 8K image resizing and non-8K image resizing modes based on resolution:

        - 8K image resizing: used to process the input image whose width or height is within the range of 4096–8192 pixels
        - Non-8K image resizing: used to process the input image whose resolution is within the range of 32 x 6 pixels to 4096 x 4096 pixels
      - Single-image cropping/resizing (supporting uncompressed format and HFBC compressed format) and single-image multi-ROI cropping/resizing (supporting uncompressed format and HFBC compressed format):

        HFBC is a compressed image format for the VDEC output. In this format, the VDEC has better processing performance.
      - Other resizing modes: for example, original image resizing and proportional image resizing.
- **Overlaying**: Crops an image out of an input image, resizes the cropped image, and places it in a specified area of the output image. The output image may be a blank image (when the output buffer allocated by the user is empty) or an existing image (when an image has been read into the output buffer allocated by the user). Note that the overlaying concept here refers only to the case when the output image is an existing image.
- **Stitching**: Crops multiple images out of an input image, resizes the cropped images, and place them in a specified area of the output image.
- **Format conversion**
  - Converts an RGB, YUV422, or YUV444 image to a YUV420 image.
  - Converts a color image to a grayscale image. For the output image, only the data of the Y component is used.

Restriction

- The following table describes the 8K image resizing and non-8K image resizing modes in the VPC based on resolution.

Input Image Resolution	Input Image Format	Width Stride x Height Stride Alignment for the Input Image	VPC Function	Output Image Resolution	Output Image Format	Width Stride x Height Stride Alignment for the Output Image
Width or height within the range of 4096–8192 pixels (excluding 4096)	YUV420 SP (NV12 and NV21)	2 x 2 alignment	8K image resizing	16 x 16 to 4096 x 4096	For details, see the **outputFormat** parameter in **Table 2-1**.	2 x 2 alignment

Input Image Resolution	Input Image Format	Width Stride x Height Stride Alignment for the Input Image	VPC Function	Output Image Resolution	Output Image Format	Width Stride x Height Stride Alignment for the Output Image
32 x 6 to 4096 x 4096 pixels (including 4096)	For details, see the **inputFormat** parameter in Table 2-1.	• For details about the width stride alignment, see the **widthStride** parameter in Table 2-1. • The height stride is 2-pixel aligned.	Non-8K image resizing	32 x 6 to 4096 x 4096	For details, see the **outputFormat** parameter in Table 2-1.	16 x 2 alignment

- Resizing ratio of the width or height: [1/32, 16]
- 8K image resizing: Resizing is supported, the format conversion between YUV420SP NV12 and YUV420SP NV21 is supported, but cropping is not supported.
- Buffer restrictions:
  - The start address of the buffer must be 16-byte aligned.
  - In the same task, the virtual addresses of the input and output buffers must be in the same 4 GB space.

VPC Function Diagram

Figure 1-2 VPC function diagram (cropping+resizing+overlaying)

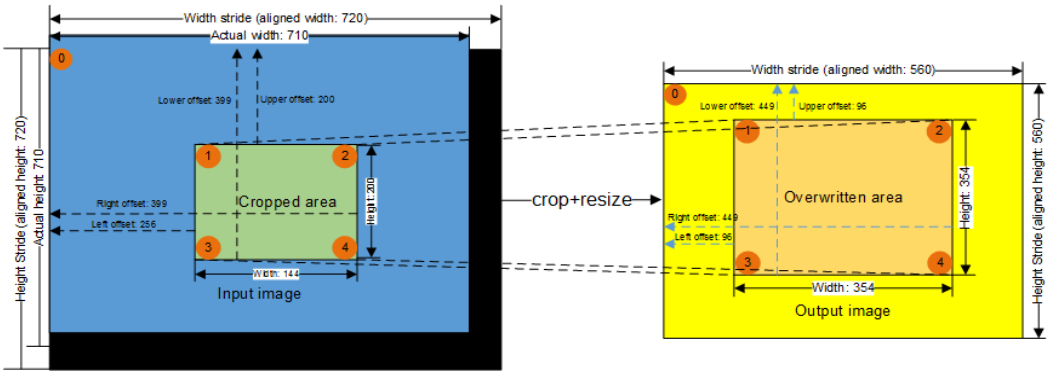


Figure 1-3 VPC function diagram (stitching)

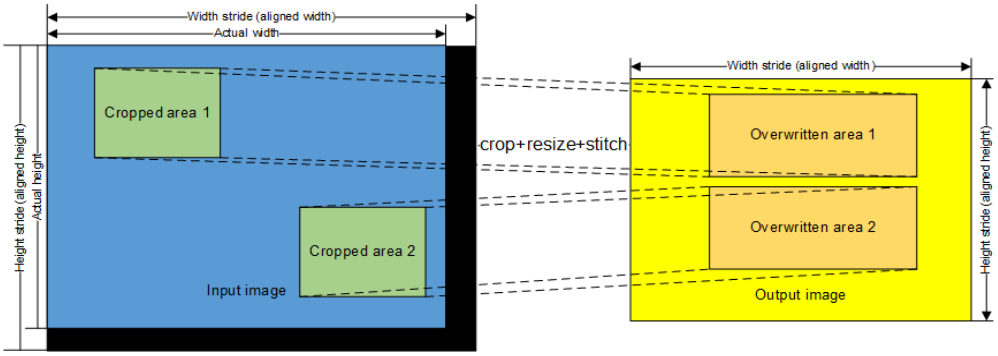


Table 1-2 Concepts

Concept	Description
Width stride	Step for a row of pixels, indicating the width of the input image after alignment. The width stride calculation of an RGB image is different from that of a YUV image. The minimum width stride is 32, and the maximum width stride is 4096 x 4 (the image width is 4096 and the image format is ARGB). • YUV400SP, YUV420SP, YUV422SP, and YUV444SP: Align the width of the input image to a 16-pixel boundary. • YUV422 packed: Multiply the width of the input image by 2 and align the result to a 16-pixel boundary. • YUV444 packed and RGB888: Multiply the width of the input image by 3 and align the result to a 16-pixel boundary. • XRGB8888: Multiply the width of the input image by 4 and align the result to a 16-pixel boundary. • HFBC format: The wide stride equals the width of the input image.

Concept	Description
Height stride	Number of lines in the buffer of an image, indicating the height of the input image after alignment. Value: The height of the input image must be 2-pixel aligned. The minimum height stride is 6, and the maximum height stride is 4096.
Top/ Bottom/ Left/ Right offset	You can configure the top offset, bottom offset, left offset, and right offset to implement the following functions: (1) Specify the position of the cropped area or overwritten area. (2) Control the width and height of the cropped area or overwritten area by using the following formulas: Right offset – Left offset + 1 = Width; Bottom offset – Top offset + 1 = Height. • Left offset: horizontal offset of points 1 and 3 in the cropped/ overwritten area relative to point 0 in the input/output image • Right offset: horizontal offset of points 2 and 4 in the cropped/ overwritten area relative to point 0 in the input/output image • Top offset: vertical offset of points 1 and 2 in the cropped/ overwritten area relative to point 0 in the input/output image • Bottom offset: vertical offset of points 3 and 4 in the cropped/ overwritten area relative to point 0 in the input/output image
Cropped area	Image area to be cropped. The minimum resolution is 10 x 6, and the maximum resolution is 4096 x 4096.
Overwritten area	Area specified by the user in the output image. The minimum resolution is 10 x 6, and the maximum resolution is 4096 x 4096. Restrictions: • For the overwritten area, the left offset and the top offset must be even numbers, and the right offset and the bottom offset must be odd numbers. • The cropped area cannot be larger than the input image, and the overwritten area cannot be larger than the output image. • The overwritten area can be directly mapped on the leftmost side of the output image, that is, the left offset of the output image is **0**. • The maximum number of overwritten areas is 256. • The left offset of the overwritten area relative to the output image is 16-pixel aligned. • It is recommended that the width of the output overwritten area be 16-pixel aligned. If the width is not 16-pixel aligned, invalid extra data is written to make the width 16-pixel aligned.

Performance Specifications

- For **non-8K image resizing**, the VPC performance involves different scenarios such as image cropping and resizing. When the resolution changes during image processing, the performance is calculated based on the larger

resolution. For example, if the resolution of the image after resizing is greater than that before resizing, the former is used to calculate the performance specifications. If the resolution of the overwritten area is greater than that of the cropped area, the former is used to calculate the performance specifications. For YUV420 SP images, the performance specifications in typical scenarios are as follows:

Scenario	Total Frame Rate
1080p x *n* channels (*n* < 4, one channel corresponds to one thread)	*n* x 360 fps
1080p x *n* channels (*n* ≥ 4, one channel corresponds to one thread)	1440 fps
4K x *n* channels (*n* < 4, one channel corresponds to one thread)	*n* x 90 fps
4K x *n* channels (*n* ≥ 4, one channel corresponds to one thread)	360 fps

- For **8K image resizing**, the VPC performance is closely related to the output resolution. A higher output resolution indicates longer processing time and lower performance. The following table lists the performance specifications in typical scenarios (output resolution of 1080p or 4K and image format of YUV420 SP).

Scenario	Total Frame Rate
1080p x *n* channels (*n* = 1, one channel corresponds to one thread)	4 fps
1080p x *n* channels (*n* ≥ 4, one channel corresponds to one thread)	16 fps
4K x *n* channels (*n* = 1, one channel corresponds to one thread)	1 fps
4K x *n* channels (*n* ≥ 4, one channel corresponds to one thread)	4 fps

Reference

The following figure shows the component layout of the RBG and YUV images. Two YUV420SP images are used as examples for SP format images and an ARGB image is used as an example for packed and RGB images.

Format	Resolution	Width Stride	Height Stride	Buffer Size					
YUV420SP	4 x 4	4	4	24					
Buffer Layout									
y11	y12	y13	y14						
y21	y22	y23	y24						
y31	y32	y33	y34						
y41	y42	y43	y44						
u11	v11	u13	v13						
u31	v31	u33	v33						
Format	Resolution	Width Stride	Height Stride	Buffer Size					
YUV420SP	4 x 4	6	6	54					
Buffer Layout	The letter x indicates invalid data.								
y11	y12	y13	y14	x	x				
y21	y22	y23	y24	x	x				
y31	y32	y33	y34	x	x				
y41	y42	y43	y44	x	x				
x	x	x	x	x	x				
x	x	x	x	x	x				
u11	v11	u13	v13	x	x				
u31	v31	u33	v33	x	x				
x	x	x	x	x	x				
Format	Resolution	Width Stride	Height Stride	Buffer Size					
ARGB	2 x 2	10	4	40					
Buffer Layout	The letter x indicates invalid data.								
a11	r11	g11	b11	a12	r12	g12	b12	x	x
a21	b21	g21	b21	a22	r22	g22	b22	x	x
x	x	x	x	x	x	x	x	x	x
x	x	x	x	x	x	x	x	x	x

1.4 JPEGE Function

Function and Restriction

The JPEGE module is used to encode JPG images.

- The JPEGE supports the following input formats:
  - YUV422 packed (YUYV, YVYU, UYVY, and VYUY)
  - YUV420 semi-planar (NV12 and NV21)
- The JPEGE supports the following resolution range:

  Maximum resolution: 8192 x 8192; minimum resolution: 32 x 32
- The JPEGE output format is JPEG. Only Huffman encoding is supported. Arithmetic encoding and progressive encoding are not supported.
- Buffer restrictions:
  - The start address of the buffer must be 128-byte aligned.
  - In the same task, the virtual addresses of the input and output buffers must respectively be in the same 4 GB space.

Performance Specifications

Scenario	Total Frame Rate
1080p x *n* channels ($n \geq 1$)	64 fps
4K x *n* channels ($n \geq 1$)	16 fps

1.5 JPEGD Function

Function Description

The JPEGD module is used to decode .jpg, .jpeg, .JPG, and .JPEG images. For the image formats not supported by the hardware, software decoding is used.

After decoding, the images are output in the following formats:

JPEG(444) -> YUV444 / YUV420 semi-planar with the V component before the U component

JPEG(422) -> YUV422 / YUV420 semi-planar with the V component before the U component

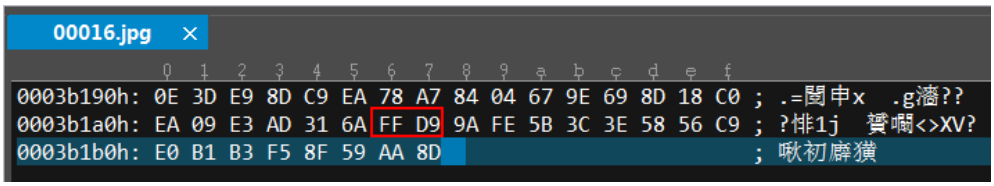
JPEG(420) -> YUV420 / YUV420 semi-planar with the V component before the U component

JPEG(400) -> YUV420/ UV data is padded by 0x80.

NOTICE

- The JPEGD uses the camel-case naming style due to the requirements of the programming standards. However, the original kernel-style input parameters and output parameters are also supported. Therefore, the JPEGD supports two sets of parameters. The camel-case naming style is recommended.
- If user-defined data exists after the end of image (EOI) flag **0XFFD9**, the JPEGD clears the 8-byte data after the EOI during decoding. If you want to retain the user-defined data, read it to the buffer and back it up before sending it to the JPEGD.

  To check whether user-defined data exists after the EOI flag in an image, use the binary viewing tool to open the image. For example, user-defined data exists after the EOI flag **FFD9** in the following figure.



Restriction

- Restrictions on the input image:
  - Maximum resolution: 8192 x 8192; minimum resolution: 32 x 32
  - Only Huffman encoding is supported. The color space of the stream is YUV, and the subsample of the stream is 444, 422, 420, or 400.
  - Arithmetic coding is not supported.
  - The progressive JPEG format is not supported.
  - The JPEG2000 format is not supported.

- Hardware restrictions:
  - Only a maximum of four Huffman tables are supported, including two direct coefficient (DC) tables and two alternating coefficient (AC) tables.
  - Only a maximum of three quantization tables are supported.
  - Only 8-bit sampling precision is supported.
  - Only images after sequential encoding can be decoded.
  - Only JPEG decoding based on discrete cosine transform (DCT) is supported.
  - Only one start of scan (SOS) flag is supported for image decoding.
- Software restrictions:
  - A maximum of three SOS flags are supported for image decoding.
  - Abnormal image decoding with insufficient minimum coded unit (MCU) data is supported.
- Buffer restrictions:
  - The start address of the buffer must be 128-byte aligned.
  - In the same task, the virtual addresses of the input and output buffers must respectively be in the same 4 GB space.

Performance Specifications

The performance specifications of the JPEGD are based on the hardware decoding performance. The JPEGD hardware decoding does not support the image decoding with three SOS flags. For the image formats that are not supported by the hardware, software decoding is used. The software decoding performance for reference is 1080p x 1 channel at 15 fps.

Scenario	Total Frame Rate
1080p x 1 channel	128 fps
1080p x *n* channels ($n \geq 2$)	256 fps
4K x 1 channel	32 fps
4K x *n* channels ($n \geq 2$)	64 fps

1.6 PNGD Function

Function and Restriction

The PNGD module is used to implement hardware decoding for PNG images.

- The PNGD supports the following input formats:
  RGBA and RGB
- The PNGD supports the following output formats:
  RGBA and RGB

- The PNGD supports the following resolution range:
  Maximum resolution: 4096 x 4096; minimum resolution: 32 x 32

Performance Specifications

Scenario	Total Frame Rate
1080p x 1 channel	4 fps
1080p x 2 channels	8 fps
1080p x 3 channels	12 fps
1080p x 4 channels	16 fps
1080p x 5 channels	20 fps
1080p x *n* channels ($n \geq 6$)	24 fps
4K x 1 channel	1 fps
4K x 2 channels	2 fps
4K x 3 channels	3 fps
4K x 4 channels	4 fps
4K x 5 channels	5 fps
4K x *n* channels ($n \geq 6$)	6 fps

1.7 VDEC Function

Function and Restriction

The VDEC module is used to decode videos.

- The VDEC supports the following input formats:
  H.264 BP/MP/HP Level 5.1 YUV420SP encoded streams
  H.265 8-Bit/10-Bit Level 5.1 YUV420SP encoded streams
- The VDEC supports the following resolution range:
  Maximum resolution: 4096 x 4096; minimum resolution: 128 x 128
- The VDEC output is HFBC compressed YUV420SP data.
  HFBC is a compressed image format for the VDEC output. In this format, the VDEC has better processing performance.
- Bad frames or frame loss in the streams may cause VDEC frame loss.
- The VDEC cannot decode the streams encoded in interlaced scanning mode.
- The VDEC module does not use the singleton pattern. Therefore, the C APIs provided by the VDEC module are different for those provided by other modules.

Performance Specifications

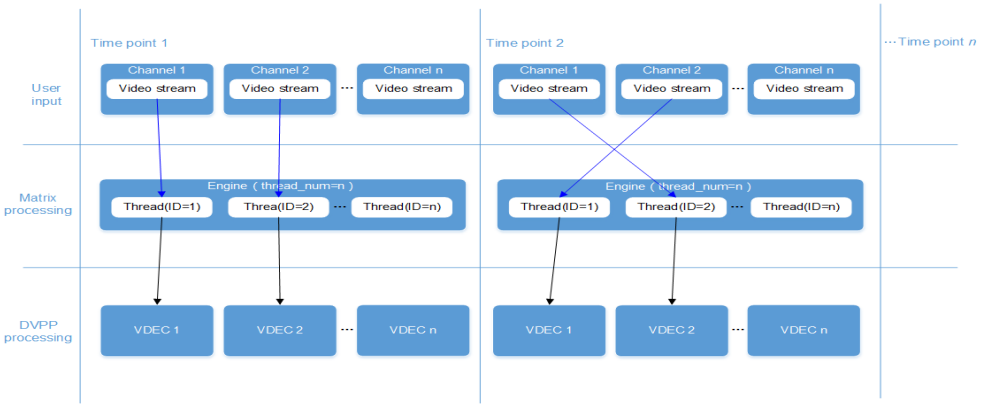
Scenario	Total Frame Rate
1080p x *n* channels (2 ≤ *n* ≤ 16)	480 fps
1080p x 1 channel	240 fps
4K x *n* channels (2 ≤ *n* ≤ 16)	120 fps
4K x 1 channel	60 fps

If Matrix is used to orchestrate the application process, when DVPP is used to decode multiple channels of input video streams, because data of each frame in the video streams is associated, each VDEC is required to fixedly correspond to one video stream to ensure the data sequence of each frame in a same video stream. Otherwise, the VDEC in DVPP cannot decode video streams.

- When multiple video streams are decoded, multiple implementation modes may be available to ensure the data sequence of each frame in the video streams. The following configurations are recommended:
  - In the graph configuration file, configure multiple VDEC engines in the graphs segment. One video stream corresponds to one engine.
  - In the graph configuration file, set **thread_num** to **1** in the VDEC engine segment. One engine corresponds to one thread.

  NOTE
  - For details about the functions of Matrix and the configuration of the graph configuration file, see *Matrix API Reference*.
  - One graph (one thread) can correspond to a maximum of 16 video streams.
- If only one VDEC engine is configured in the graph configuration file and **thread_num** is set to **n** (the value of **n** is the number of video stream channels) during multi-channel video stream decoding, video streams in Matrix may be processed in different threads at different time points. However, in DVPP, one VDEC requires one fixed channel of video stream data. One channel of video stream data corresponds to one thread. In this case, the data sequence of each frame in the video streams may not be ensured during video decoding, as shown in the figure.

Figure 1-4 VDEC process



1.8 VENC Function

The VENC module implements encoding in three steps: instantiation, encoding, and resource release.

Function and Restriction

The VENC module is used to encode YUV420 and YVU420 image data.

- The VENC supports the following input formats:
  YUV420 semi-planar NV12/NV21-8bit
- The VENC supports the following resolution range:
  Maximum resolution: 1920 x 1920; minimum resolution: 128 x 128
- The VENC supports the following output formats:
  H.264 BP/MP/HP
  H.265 MP (Only slice streams are supported.)

Performance Specifications

Scenario	Total Frame Rate
1080p x *n* channels (one process corresponds to one channel)	30 fps

Note: The performance at other resolutions can be estimated in the same way.

1.9 Input and Output Memory

For details about the **HIAI_DMalloc**/**HIAI_DFree** and **HIAI_DVPP_DMalloc**/**HIAI_DVPP_DFree** APIs as well as the graph configuration file (**receive_memory_without_dvpp** parameter), see *Matrix API Reference*.

Table 1-3 Memory requirements

Module	Input Memory	Output Memory
VPC	Use the APIs provided by Matrix to allocate or free memory. • Use **HIAI_DVPP_DMalloc** to allocate memory, which must meet the DVPP requirement (the start address of the memory must be 16-byte aligned). • Use **HIAI_DVPP_DFree** to free memory.	Use the APIs provided by Matrix to allocate or free memory. • Use **HIAI_DVPP_DMalloc** to allocate memory, which must meet the DVPP requirement (the start address of the memory must be 16-byte aligned). • Use **HIAI_DVPP_DFree** to free memory.
JPEGE and JPEGD	Use the APIs provided by Matrix to allocate or free memory. • Use **HIAI_DVPP_DMalloc** to allocate memory, which must meet the DVPP requirement (the start address of the memory must be 128-byte aligned). • Use **HIAI_DVPP_DFree** to free memory.	• If the output memory is specified by the user, it should also be freed by the user. Use the APIs provided by Matrix to allocate or free memory. – Use **HIAI_DVPP_DMalloc** to allocate memory, which must meet the DVPP requirement (the start address of the memory must be 128-byte aligned). Before memory allocation, call **DvppGetOutParameter** to obtain the output memory size. – Use **HIAI_DVPP_DFree** to free memory. • If the output memory is not specified by the user, DVPP allocates the memory internally. The **cbFree()** callback function in the JPEGE or JPEGD output parameter structure should be called to free the memory and set the memory address pointer to null.

Module	Input Memory	Output Memory
PNGD	Use the APIs provided by Matrix to allocate or free memory. • Use **HIAI_DVPP_DMalloc** to allocate memory, which must meet the DVPP requirement (the start address of the memory must be 128-byte aligned). • Use **HIAI_DVPP_DFree** to free memory.	If the output memory is specified by the user, it should also be freed by the user. Use the APIs provided by Matrix to allocate or free memory. • Use **HIAI_DVPP_DMalloc** to allocate memory, which must meet the DVPP requirement (the start address of the memory must be 128-byte aligned). Before memory allocation, call **DvppGetOutParameter** to obtain the output memory size. • Use **HIAI_DVPP_DFree** to free memory.
VDEC and VENC	There is no requirement on the memory. Native APIs such as **malloc**/**free** and **new**/**delete** can be called to allocate or free the memory. The **HIAI_DMalloc**/ **HIAI_DFree** and **HIAI_DVPP_DMalloc**/ **HIAI_DVPP_DFree** APIs provided by Matrix can also be called to allocate or free the memory. The **receive_memory_without_dvpp** parameter provided by Matrix can be used to control whether the memory is used by DVPP.	The HFBC data output by the VDEC is directly used as the input of the VPC. The VENC output memory is internally managed by the DVPP. You can copy the data in the output memory when using the VENC.

2 VPC/JPEGE/JPEGD/PNGD Function Interfaces

2.1 CreateDvppApi

Syntax	**int CreateDvppApi(IDVPPAPI*& pIDVPPAPI)**
Function	Creates a DVPP API instance, which is equivalent to creating the handle to the DVPP executor. The caller can use the applied DVPP API instance to call **DvppCtl** to process an image, either across functions or across threads.
Input	**IDVPPAPI** pointer reference. The input pointer must be **NULL**.
Output	**IDVPPAPI** pointer reference. If a DVPP API instance fails to be obtained, the output pointer is **NULL**. If a DVPP API instance is successfully obtained, the output pointer is not **NULL**.
Return Value	• **0**: success • **–1**: failure
Instructions	The caller creates the **IDVPPAPI** object pointer, which is initialized to **NULL**. The **IDVPPAPI** object pointer is passed by calling the **CreateDvppApi** function. If the application is successful, the **CreateDvppApi** function returns the DVPP API instance. Otherwise, **NULL** is returned. The caller needs to verify the return value.

Restriction	The caller is responsible for the life cycle of the DVPP API instance, including the application and release, which are implemented by using **CreateDvppApi** and **DestroyDvppApi**, respectively.

Calling Example

```
IDVPPAPI *pidvppapi = NULL;
CreateDvppApi(pidvppapi);
```

2.2 DvppCtl

2.2.1 API Description

Syntax	**int DvppCtl(IDVPPAPI*& pIDVPPAPI, int CMD, dvppapi_ctl_msg* MSG)**
Function	Controls the execution of DVPP modules, such as the VPC, JPEGE, JPEGD and PNGD. The **DvppCtl** API is called by using the instance created by **CreateDvppApi**.
Input	• **IDVPPAPI** pointer reference • **Command word (CMD)** • DVPP executor configuration information (**dvppapi_ctl_msg** type). Different DVPP modules have different configuration information. Therefore, the corresponding configuration information needs to be passed when the functions of each module are invoked.
Output	The output parameters vary according to the functions of the DVPP submodules. For details, see **dvppapi_ctl_msg**.
Return Value	• **0**: success • **–1**: failure
Instructions	The caller calls the **DvppCtl** function to pass the **IDVPPAPI** object pointer, the correct **command word (CMD)**, and the configured **dvppapi_ctl_msg**.
Restriction	None

Command Words (CMD)

The CMDs are defined in the **ddk/include/inc/dvpp/DvppCommon.h** file in the DDK installation directory.

Member Variable	Description	Value Range
DVPP_CTL_VPC_PROC	VPC function command word	0
DVPP_CTL_PNGD_PROC	PNGD function command word	1
DVPP_CTL_JPEGE_PROC	JPEGE function command word	2
DVPP_CTL_JPEGD_PROC	JPEGD function command word	3
DVPP_CTL_VENC_PROC	Reserved	5
DVPP_CTL_DVPP_CAPABILITY	Command word for querying the DVPP capability	6
DVPP_CTL_CMDLIST_PROC	Reserved	7
DVPP_CTL_TOOL_CASE_GET_RESIZE_PARAM	Reserved	8

dvppapi_ctl_msg

This type is defined in the **ddk/include/inc/dvpp/DvppCommon.h** file in the DDK installation directory.

Member Variable	Description	Value Range
int32_t in_size	Input parameter size	-
int32_t out_size	Output parameter size	-

Member Variable	Description	Value Range
void *in	Input parameter	• Input parameter of the VPC function: **VpcUserImageConfigure** • Input parameter of the JPEGE function: **sJpegeIn** • Input parameter of the JPEGD function: **JpegdIn** (camel-case-style JPEGD input structure, which is recommended) • Input parameter of the JPEGD function: **jpegd_raw_data_info** (kernel-style JPEGD input structure, which is not recommended) • Input parameter of the PNGD function: **PngInputInfoAPI** • Input parameter for querying the DVPP engine: **device_query_req_stru**
void *out	Output parameter	• Output parameter of the VPC function: none • Output parameter of the JPEGE function: **sJpegeOut** • Output parameter of the JPEGD function: **JpegdOut** (camel-case-style JPEGD output structure, which is recommended) • Output parameter of the JPEGD function: **jpegd_yuv_data_info** (kernel-style JPEGD output structure, which is not recommended) • Output parameter of the PNGD function: **PngOutputInfoAPI** • Output parameter for querying the DVPP engine: **dvpp_engine_capability_stru**

2.2.2 VPC Parameters

Input Parameter: VpcUserImageConfigure

Table 2-1 Input parameter VpcUserImageConfigure

Member Variable	Description	Value Range
uint8_t* bareDataAddr	Address of an uncompressed input image. If the input image is compressed, this parameter does not need to be set and the default value is **null**. You are advised to allocate the buffer by calling **HIAI_DVPP_DMalloc** provided by Matrix. The allocated buffer must meet the DVPP requirements, that is, the buffer addresses of the input and output images must be in the same 4 GB space, and the start addresses must be 16-pixel aligned. For details about **HIAI_DVPP_DMalloc**, see *Matrix API Reference*.	-
uint32_t bareDataBufferSize	Buffer size of an uncompressed input image. The buffer size is calculated based on the width stride and height stride of the image. The buffer size pointed by **bareDataAddr** must be the same as the value of **bareDataBufferSize**. When the input image is not compressed, this parameter does not need to be set and the default value is **0**.	For the uncompressed format, the buffer size is calculated as follows based on the image format: • YUV400SP and YUV420SP: widthStride x heightStride x 3/2 • YUV422SP: widthStride x heightStride x 2 • YUV444SP: widthStride x heightStride x 3 • YUV422 packed: widthStride x heightStride • YUV444 packed and RGB888: widthStride x heightStride • XRGB8888: widthStride x heightStride

Member Variable	Description	Value Range
uint32_t widthStride	Width stride of the image	The minimum width stride is 32, and the maximum width stride is 4096 x 4 (the image width is 4096 and the image format is ARGB). For 8K image resizing, the width stride must be 2-pixel aligned. For non-8K image resizing, the formula for calculating **widthStride** varies according to the image format. • YUV400SP, YUV420SP, YUV422SP, and YUV444SP: Align the width of the input image to a 16-pixel boundary. • YUV422 packed: Align the width of the input image to a 16-pixel boundary and multiply the result by 2 (the width is 16-pixel aligned and each pixel occupies 2 bytes). • YUV444 packed and RGB888: Align the width of the input image to a 16-pixel boundary and multiply the result by 3 (the width is 16-pixel aligned and each pixel occupies 3 bytes). • XRGB8888: Align the width of the input image to a 16-pixel boundary and multiply the result by 4 (the width is 16-pixel aligned and each pixel occupies 4 bytes). • HFBC format: The wide stride equals the width of the input image.
uint32_t heightStride	Height stride. You can calculate the UV data start addresses of YUV SP images based on this parameter.	Value: The height of the input image must be 2-pixel aligned. The minimum height stride is 6, and the maximum height stride is 4096.

Member Variable	Description	Value Range
enum VpcInputFormat inputFormat	Input image format	enum VpcInputFormat { INPUT_YUV400, // 0 INPUT_YUV420_SEMI_PLANNER_UV, // 1 INPUT_YUV420_SEMI_PLANNER_VU, // 2 INPUT_YUV422_SEMI_PLANNER_UV, // 3 INPUT_YUV422_SEMI_PLANNER_VU, // 4 INPUT_YUV444_SEMI_PLANNER_UV, // 5 INPUT_YUV444_SEMI_PLANNER_VU, // 6 INPUT_YUV422_PACKED_YUYV, // 7 INPUT_YUV422_PACKED_UYVY, // 8 INPUT_YUV422_PACKED_YVYU, // 9 INPUT_YUV422_PACKED_VYUY, // 10 INPUT_YUV444_PACKED_YUV, // 11 INPUT_RGB, // 12, input image format: RGB888 INPUT_BGR, // 13, input image format: BGR888 INPUT_ARGB, // 14. The component sequence of an input image in this format during storage is similar to RGB888. **A** indicates transparency. INPUT_ABGR, // 15. The component sequence of an input image in this format during storage is similar to BGR888. **A** indicates transparency. INPUT_RGBA, // 16. The component sequence of an input image in this format during storage is similar to

Member Variable	Description	Value Range
		RGB888. **A** indicates transparency. INPUT_BGRA, // 17. The component sequence of an input image in this format during storage is similar to BGR888. **A** indicates transparency. INPUT_YUV420_SEMI_PLANNER_UV_10BIT, // 18 INPUT_YUV420_SEMI_PLANNER_VU_10BIT, // 19 };

enum VpcOutputFormat outputFormat	Output image format	enum VpcOutputFormat {
		OUTPUT_YUV420SP_UV,
		OUTPUT_YUV420SP_VU
		};

VpcUserRoiConfigure* roiConfigure	Cropped area configuration. For details, see **VpcUserRoiConfigure structure**.	-
bool isCompressData	Whether the HFBC compressed image format of the VDEC output is used	When the image comes from the VDEC, set this parameter to **true**. For other image formats, set this parameter to **false**. The default value is **false**. The HFBC compression format supports single-image cropping/resizing and single-image multi-ROI cropping/resizing.
VpcCompressDataConfigure compressDataConfigure	Data configuration of the HFBC compressed image output by the VDEC. For details, see **VpcCompressDataConfigure structure**.	-

Member Variable	Description	Value Range
bool yuvSumEnable	Whether to calculate the **yuvSum** value. Calculation of the **yuvSum** value supports only the single-image single-ROI mode.	When the **yuvSum** value needs to be calculated, set this parameter to **true**. Otherwise, set this parameter to **false**. The default value is **false**. The restrictions for calculating the **yuvSum** value are as follows: 1. The input image resolution is less than or equal to 4096 x 4096. 2. The start point of the overwritten area is (0, 0). 3. The input is a single image with a single ROI.
VpcUserYuvSum yuvSum	**yuvSum** calculation configuration. For details, see **VpcUserYuvSum structure**.	-
VpcUserPerformanceTuningParameter tuningParameter	Reserved. This parameter is used for parameter tuning. For details, see **VpcUserPerformanceTu-nin...**.	-
uint32_t* cmdListBufferAddr	Reserved	-
uint32_t cmdListBufferSize	Reserved	-
uint64_t yuvScalerParaSetAddr	Reserved. This parameter indicates the file path and file name array of the filtering parameter set. **NOTE** • The parameter set must be on the device. • Ensure that the input file path is correct.	A parameter set file path consists of an absolute file path on the device side and the file name, for example, **{"/root/share/vpc/YUVScaler_pra.h"}**.

Member Variable	Description	Value Range
uint16_t yuvScalerParaSetSize	Reserved. This parameter indicates the number of elements in the parameter set array to which the parameter set address points. The default value is **1**.	1 ≤ yuvscaler_paraset_size ≤ 10
uint16_t yuvScalerParaSetIndex	Reserved. This parameter indicates the index of **yuvScalerParaSetAddr** corresponding to **yuvScalerParaSetIndex** (The default value is **0**.)	0≤ yuvScalerParaSetIndex < 10
uint8_t isUseMultiCoreAccelerate	Reserved	-
uint8_t reserve1	Reserved	-
uint16_t reserve2;	Reserved	-

Output Parameters

None

2.2.3 JPEGE Parameters

Input Parameter: sJpegeIn

Table 2-2 Input parameter sJpegeIn

Member Variable	Description
eEncodeFormat format	Format of the YUV input data. YUV422 packed (YUYV, YVYU, UYVY, and VYUY) and YUV420 semi-planar (NV12 and NV21) are supported. typedef enum { JPGENC_FORMAT_UYVY = 0x0, JPGENC_FORMAT_VYUY = 0x1, JPGENC_FORMAT_YVYU = 0x2, JPGENC_FORMAT_YUYV = 0x3, JPGENC_FORMAT_NV12 = 0x10, JPGENC_FORMAT_NV21 = 0x11, } eEncodeFormat;
unsigned char* buf	The YUV input data needs to be allocated by the caller and must be properly aligned. For details, see the information about the **stride** and **heightAligned** parameters. You are advised to allocate the buffer by calling **HIAI_DVPP_DMalloc** provided by Matrix. The allocated buffer must meet the DVPP requirements, that is, the buffer addresses of the input and output images must be in the same 4 GB space, and the start addresses must be 128-pixel aligned. For details about **HIAI_DVPP_DMalloc**, see *Matrix API Reference*. **NOTICE** When **HIAI_DVPP_DMalloc** is used to allocate a buffer, ensure that the size of the allocated buffer is the same as the value of the input parameter **bufSize**.
uint32_t bufSize	Length of the input buffer, that is, the length of the data after the width and height are aligned
uint32_t width	Width of the input image. The value range is [32, 8192].

Member Variable	Description
uint32_t height	Height of the input image. The value range is [32, 8192].
uint32_t stride	Width of the input image after alignment. The width alignment boundary can be 16 pixels or a multiple of 16 pixels, for example, 128 pixels. For YUV422 packed data, the stride must be the result after the width is multiplied by 2 and then aligned to a 16-pixel boundary.
uint32_t heightAligned	1. The value can be the same as the height. 2. The value can be the aligned image height after Matrix cross-side transmission. 3. The value supports 16-pixel alignment.
uint32_t level	Encoding quality in a range of [0, 100]. The encoding quality of level 0 is similar to that of level 100. For the value range of [1, 100], a smaller value indicates poorer output image quality.

Output Parameter: sJpegeOut

Table 2-3 Output parameter sJpegeOut

Member Variable	Description
unsigned char* jpgData	Start address of the JPG data in the output buffer You are advised to allocate the buffer by calling **HIAI_DVPP_DMalloc** provided by Matrix. The allocated buffer must meet the DVPP requirements, that is, the buffer addresses of the input and output images must be in the same 4 GB space, and the start addresses must be 128-pixel aligned. For details about **HIAI_DVPP_DMalloc**, see *Matrix API Reference*. **NOTICE** When **HIAI_DVPP_DMalloc** is used to allocate a buffer, ensure that the size of the allocated buffer is the same as the value of the input parameter **jpgSize**.
uint32_t jpgSize	Data length of the encoded JPG image

Member Variable	Description
JpegCalBackFree cbFree	Callback function for freeing the output buffer • If the output buffer is specified by the user, it should also be freed by the user. • If the output buffer is not specified by the user, DVPP allocates the buffer internally. The **cbFree()** callback function should be called to free the buffer, and **jpgData** needs to be set to a null pointer. For details about the calling example, see **6.2 Implementing the JPEGE Function**.

2.2.4 JPEGD Parameters

Input Parameter: JpegdIn

Table 2-4 Input parameter JpegdIn

Member Variable	Description
unsigned char* jpegData	JPG input image data. The start address is 128-byte aligned and the buffer is allocated by using the 2 MB huge page table. In the same task, the virtual addresses of the input and output buffers must be in the same 4 GB space. You are advised to allocate the buffer by calling **HIAI_DVPP_DMalloc** provided by Matrix. The allocated buffer must meet the DVPP requirements, that is, the buffer addresses of the input and output images must be in the same 4 GB space, and the start addresses must be 128-pixel aligned. For details about **HIAI_DVPP_DMalloc**, see *Matrix API Reference*. **NOTICE** When **HIAI_DVPP_DMalloc** is used to allocate memory, ensure that the size of the allocated memory is the same as the value of the input parameter **jpegDataSize**.

Member Variable	Description
uint32_t jpegDataSize	Length of the input buffer. Use **HIAI_DVPP_DMalloc** provided by Matrix to allocate the buffer. The size of the allocated buffer is (Actual data size + 8 bytes). 8 bytes is the hardware restriction requirement.
bool isYUV420Need	Whether to output data in YUV420 semi-planar format. • **true**: Yes • **false**: No. The output is in original format. The JPEGD supports raw format output (including YUV420SP, YUV422SP, and YUV444SP) and down-sampling YUV420 semi-planar output. YUV420 grayscale image output is in fake420 format.
bool isVBeforeU	Reserved. This parameter must be set to **true**. The V component is before the U component.

Output Parameter: JpegdOut

Table 2-5 Output parameter JpegdOut

Member Variable	Description
unsigned char* yuvData	Buffer for the output YUV data. The width and height of the image are values after alignment based on 128 x 16. You are advised to allocate the buffer by calling **HIAI_DVPP_DMalloc** provided by Matrix. The allocated buffer must meet the DVPP requirements, that is, the buffer addresses of the input and output images must be in the same 4 GB space, and the start addresses must be 128-pixel aligned. For details about **HIAI_DVPP_DMalloc**, see *Matrix API Reference*. **NOTICE** When **HIAI_DVPP_DMalloc** is used to allocate a buffer, ensure that the size of the allocated buffer is the same as the value of the input parameter **yuvDataSize**.
uint32_t yuvDataSize	Length of the output YUV data. The data length is calculated based on the aligned width and height or can be obtained by calling **DvppGetOutParameter**.

Member Variable	Description
uint32_t imgWidth	Width of the output YUV image
uint32_t imgHeight	Height of the output YUV image
uint32_t imgWidthAligned	Width of the output image, which is 128-pixel aligned
uint32_t imgHeightAligned	Height of the output image, which is 16-pixel aligned
JpegCalBackFree cbFree	Callback function for freeing the output buffer • If the output buffer is specified by the user, it should also be freed by the user. • If the output buffer is not specified by the user, DVPP allocates the buffer internally. The **cbFree()** callback function should be called to free the buffer, and **yuvData** needs to be set to a null pointer. For details about the calling example, see **6.3 Implementing the JPEGD Function**.

jpegd_color_space outFormat	Format of the output YUV data.
	enum jpegd_color_space{
	DVPP_JPEG_DECODE_OUT_UNKNOWN = -1,
	DVPP_JPEG_DECODE_OUT_YUV444 = 0,
	DVPP_JPEG_DECODE_OUT_YUV422_H2V1 = 1,
	/* YUV422 */
	DVPP_JPEG_DECODE_OUT_YUV422_H1V2 = 2,
	/* YUV440 */
	DVPP_JPEG_DECODE_OUT_YUV420 = 3,
	DVPP_JPEG_DECODE_OUT_YUV400 = 4,
	DVPP_JPEG_DECODE_OUT_FORMAT_MAX,
	};

Input Parameter: jpegd_raw_data_info

Table 2-6 Input parameter jpegd_raw_data_info

Member Variable	Description
unsigned char* jpeg_data	Data of the input JPG image. The start address is 128-byte aligned, and the 2 MB huge page table is used for allocation. You are advised to allocate the buffer by calling **HIAI_DVPP_DMalloc** provided by Matrix. The allocated buffer must meet the DVPP requirements, that is, the buffer addresses of the input and output images must be in the same 4 GB space, and the start addresses must be 128-pixel aligned. For details about **HIAI_DVPP_DMalloc**, see *Matrix API Reference*. **NOTICE** When **HIAI_DVPP_DMalloc** is used to allocate a buffer, ensure that the size of the allocated buffer is the same as the value of the input parameter **jpeg_data_size**.
uint32_t jpeg_data_size	Length of the input buffer. Use **HIAI_DVPP_DMalloc** provided by Matrix to allocate the buffer. The size of the allocated buffer is (Actual data size + 8 bytes). 8 bytes is the hardware restriction requirement.

jpegd_raw_format in_format	YUV sampling format of the input image. Retain the default value.
	enum jpegd_raw_format{
	DVPP_JPEG_DECODE_RAW_YUV_UNSUPPORT = -1,
	DVPP_JPEG_DECODE_RAW_YUV444 = 0,
	DVPP_JPEG_DECODE_RAW_YUV422_H2V1 = 1, // 422
	// YUV440 is not supported anymore. This field is reserved.
	DVPP_JPEG_DECODE_RAW_YUV422_H1V2 = 2, // 440
	DVPP_JPEG_DECODE_RAW_YUV420 = 3,
	DVPP_JPEG_DECODE_RAW_MAX,
	};

Member Variable	Description
bool IsYUV420Need	Whether to output data in YUV420 semi-planar format. • **true**: Yes • **false**: No. The output is in original format. The JPEGD supports raw format output (including YUV420SP, YUV422SP, and YUV444SP) and down-sampling YUV420 semi-planar output. YUV420 grayscale image output is in fake420 format.
bool isVBeforeU	Reserved. This parameter must be set to **true**. The V component is before the U component.

Output Parameter: jpegd_yuv_data_info

Table 2-7 Output parameter jpegd_yuv_data_info

Member Variable	Description
unsigned char* yuv_data	Buffer for the output YUV data. The width and height of the image are aligned values. **NOTICE** The buffer is allocated and managed by DVPP and cannot be specified by users.
uint32_t yuv_data_size	Length of the output YUV data. The data length is calculated based on the aligned width and height.
uint32_t img_width	Width of the output YUV image
uint32_t img_height	Height of the output YUV image
uint32_t img_width_aligned	Width of the output image, which is 128-pixel aligned
uint32_t img_height_aligned	Height of the output image, which is 16-pixel aligned
JpegCalBackFree cbFree	Callback function for freeing the output buffer DVPP internally allocates buffers. You need to call the **cbFree()** callback function to free buffers and set **yuv_data** to a null pointer. For details about the calling example, see **6.3 Implementing the JPEGD Function**.

Member Variable	Description

enum jpegd_color_space out_format	Format of the output YUV data.
	enum jpegd_color_space {
	DVPP_JPEG_DECODE_OUT_UNKNOWN = -1,
	DVPP_JPEG_DECODE_OUT_YUV444 = 0,
	DVPP_JPEG_DECODE_OUT_YUV422_H2V1 = 1,// 422
	// YUV440 is not supported anymore. This field is reserved.
	DVPP_JPEG_DECODE_OUT_YUV422_H1V2 = 2,// 440
	DVPP_JPEG_DECODE_OUT_YUV420 = 3,
	DVPP_JPEG_DECODE_OUT_YUV400 = 4,
	DVPP_JPEG_DECODE_OUT_FORMAT_MAX,
	};

2.2.5 PNGD Parameters

Input Parameter: PngInputInfoAPI

Table 2-8 Input parameter PngInputInfoAPI

Member Variable	Description
void* inputData	Address of the input image data. You are advised to allocate the buffer by calling **HIAI_DVPP_DMalloc** provided by Matrix. The allocated buffer must meet the DVPP requirements, that is, the buffer addresses of the input and output images must be in the same 4 GB space, and the start addresses must be 128-pixel aligned. For details about **HIAI_DVPP_DMalloc**, see *Matrix API Reference*. **NOTICE** When **HIAI_DVPP_DMalloc** is used to allocate a buffer, ensure that the size of the allocated buffer is the same as the value of the input parameter **inputSize**.
uint64_t inputSize	Input buffer length, which is used to verify the input data

Member Variable	Description
void* address	Address of the input image data. The current version does not support this parameter.
uint64_t size	Length of the input buffer. The current version does not support this parameter.
int32_t transformFlag	Transform flag. The value **1** indicates that RGBA is converted into RGB, whereas the value **0** indicates that the original format is retained.

Output Parameter: PngOutputInfoAPI

Table 2-9 Output parameter PngOutputInfoAPI

Member Variable	Description
void* outputData	Address of the output image data. The current version does not support this parameter.
uint64_t outputSize	Address of the output image data. The current version does not support this parameter.
void* address	Address of the output buffer. You are advised to allocate the buffer by calling **HIAI_DVPP_DMalloc** provided by Matrix. The allocated buffer must meet the DVPP requirements, that is, the buffer addresses of the input and output images must be in the same 4 GB space, and the start addresses must be 128-pixel aligned. For details about **HIAI_DVPP_DMalloc**, see *Matrix API Reference*. **NOTICE** When **HIAI_DVPP_DMalloc** is used to allocate a buffer, ensure that the size of the allocated buffer is the same as the value of the input parameter **outputSize**.
uint64_t size	Buffer size
int32_t format	Output image format • **2**: RGB output • **6**: RGBA output

Member Variable	Description
uint32_t width	Output image width
uint32_t high	Output image height
uint32_t widthAlign	Width alignment. Align the size of the buffer occupied by a line of image data to a 128-byte boundary. If the output format is RGB, multiply the width by 3 and align the result to a 128-byte boundary. If the output format is RGBA, multiply the width by 4 and align the result to a 128-byte boundary.
uint32_t highAlign	Height alignment. Currently, 16-byte alignment is used.

2.2.6 Parameters for Querying the DVPP Engine

Function

The parameters are used to query the DVPP engine capabilities, including the capabilities, resolution restrictions, and performance parameters of all modules.

Input Parameter: device_query_req_stru

Table 2-10 Input parameter device_query_req_stru

Member Variable	Description	Value Range
uint32_t module_id	Module ID	The value is fixed at 1.
uint32_t engine_type	DVPP engine type	VDEC: 0 JPEGD: 1 PNGD: 2 JPEGE: 3 VPC: 4 VENC: 5

Output Parameter: dvpp_engine_capability_stru

Table 2-11 Output parameter dvpp_engine_capability_stru

Member Variable	Description	Value Range
int32_t engine_type	DVPP engine type	VDEC: 0 JPEGD: 1 PNGD: 2 JPEGE: 3 VPC: 4 VENC: 5
struct dvpp_resolution_stru max_resolution	Maximum resolution	For details, see **7.4 Structures in dvpp_engine_capability_stru**.
struct dvpp_resolution_stru min_resolution;	Minimum resolution	For details, see **7.4 Structures in dvpp_engine_capability_stru**.
uint32_t protocol_num;	Number of standard protocol types supported by the engine	VDEC: 5 JPEGD: 1 PNGD: 1 JPEGE: 1 VPC: 0 VENC: 4

Member Variable	Description	Value Range

uint32_t protocol_type[DVPP_PROTOCOL_TYPE_MAX];	Table of the standard protocol types supported by the engine	*enum* dvpp_proto_type {
		dvpp_proto_unsupport =-1,
		dvpp_itu_t81,
		iso_iec_15948_2003,
		h265_main_profile_level_5_1_hightier,
		h265_main_10_profile_level_5_1_hightier,
		h264_main_profile_level_5_1,
		h264_baseline_profile_level_5_1,
		h264_high_profile_level_5_1,
		h264_high_profile_level_4_1,
		h264_main_profile_level_4_1,
		h264_baseline_profile_level_4_1,
		h265_main_profile_level_4_1
		};

uint32_t input_format_num;	Number of supported input formats	VDEC: 2 JPEGD: 1 PNGD: 1 JPEGE: 6 VPC: 51 VENC: 2
struct dvpp_format_unit_stru engine_input_format_table[DVPP_VADIO_FORMAT_MAX];	Table of the input formats supported by the engine	For details, see **7.4 Structures in dvpp_engine_capability_stru**.
uint32_t output_format_num;	Number of supported output formats	VDEC: 4 JPEGD: 4 PNGD: 2 JPEGE: 1 VPC: 2 VENC: 2

Member Variable	Description	Value Range
struct dvpp_format_unit_stru engine_output_format_table[DVPP_VADIO_FORMAT_MAX];	Table of the output formats supported by the engine	For details, see **7.4 Structures in dvpp_engine_capability_stru**.
uint32_t performance_mode_num;	Number of performance modes	The value is fixed at 1.
struct dvpp_perfomance_unit_stru performance_mode_table[DVPP_PERFOMANCE_MODE_MAX];	Structure of the performance in the module	For details, see **7.4 Structures in dvpp_engine_capability_stru**.
struct dvpp_pre_contraction_stru pre_contraction;	Structure of pre-contraction information	For details, see **7.4 Structures in dvpp_engine_capability_stru**.
struct dvpp_pos_scale_stru pos_scale;	Structure of post-scaling information	For details, see **7.4 Structures in dvpp_engine_capability_stru**.
uint32_t spec_input_num	Number of types of the input formats. Currently, the following two types are supported: • HFBC • YUV or RGB	-
struct dvpp_vpc_data_spec_stru spec_input[DVPP_DATA_INPUT_SPEC_TYPE_MAX]	Description of the types of the input formats	For details, see **dvpp_vpc_data_spec_stru. ...**.

2.3 DestroyDvppApi

Syntax	**int DestroyDvppApi(IDVPPAPI*& pIDVPPAPI)**
Function	Destroys the DVPP API instance created by calling **CreateDvppApi** and closes the DVPP executor.
Input	**IDVPPAPI** pointer reference
Output	None

Return Value	• **0**: success • **–1**: failure
Instructions	None
Restriction	To continue to invoke DVPP after **DestroyDvppApi** is called, call **CreateDvppApi** to create a DVPP API instance.

Calling Example

```
DestroyDvppApi(pidvppapi);
```

2.4 DvppGetOutParameter

Syntax	**int32_t DvppGetOutParameter(void* in, void* out, int32_t cmd)**
Function	Obtains the output buffer size of the JPEGD/JPEGE/PNGD module.
Input	• **in** pointer, which varies according to the module. For details, see **Table 2-12**. • **out** pointer, which varies according to the module. For details, see **Table 2-12**. • **Command word (CMD)**
Output	Output buffer size of each module: • For the JPEGE function, the output buffer size is the value of **jpgSize** in the **sJpegeOut** struct. • For the JPEGD function, the output buffer size is the value of **yuvDataSize** in the **JpegdOut** struct. • For the PNGD function, the output buffer size is the value of **outputSize** in the **PngOutputInfoAPI** struct.
Return Value	• **0**: success • **–1**: failure
Instructions	The caller calls the **DvppGetOutParameter** function, passes the **in** and **out** pointers and the correct CMD command word. The calling example is as follows: • For details about the JPEGD module, see **6.3 Implementing the JPEGD Function**. • For details about the JPEGE module, see **6.2 Implementing the JPEGE Function**. • For details about the PNGD module, see **6.4 Implementing the PNGD Function**.
Restriction	None

Table 2-12 Input parameter description

Input Parameter	Description	Value Range
void* in	Pointer to the input structure	• Input parameter of the JPEGE function: **sJpegeIn** • Input parameter of the JPEGD function: **JpegdIn** • Input parameter of the PNGD function: **PngInputInfoAPI** **NOTICE** This function does not support the **jpegd_raw_data_info** structure of the JPEGD.
void* out	Pointer to the output structure	• Output parameter of the JPEGE function: **sJpegeOut** • Output parameter of the JPEGD function: **JpegdOut** • Output parameter of the PNGD function: **PngOutputInfoAPI** **NOTICE** This function does not support the **jpegd_yuv_data_info** structure of the JPEGD.
int32_t cmd	Command word for obtaining output parameters	• Command word of the JPEGE function: **GET_JPEGE_OUT_PARAMETER** • Command word of the JPEGD function: **GET_JPEGD_OUT_PARAMETER** • Command word of the PNGD function: **GET_PNGD_OUT_PARAMETER**

3 VDEC Function Interfaces

3.1 General Description

For a video stream, after you create an instance by calling **CreateVdecApi**, you must use the same instance to call **VdecCtl** for video decoding, and then call **DestroyVdecApi** to release the instance.

During video decoding, if you need to switch from a video stream to another video stream, you must release the instance of the previous stream by calling **DestroyVdecApi**, and then create an instance by calling **CreateVdecApi** to process the new video stream.

3.2 CreateVdecApi

Syntax	**int CreateVdecApi(IDVPPAPI*& pIDVPPAPI, int singleton)**
Function	Obtains a VDEC API instance, which is equivalent to the handle to the VDEC executor. The caller can use the obtained VDEC API instance to call **CreateVdecApi** for video decoding. Cross-function calling and cross-thread calling are supported.
Input	• **IDVPPAPI** pointer reference. The input pointer must be **NULL**. • The singleton is reserved for internal use to implement a single **pIDVPPAPI** instance in the future. It is recommended that the caller set the parameter to **0** currently.

Output	**IDVPPAPI** pointer reference. The output pointer may be or may not be **NULL**. If a DVPP API instance fails to be obtained, the output pointer is **NULL**. If a DVPP API instance is successfully obtained, the output pointer is not **NULL**.
Return Value	• **0**: success • **–1**: failure
Instructions	The caller creates the **IDVPPAPI** object pointer, which is initialized to **NULL**. The **IDVPPAPI** object pointer is passed by calling the **CreateVdecApi** function. If the application is successful, **CreateVdecApi** returns the **DvppApi** instance. Otherwise, **NULL** is returned. The caller needs to verify the return value.
Restriction	The caller is responsible for the life cycle of the VDEC API instance, including the application and release, which are implemented by using **CreateVdecApi** and **DestroyVdecApi**, respectively.

3.3 VdecCtl

Syntax	**int VdecCtl(IDVPPAPI*& pIDVPPAPI, int CMD, dvppapi_ctl_msg* MSG, int singleton)**
Function	Controls the DVPP executor to implement video decoding. The **VdecCtl** API is called by using the instance created by **CreateVdecApi**.
Input	• **IDVPPAPI** pointer reference • For the VDEC module, the command word (**CMD**) is **DVPP_CTL_VDEC_PROC**. • Configuration information (**MSG**) of the VDEC executor in the **dvppapi_ctl_msg** type. For details about the **in** pointer in this struct, see **Input Parameter: vdec_in_msg**. • The singleton is reserved for internal use to implement a single **pIDVPPAPI** instance in the future. It is recommended that the caller set the parameter to **0** currently.
Output	Output buffer in the configuration information (**MSG**) and output status information of the VDEC (which is all stored in **MSG**)
Return Value	• **0**: This API is called successfully, but it does not mean that the decoding is successful (because the VDEC is asynchronous). • **–1**: This API fails to be called.

Instructions	The caller calls the **VdecCtl** function to pass the **IDVPPAPI** object pointer, the configured **dvppapi_ctl_msg**, and the correct command word (**CMD**). The VDEC is asynchronous. When **VdecCtl** is called, the data in the user stream buffer is copied to the internal input buffer of the VDEC and then returned. Therefore, the decoding is not necessarily successful (only the data is copied successfully) after **VdecCtl** is called. In addition, after **VdecCtl** is called, the user stream buffer is released.
Restriction	For a video stream, after you create an instance by calling **CreateVdecApi**, you must use the same instance to call **VdecCtl** for video decoding, and then call **DestroyVdecApi** to release the instance.

Input Parameter: vdec_in_msg

Table 3-1 Input parameter vdec_in_msg

Member Variable	Description
char video_format[10]	Input video format, for example, **h264** or **h265**. The default value is **h264**. Only H.264 and H.265 streams after YUV420SP (NV12 and NV21) encoding are supported.
char image_format[10]	Output frame format, for example, **nv12** or **nv21**. The default value is **nv12**.
void (*call_back)(FRAME* frame,void * hiai_data)	Callback function. **FRAME** is the output structure after VDEC decoding. For details, see **FRAME structure** in **7.2 Structures and Classes in vdec_in_msg**. You can obtain the output result based on this pointer. It is recommended that only **DvppCtl** be called in the callback function to output YUV image data. Do not implement other functionality in the callback function. Otherwise, it may take a long time and block the waiting resources during decoding. The maximum consumption allowed by the callback function is related to the frame rate. The calculation formula is as follows: Maximum time consumption = 1/Frame rate. For example, if the frame rate is 30 fps, the maximum time consumption is 0.033s (1/30 fps). If the frame rate is 25 fps, the maximum consumption is 0.04s (1/25 fps).

Member Variable	Description
char* in_buffer	Buffer of input H.264 or H.265 raw video streams. The buffer for storing the streams (allocated by the user) before decoding is assigned to **in_buffer**. After **VdecCtl** is called and a return value is received, this buffer can be freed.
int32_t in_buffer_size	Buffer size of input video streams
void * hiai_data	Pointer to the formal parameter of the output frame callback function after decoding. The object that the pointer points to is defined by the caller. **NOTICE** Only **hiai_data** that is passed to the VDEC for the first time is called. To use **hiai_data** to pass different objects for multiple times, use the following smart pointer object.

Member Variable	Description
std::shared_ptr<HIAI_DATA_SP> hiai_data_sp	If the frame sequence number is not required, this parameter does not need to be specified and the default value is **NULL**. The frame sequence number is described as follows. Pointer to the formal parameter of the callback function. **HIAI_DATA_SP** is the parent class defined in the VDEC. For details, see **HIAI_DATA_SP class** in **7.2 Structures and Classes in vdec_in_msg**. You can inherit the class to define subclasses. For details, see the sample code in **6.5 Implementing the VDEC Function**. The class object to which this pointer points must be configured with the frame sequence number. Each frame (I-frame, P-frame, or B-frame) maps to a unique sequence number. The frame sequence numbers start from 1 with an increment of 1. **Notes:** 1. The objects of the user-defined subclass **hiai_data_sp** cannot contain member variables that require large memory. For the decoding of each video stream, the VDEC supports a maximum of 100 **hiai_data_sp** objects. If the memory requested by the member variables is too large, the memory usage may be too high. 2. If the objects of the user-defined subclass **hiai_data_sp** contain member variables that request allocation of memory, free the memory in the destructor function after the memory is allocated to avoid memory leaks. 3. During video stream decoding, if a **hiai_data_sp** object is used but an abnormal frame exists in the video stream, the VDEC directly discards the **hiai_data_sp** object of the abnormal frame. Therefore, you are advised to handle the abnormal frame in the implementation code of the destructor function. 4. During video stream decoding, if a **hiai_data_sp** object is used, the reference frame interval of the stream cannot be greater than 30 frames. For example, frame 1 can be referenced for frame 30 rather than frame 31. 5. The VDEC caches the **hiai_data_sp** objects passed by the user to a queue. A maximum number of 100 objects are supported. If a frame fails to be decoded, the corresponding **hiai_data_sp** object is discarded at the earliest

Member Variable	Description
	time when all the subsequent 30 frames are decoded. For example, if frame 1 fails to be decoded, the corresponding **hiai_data_sp** object is discarded after frame 31 is decoded. If all frames in the stream are abnormal, the **hiai_data_sp** object corresponding to frame 1 is discarded when frame 101 starts to be decoded. 6. Either **hiai_data_sp** or **hiai_data** can be used. **hiai_data_sp** must be used in conjunction with **channelId**. 7. If the stream contains B-frames, the frames are numbered according to the display sequence instead of the decoding sequence.
int32_t channelId	ID of the VDEC channel corresponding to the input stream. Different values must be set when different streams are decoded at the same time. The value range is 0–15.
void (*err_report) (VDECERR* vdecErr)	Error reporting callback function, which is used to notify users of decoding exceptions, such as stream errors, hardware faults, and decoder status errors, for subsequent troubleshooting. The **VDECERR** formal parameter is a structure defined in the VDEC. For details, see **VDECERR structure** in **7.2 Structures and Classes in vdec_in_msg**.
bool isEOS	End of stream (EOS) flag indicating that the decoding of the current channel is complete. You can ignore this flag. By default, **isEOS** is set to **true** in the **DestroyVdecApi** API so that resources are released when the decoding of the channel ends. If **isEOS** is set to **true**, the user-specified configuration is preferentially used in the calling of the **VdecCtl** API so that resources are released when the decoding of the channel ends. For details, see **6.5 Implementing the VDEC Function**.
int32_t isOneInOneOutMode	Reserved. Use the default value **0**.

3.4 DestroyVdecApi

Syntax	int DestroyVdecApi(IDVPPAPI*& pIDVPPAPI, int singleton)

Function	Releases the VDEC API instance created by **CreateVdecApi** and closes the VDEC executor.
Input	**IDVPPAPI** pointer reference. The singleton is reserved for internal use to implement a single **pIDVPPAPI** instance in the future. It is recommended that the caller set the parameter to **0** currently.
Output	None
Return Value	• **0**: success • **–1**: failure
Instructions	None
Restriction	To continue to invoke the VDEC after **DestroyVdecApi** is called, create a VDEC API instance.

4 VENC Function Interfaces

4.1 General Description

To encode multiple images into a video, call **RunVenc** to encode the video by using the same instance after creating an instance by calling **CreateVenc**, and then call **DestroyVenc** to release the instance.

4.2 CreateVenc

Syntax	**int32_t CreateVenc(struct VencConfig* vencConfig)**
Function	Obtains the VENC instance, which is equivalent to obtaining the handle of the VENC executor. The caller can call **RunVenc** to encode images by using the obtained VENC instance.
Input	Pointer to the **VencConfig** structure. For details about the **VencConfig** structure, see **Input parameter: VencConfig**. You need to configure a callback function to process the encoding result.
Output	None
Return Value	• If the return value is a non-negative number, a VENC instance is successfully created. • The return value **–1** indicates that a VENC instance fails to be created.

Instructions	The caller creates the **VencConfig** object pointer. The **VencConfig** object pointer is passed by calling the **CreateVenc** function. If the application is successful, **CreateVenc** returns the handle to the VENC instance. Otherwise, **–1** is returned. The caller needs to verify the return value.
Restriction	The caller is responsible for the life cycle of the VENC instance, including the application and release, which are implemented by using **CreateVenc** and **DestroyVenc**, respectively.

Input parameter: VencConfig

This input parameter is used when the VENC module is initialized. All member variables of the data structure must be initialized before being used.

Table 4-1 Input parameter VencConfig

Member Variable	Description	Value Range
uint32_t width	Image width	The value is an even number ranging from 128 to 1920.
uint32_t height	Image height	The value is an even number ranging from 128 to 1920.
uint32_t codingType	Video encoding protocols	0–3 • **0**: H.265 Main Profile Level (Only slice streams are supported.) • **1**: H.264 Baseline Profile Level • **2**: H.264 Main Profile Level • **3**: H.264 High Profile Level
uint32_t yuvStoreType	YUV image storage format	The value is **0** or **1**. • **0**: YUV420 semi-planar (NV12) • **1**: YVU420 semi-planar (NV21)
uint32_t keyFrameInterval	I-frame interval	The value range is (0, 65535).

Member Variable	Description	Value Range
VencOutMsgCallBack vencOutMsgCallBack	Callback function, which is used to process the encoding result. When encoding each channel of image data, DVPP (the VENC module) calls the callback function twice for the first frame of image data. The first call is used to process the file header information, and the second call is used to process the image data of the current frame.	**typedef void (*VencOutMsgCallBack)(struct VencOutMsg* vencOutMsg, void* userData)**. The value cannot be null.
void* userData	The user records the information to be passed. The data is returned to the user in the callback function.	The value can be **NULL**.

4.3 RunVenc

Syntax	**int32_t RunVenc(int32_t vencHandle, struct VencInMsg* vencInMsg)**
Function	Controls the DVPP executor to implement video encoding. The **RunVenc** API is called by using the instance created by **CreateVenc**.
Input	**int32_t** handle and **VencInMsg** pointer. **vencHandle** is the return value of **CreateVenc**. For details about **VencInMsg**, see **Input Parameter: VencInMsg**. **VencInMsg** indicates the VENC executor configuration information. This structure is used to transfer the video information to be encoded to the executor for encoding.
Output	None
Return Value	• **0**: success • **–1**: failure
Instructions	The caller calls the **RunVenc** function to transfer the encoding instance, the **VencInMsg** object pointer, and the configured **vencInMsg**.

Restriction	To encode multiple images into a video, call **RunVenc** to encode the video by using the same instance after creating an instance by calling **CreateVenc**, and then call **DestroyVenc** to release the instance.

Input Parameter: VencInMsg

This input parameter is used when the VENC module is called to perform encoding. All member variables of the structure must be initialized before being used.

Table 4-2 Input parameter VencInMsg

Member Variable	Description	Value Range
void* inputData	Address of the input data	The value cannot be null.
uint32_t inputDataSize	Input data size	The value cannot be greater than the size of the input data buffer. The recommended value is the same as the size of the input data buffer.
uint32_t keyFrameInterval	I-frame interval	The value range is [0, 65535). The value **0** indicates that the parameter is invalid.
uint32_t forceIFrame	Whether to forcibly restart the I-frame interval. **0**: No. **1**: Yes.	The value is **0** or **1**.
uint32_t eos	Whether the frame is an end frame. **0**: No. **1**: Yes.	The value is **0** or **1**.

Output Parameter: VencOutMsg

Table 4-3 Output parameter VencOutMsg

Member Variable	Description	Value Range
void* outputData	Output data address	Applied by the VENC internally
uint32_t outputDataSize	Output data size	Configured by the VENC internally

Member Variable	Description	Value Range
uint32_t timeStamp	Timestamp of calling the callback function	Configured by the VENC internally

4.4 DestroyVenc

Syntax	**int32_t DestroyVenc(int32_t vencHandle)**
Function	Releases the VENC instance created by calling **CreateVenc** and closes the VENC executor.
Input	Handle to the **int32_t** VENC instance, which is the return value of the **CreateVenc** function
Output	None
Return Value	• **0**: success • **–1**: failure
Instructions	None
Restriction	To continue to invoke the VENC after **DestroyVenc** is called, create a VENC instance.

5 Compatibility with the Functions of Earlier Versions

5.1 Compatibility

To be compatible with the functions in earlier versions, the current version supports the VPC and VENC functions through the following methods:

- Implementing the VPC function:
  - Pass input parameters to **DvppCtl**, that is, the **DVPP_CTL_VPC_PROC** command word and **resize_param_in_msg** structure parameters.
  - In scenarios that have low requirements on latency and process a large number of low-resolution images, pass input parameters to **DvppCtl**, that is, the **DVPP_CTL_CMDLIST_PROC** command word and **IMAGE_CONFIG** structure parameters.

  This method is not recommended. It is used to be compatible with the functions in earlier versions and will be deleted in later versions. If you use functions that are not recommended, you are advised to migrate them to the recommended VPC function method described in **2 VPC/JPEGE/JPEGD/PNGD Function Interfaces**.
- Implementing the VENC function: Call **CreateDvppApi**, **DvppCtl**, and **DestroyDvppApi** and pass input parameters to **DvppCtl**, that is, the **DVPP_CTL_VENC_PROC** command word and **venc_in_msg** structure parameters.

  This method is not recommended. It is used to be compatible with the functions in earlier versions and will be deleted in later versions. If you use functions that are not recommended, you are advised to migrate them to the recommended VENC function method described in **4 VENC Function Interfaces**.

5.2 VPC Functions and Parameters

You are not advised to use the VPC function of the earlier version, which will be removed in later versions. The VPC and CMDLIST functions in the earlier version have been combined into the VPC function. For details, see **2.2.2 VPC Parameters**.

The VPC APIs of the earlier version are used for media image conversion including scaling, color space conversion (CSC), bit depth reduction, format conversion, block partitioning, overlay, collage, and CMDLIST calling.

- There is no limit on the number of VPC channels.

  The cropping and resizing operations affect the VPC performance. If the image resolution is changed during image processing, the performance is affected. The following table shows the reference performance specifications in typical scenarios when the image resolution during image processing does not exceed the original resolution.

Scenario	Total Frame Rate
1080p x 32 channels	1440 fps
4K x 4 channels	360 fps

  When image resolution during image processing is greater than the original resolution, use the following formula to calculate the frame rate:

  Total frame rate = (3840 x 2160 x 360)/(W x H) fps

  Where, **W** and **H** are the maximum width and height during the VPC processing. For example, if a 1080p image is cropped to 960 x 540 and then resized to 3840 x 2160, **W** is 3840 and **H** is 2160.

  During the processing, the VPC performance slightly decreases when the image scaling-down ratio is increased.

- Restrictions on the width and height of the input and output images of the VPC:
  - The width and height of the input image must be 2-pixel aligned, that is, the width and height must be an even number.
  - The width and height of the output image must be 2-pixel aligned, that is, the width and height must be an even number.

  NOTICE

  The image width and height mentioned here refer to the valid area of an image. The actual buffer sent to the VPC must be 128 x 16 aligned.

- Restrictions on the buffers of the VPC input and output images:
  - Restrictions on the VPC input buffer:
    - Input data address (OS virtual address): 128-byte aligned

- Input image buffer width restriction: 128-byte aligned
- Input image buffer height restriction: 16-byte aligned

NOTE

A YUV400SP image requires a buffer size that matches the YUV420SP format, that is, the required buffer size is (width_align_128 x high_align_16 x 1.5).

– Restrictions on the VPC output buffer:
  - Output data address (OS virtual address): 128-byte aligned
  - Output image buffer width restriction: 128-byte aligned
  - Output image buffer height restriction: 16-byte aligned
– In the same task, the virtual addresses of the input and output buffers must be in the same 4 GB space.

NOTE

When the VPC is used to implement image cropping and scaling, **DvppCtl** needs to be called twice. In the first calling, the input parameter is **resize_param_in_msg**, and the output parameter is **resize_param_out_msg**. In the second calling, the input parameter is **vpc_in_msg**. For details, see **Input Parameter: resize_param_in_msg** to **Input Parameter: vpc_in_msg**.

Input Parameter: resize_param_in_msg

Member Variable	Description	Value Range
int src_width	Width of the original image (2-pixel aligned) **NOTICE** The original image mentioned here refer to the valid area of an image. The actual buffer sent to the VPC must be 128 x 16 aligned.	Minimum resolution: • 128 (from VDEC) • 16 (not from VDEC) Maximum resolution: 4096 This parameter is mandatory.
int src_high	Height of the original image (2-pixel aligned) **NOTICE** The original image mentioned here refer to the valid area of an image. The actual buffer sent to the VPC must be 128 x 16 aligned.	Minimum resolution: • 128 (from VDEC) • 16 (not from VDEC) Maximum resolution: 4096 This parameter is mandatory.
int hmax	Maximum horizontal offset from the origin	The value is an odd number. For details, see **Input Parameter: vpc_in_msg**.

Member Variable	Description	Value Range
int hmin	Minimum horizontal offset from the origin	The value is an even number. For details, see **Input Parameter: vpc_in_msg**.
int vmax	Maximum vertical offset from the origin	The value is an odd number. For details, see **Input Parameter: vpc_in_msg**.
int vmin	Minimum vertical offset from the origin	The value is an even number. For details, see **Input Parameter: vpc_in_msg**.
int dest_width	Width of the output image	The value is an even number
int dest_high	Height of the output image	The value is an even number

Output Parameter: resize_param_out_msg

Member Variable	Description	Value Range
int dest_width_stride	Width alignment value of the output image	128
int dest_high_stride	Height alignment value of the output image	16
int hmax	Maximum horizontal offset from the origin	The value is an odd number. For details, see **Input Parameter: vpc_in_msg**.
int hmin	Minimum horizontal offset from the origin	The value is an even number. For details, see **Input Parameter: vpc_in_msg**.
int vmax	Maximum vertical offset from the origin	The value is an odd number. For details, see **Input Parameter: vpc_in_msg**.

Member Variable	Description	Value Range
int vmin	Minimum vertical offset from the origin	The value is an even number. For details, see **Input Parameter: vpc_in_msg**.
double hinc	Horizontal resizing coefficient	For details, see **Input Parameter: vpc_in_msg**.
double vinc	Vertical resizing coefficient	For details, see **Input Parameter: vpc_in_msg**.

Input Parameter: vpc_in_msg

Member Variable	Description	Value Range
int format	Image format defined by the caller. Use a correct number. For example, the number corresponding to **yuv420_semi_plannar** is **0**. If YUV420SP is declared, set this parameter to **0**.	yuv420_semi_plannar = 0,//0 yuv400_semi_plannar = 0,//0 yuv422_semi_plannar,//1 yuv444_semi_plannar,//2 yuv422_packed,//3 yuv444_packed,//4 rgb888_packed,//5 xrgb8888_packed,//6 This parameter is mandatory.
int rank	Image rank mode defined by the caller. Use a correct number.	For details, see **Table 5-1**. This parameter is mandatory.
int bitwidth	Bit depth. The default value is 8 bits. The 10-bit depth is used only when the image format is YUV/YVU420 Semi-Planar and HFBC compression is used.	**8** or **10** This parameter is optional.
int cvdr_or_rdma	Whether the input image is transmitted through the CVDR or RDMA channel. The CVDR channel is used by default. The input of the RDMA channel can only be HFBC data output by the VDEC.	**1**: CVDR. **0**: RDMA. This parameter is optional.

Member Variable	Description	Value Range
int width	Width of the original input image, which is 2-pixel aligned (even number) **NOTICE** The original image mentioned here refer to the valid area of an image. The actual buffer sent to the VPC must be 128 x 16 aligned.	Minimum resolution: ● 128 (from VDEC) ● 16 (not from VDEC) Maximum resolution: 4096 This parameter is mandatory.
int high	Height of the original input image, which is 2-pixel aligned (even number) **NOTICE** The original image mentioned here refer to the valid area of an image. The actual buffer sent to the VPC must be 128 x 16 aligned.	Minimum resolution: ● 128 (from VDEC) ● 16 (not from VDEC) Maximum resolution: 4096 This parameter is mandatory.
int stride	Image stride **NOTICE** The stride is different for different input formats.	YUV400SP, YUV420SP, YUV422SP, and YUV444SP: Align the width to a 128-pixel boundary. YUV422 Packed: Align (width x 2) to a 128-pixel boundary. YUV444 Packed and RGB888: Align (width x 3) to a 128-pixel boundary. XRGB8888: Align (width x 4) to a 128-pixel boundary. This parameter is mandatory.
double hinc	Horizontal magnification for the AI core output channel	[0.03125, 1) or (1, 4] If a resizing coefficient exceeds the value range, the VPC reports an error. If resizing is not required, set the value to **1**. This parameter is mandatory.
double vinc	Vertical magnification for the AI core output channel	[0.03125, 1) or (1, 4] If a resizing coefficient exceeds the value range, the VPC reports an error. If resizing is not required, set the value to **1**. This parameter is mandatory.

Member Variable	Description	Value Range
double jpeg_hinc	Horizontal magnification for the JPEGE output channel	[0.03125, 1) or (1, 4] If a resizing coefficient exceeds the value range, the VPC reports an error. This parameter is optional and is not applicable currently.
double jpeg_vinc	Vertical magnification for the JPEGE output channel	[0.03125, 1) or (1, 4] If a resizing coefficient exceeds the value range, the VPC reports an error. This parameter is optional and is not applicable currently.
int hmax	Maximum horizontal offset from the origin	The value must be an odd number. If cropping is not required, set the value to (Input image width – 1). This parameter is mandatory.
int hmin	Minimum horizontal offset from the origin	The value must be an even number. If cropping is not required, set the value to **0**. This parameter is mandatory.
int vmax	Maximum vertical offset from the origin	The value must be an odd number. If cropping is not required, set the value to (Input image width – 1). This parameter is mandatory.
int vmin	Minimum vertical offset from the origin	The value must be an even number. If cropping is not required, set the value to **0**.
int out_width	Width of the output image	The value must be an even number (2-pixel aligned). When a smart pointer is used to allocate the output buffer, this parameter is not required. When a buffer allocated by the user is used as the output buffer, this parameter is mandatory.

Member Variable	Description	Value Range
int out_high	Height of the output image	The value must be an even number (2-pixel aligned). When a smart pointer is used to allocate the output buffer, this parameter is not required. When a buffer allocated by the user is used as the output buffer, this parameter is mandatory.
int h_stride	Image height stride, which is the same as that described in **int stride**. You do not need to set this parameter.	The default value is **16**. That is, the default input image height is 16-pixel aligned. This parameter is optional.
char* in_buffer	Input buffer	Ensure that the input buffer size is the same as the length and width configured in **in_msg**. If the input image is in YUV420 Semi-Planar NV12 format, the image size is (Width x Height x 1.5). If the input buffer size is inconsistent with this calculation result, the VPC fails to be called. This parameter is mandatory.
int in_buffer_size	Size of the input buffer	This value is used to verify the input buffer. This parameter is mandatory.
shared_ptr<AutoBuffer> auto_out_buffer_1	Output buffer 1	This buffer needs to be configured when the basic VPC functions (cropping and resizing) are used. This buffer is used for the smart pointer allocated by the caller and is used as an input parameter of the DVPP. The output buffer is allocated inside the DVPP. You can read this buffer to obtain the output image. You only need to set either this parameter or the subsequent parameter **out_buffer**.

Member Variable	Description	Value Range
char* out_buffer	Output buffer	This buffer needs to be configured when the basic VPC functions (cropping and resizing) are used. This buffer is allocated by the caller. The start address of the buffer must be 128-byte aligned, and the allocated memory must be in the 4 GB space. You can read this buffer to obtain the output image. You only need to set either this parameter or the previous parameter **auto_out_buffer_1**.
int out_buffer_1_size	Size of output buffer 1	This value is used to verify the output buffer. If **shared_ptr<AutoBuffer> auto_out_buffer_1** is used to allocate the output buffer, this parameter is optional. If **char* out_buffer** is used to allocate the output buffer, this parameter is mandatory.
shared_ptr<AutoBuffer> auto_out_buffer_2	Output of the other VPC channel (reserved interface)	--
int out_buffer_2_size	Size of output buffer 2 of the **auto_out_buffer2** smart pointer. This is a reserved interface.	--
int use_flag	Flag indicating whether the VPC function is used	• **0**: basic functions of the VPC • **1**: raw data 8K resizing • **2**: raw data overlay (reserved) • **3**: raw data stitching (reserved) The default value is **0**. This parameter is optional.
VpcOverLayReverse overlay	Overlay structure, which is reserved	Reserved
VpcCollageReverse collage	Collage structure, which is reserved	Reserved

Member Variable	Description	Value Range
RDMACHANNEL rdma	Configuration structure dedicated to the HFBC data	For details, see **7.5.1 RDMACHANNEL Structure**.
VpcTurningReverse turningReverse1	Reserved interface for VPC tuning	For details, see **7.5.2 VpcTurningReverse Structure**.
bool isVpcUseTurning1	Flag indicating whether VPC tuning is used. This is a reserved interface and is used together with **turningReverse1**.	--
VpcTurningReverse turningReverse2	Reserved interface for VPC tuning	For details, see **7.5.2 VpcTurningReverse Structure**.
bool isVpcUseTurning2	Flag indicating whether VPC tuning is used. This is a reserved interface and is used together with **turningReverse2**.	--
string *yuvscaler_paraset	Pointer to the file path and file name array of the VPC filtering parameter set. **NOTE** • The parameter set must be on the device. • Ensure that the input file path is correct.	A parameter set file path consists of an absolute file path on the device and the file name, for example, **{"/home/HwHiAiUser/matrix/YUVScaler_pra.h"}**.
unsigned int yuvscaler_paraset_size	Number of elements in the string array to which **yuvscaler_paraset** points. The default value is **1**.	yuvscaler_paraset_size<=10
unsigned int index	Index of the **yuvscaler_paraset** array	0<=index<10

Table 5-1 Rank configuration table of the input and output image formats

Input Image Format	Rank Parameter Configuration of the VPC	Output Image Format
YUV420sp NV12	NV12 = 0	YUV420sp NV21

Input Image Format	Rank Parameter Configuration of the VPC	Output Image Format
YUV420sp NV21	NV21 = 1	YUV420sp NV21
YUV420sp NV12	NV12 = 1	YUV420sp NV12
YUV420sp NV21	NV21 = 0	YUV420sp NV12
YUV422sp NV16	NV16 = 1	YUV420sp NV12
YUV422sp NV61	NV61 = 0	YUV420sp NV12
YUV422sp NV16	NV16 = 0	YUV420sp NV21
YUV422sp NV61	NV61 = 1	YUV420sp NV21
YUV444sp 444spUV	444spUV = 1	YUV420sp NV12
YUV444sp 444spVU	444spVU= 0	YUV420sp NV12
YUV444sp 444spUV	444spUV = 0	YUV420sp NV21
YUV444sp 444spVU	444spVU= 1	YUV420sp NV21
YUV422packet YUYV	YUYV = 2	YUV420sp NV12
YUV422packet YVYU	YVYU = 3	YUV420sp NV21
YUV422packet UYVY	UYVY = 4	YUV420sp NV12
YUV444packet YUV	YUV = 5	YUV420sp NV12
RGB888 RGB	RGB = 6	YUV420sp NV12
RGB888 BGR	BGR = 7	YUV420sp NV21
RGB888 RGB	BGR = 7	YUV420sp NV21
RGB888 BGR	RGB = 6	YUV420sp NV12
XRGB8888 RGBA	RGBA = 8	YUV420sp NV12
XRGB8888 RGBA	BGRA = 9	YUV420sp NV21

Calling Example

- Example of calling basic VPC functions:

  g*Xxxxx* is a configuration parameter. Configuring the parameter and calling the VPC are two steps.

  Step 1: Use the **DVPP_CTL_TOOL_CASE_GET_RESIZE_PARAM** command word to call **DvppCtl** to obtain the configuration parameters. The input parameter of **DvppCtl** is **resize_param_in_msg**, and the output parameter is **resize_param_out_msg**. Use **resize_param_in_msg** to send the width and height of the original image, cropped area (hmax,hmin,vmax,vmin), and output width and height to the DVPP. The DVPP returns the regenerated

cropped area (hmax,hmin,vmax,vmin) and resizing ratio (hinc,vinc) to the caller after calculation.

Step 2: The caller transfers the parameters returned in step 1 to the DVPP by using **vpc_in_msg**, changes the **DvppCtl** command word to **DVPP_CTL_VPC_PROC** to obtain the processing result.

```
dvppapi_ctl_msg dvppApiCtlMsg;
vpc_in_msg vpcInMsg;
resize_param_in_msg resize_in_param;
resize_param_out_msg resize_out_param;
resize_in_param.src_width = gWidth;
resize_in_param.src_high = gHigh;
resize_in_param.hmax = gHmax;
resize_in_param.hmin = gHmin;
resize_in_param.vmax = gVmax;
resize_in_param.vmin = gVmin;
resize_in_param.dest_width = floor(gHinc * (gHmax - gHmin + 1) + 0.5);
resize_in_param.dest_high = floor(gVinc * (gVmax - gVmin + 1) + 0.5);
printf("width  =%d\n", gWidth);
printf("high   =%d\n", gHigh);
printf("hmax   =%d\n", gHmax);
printf("hmin   =%d\n", gHmin);
printf("vmax   =%d\n", gVmax);
printf("vmin   =%d\n", gVmin);
printf("out width   =%d\n", resize_in_param.dest_width);
printf("out hight   =%d\n", resize_in_param.dest_high );

dvppApiCtlMsg.in = (void *)(&resize_in_param);
dvppApiCtlMsg.out = (void *)(&resize_out_param);
IDVPPAPI *pidvppapi = NULL;
CreateDvppApi(pidvppapi);
if (DvppCtl(pidvppapi, DVPP_CTL_TOOL_CASE_GET_RESIZE_PARAM, &dvppApiCtlMsg) != 0) {
    printf("call DVPP_CTL_TOOL_CASE_GET_RESIZE_PARAM process faild!\n");
    DestroyDvppApi(pidvppapi);
    return;
}

int bufferSize        = 0;
vpcInMsg.format       = gFormat;
vpcInMsg.cvdr_or_rdma = gcvdr_or_rdma;
vpcInMsg.bitwidth = gBitwidth;
vpcInMsg.rank = gRank;
vpcInMsg.width = gWidth;
vpcInMsg.high = gHigh;
vpcInMsg.stride = gStride;
vpcInMsg.hmax = resize_out_param.hmax;
vpcInMsg.hmin = resize_out_param.hmin;
vpcInMsg.vmax = resize_out_param.vmax;
vpcInMsg.vmin = resize_out_param.vmin;
vpcInMsg.vinc = resize_out_param.vinc;
vpcInMsg.hinc = resize_out_param.hinc;
printf("resize_out_param.hinc = %lf\n", resize_out_param.hinc);
printf("resize_out_param.vinc = %lf\n", resize_out_param.vinc);
printf("resize_out_param.hmax = %d\n", resize_out_param.hmax);
printf("resize_out_param.hmin = %d\n", resize_out_param.hmin);
printf("resize_out_param.vmax = %d\n", resize_out_param.vmax);
printf("resize_out_param.vmin = %d\n", resize_out_param.vmin);

printf("format =%d\n", vpcInMsg.format);
printf("rank   =%d\n", vpcInMsg.rank);
printf("width  =%d\n", vpcInMsg.width);
printf("high   =%d\n", vpcInMsg.high);
printf("stride =%d\n", vpcInMsg.stride);
printf("hmax   =%d\n", vpcInMsg.hmax);
printf("hmin   =%d\n", vpcInMsg.hmin);
printf("vmax   =%d\n", vpcInMsg.vmax);
printf("vmin   =%d\n", vpcInMsg.vmin);
printf("vinc   =%F\n", vpcInMsg.vinc);
```



```
printf("hinc   =%F\n", vpcInMsg.hinc);
if (vpcInMsg.cvdr_or_rdma == 0) {
    printf("rdma channel\n");
    int buffersize ;
    char * payloadY ;
    char * payloadC ;
    char * headY ;
    char * headC ;
    FILE * rdmaimage = fopen(in_file_name, "rb");

    if (vpcInMsg.bitwidth == 10) {
        buffersize                  = vpcInMsg.width  * vpcInMsg.high * 2 + 512 * 1024;
        payloadY                    = (char*)memalign(128, buffersize);
        headY                       = payloadY + vpcInMsg.width  * vpcInMsg.high * 2;
        headC                       = headY + 256 * 1024;
        vpcInMsg.rdma.luma_payload_addr   = (long)payloadY;
        vpcInMsg.rdma.chroma_payload_addr = (long)payloadY + 640;
    } else {
        buffersize                  = vpcInMsg.width * vpcInMsg.high * 1.5 + 512 * 1024;
        payloadY                    = (char*)memalign(128, buffersize);
        payloadC                    = payloadY + vpcInMsg.width * vpcInMsg.high;
        headY                       = payloadC + vpcInMsg.width * vpcInMsg.high / 2;
        headC                       = headY + 256 * 1024;
        vpcInMsg.rdma.luma_payload_addr   = (long)payloadY;
        vpcInMsg.rdma.chroma_payload_addr = (long)payloadC;
    }

    fread(payloadY, 1, buffersize, rdmaimage);
    vpcInMsg.rdma.luma_head_addr        = (long)headY;
    vpcInMsg.rdma.chroma_head_addr      = (long)headC;
    vpcInMsg.rdma.luma_head_stride      = vpcInMsg.stride;
    vpcInMsg.rdma.chroma_head_stride    = vpcInMsg.stride;
    vpcInMsg.rdma.luma_payload_stride   = gpYStride;
    vpcInMsg.rdma.chroma_payload_stride = gpUVStride;
    fclose(rdmaimage);
    printf("luma payload stride= %d\n", vpcInMsg.rdma.luma_payload_stride );
    printf("chroma payload stride= %d\n", vpcInMsg.rdma.chroma_payload_stride );
} else {
    alloc_buffer(vpcInMsg.width, vpcInMsg.high, vpcInMsg.stride, bufferSize);
    if (open_image(in_file_name, vpcInMsg.width, vpcInMsg.high, vpcInMsg.stride) != 0) {
        printf("open file failed~\n");
        free(in_buffer);
        return;
    }
    vpcInMsg.in_buffer      = in_buffer;
    vpcInMsg.in_buffer_size = bufferSize;
}
//alloc in and out buffer
shared_ptr<AutoBuffer> auto_out_buffer = make_shared<AutoBuffer>();
vpcInMsg.auto_out_buffer_1              = auto_out_buffer;
dvppApiCtlMsg.in                       = (void *)(&vpcInMsg);
dvppApiCtlMsg.in_size                   = sizeof(vpc_in_msg);

if (pidvppapi != NULL) {

    for (int i = 0; i < gLoop; i++) {
        if (DvppCtl(pidvppapi, DVPP_CTL_VPC_PROC, &dvppApiCtlMsg) != 0) {
            printf("call dvppctl process faild!\n");
            DestroyDvppApi(pidvppapi);
            free(in_buffer);
            return;
        }
    }

} else {
    printf("pidvppapi is null!\n");
}

free(in_buffer);
```

```
    FILE *fp = NULL;
    fp = fopen(out_file_name, "wb+");
    if (fp == NULL) {
        printf("open file %s failed~\n", out_file_name);
        free(in_buffer);
        return;
    }

    fwrite(vpcInMsg.auto_out_buffer_1->getBuffer(), 1, vpcInMsg.auto_out_buffer_1->getBufferSize(),
fp);
    fflush(fp);
    fclose(fp);
    DestroyDvppApi(pidvppapi);
    return;
```

- Example of calling basic raw data 8K functions:

  Parameter settings:

```
    dvppapi_ctl_msg dvppApiCtlMsg;
    vpc_in_msg vpcInMsg;
    vpcInMsg.width = gWidth;
    vpcInMsg.high = gHigh;
    vpcInMsg.stride = gStride;
    vpcInMsg.out_width = gOutWidth;
    vpcInMsg.out_high = gOutHigh;
    printf("width     =%d\n", vpcInMsg.width);
    printf("high      =%d\n", vpcInMsg.high);
    printf("stride    =%d\n", vpcInMsg.stride);
    printf("out_width =%d\n", vpcInMsg.out_width);
    printf("out_high  =%d\n", vpcInMsg.out_high);
    //alloc in and out buffer
    char *in_buffer = (char *)malloc(vpcInMsg.width * vpcInMsg.high * 1.5);
    if (in_buffer = NULL) {
        printf("malloc fail\n");
        return;
    }

    shared_ptr<AutoBuffer> auto_out_buffer = make_shared<AutoBuffer>();
    FILE *yuvimage = fopen(in_file_name, "rb");
    if (yuvimage == NULL) {
        printf("no such file!,file_name=[%s]\n", in_file_name);
        free(in_buffer);
        return;
    } else {
        fread(in_buffer, 1, vpcInMsg.width * vpcInMsg.high * 3 / 2, yuvimage);
    }

    vpcInMsg.rank = gRank;
    vpcInMsg.in_buffer = in_buffer;
    vpcInMsg.in_buffer_size = vpcInMsg.width * vpcInMsg.high * 1.5;
    vpcInMsg.auto_out_buffer_1 = auto_out_buffer;
    vpcInMsg.use_flag = 1;
    dvppApiCtlMsg.in = (void *)(&vpcInMsg);
    dvppApiCtlMsg.in_size = sizeof(vpc_in_msg);
    IDVPPAPI *pidvppapi = NULL;
    CreateDvppApi(pidvppapi);
    if (pidvppapi != NULL) {
        if (DvppCtl(pidvppapi, DVPP_CTL_VPC_PROC, &dvppApiCtlMsg) != 0) {
            printf("call dvppctl process faild!\n");
            DestroyDvppApi(pidvppapi);
            free(in_buffer);
            fclose(yuvimage);
            return;
        }
    } else {
        printf("pidvppapi is null!\n");
    }

    DestroyDvppApi(pidvppapi);
```



```
    FILE *fp = NULL;
    fp = fopen(out_file_name, "wb+");
    if (fp != NULL) {
        fwrite(vpcInMsg.auto_out_buffer_1->getBuffer(), 1, vpcInMsg.auto_out_buffer_1-
>getBufferSize(), fp);
        fflush(fp);
        fclose(fp);
    } else {
        fclose(yuvimage);
        return;
    }

    free(in_buffer);
    fclose(yuvimage);
    return;
```

- Example of calling basic raw data overlay functions:

```
    dvppapi_ctl_msg dvppApiCtlMsg;
    vpc_in_msg vpcInMsg;
    // Configures raw data overlay parameters.
    vpcInMsg.use_flag = 2;
    vpcInMsg.overlay.image_type = 0;
    vpcInMsg.overlay.image_rank_type = 0;
    vpcInMsg.overlay.bit_width = 8;
    vpcInMsg.overlay.in_width_image = 1000;
    vpcInMsg.overlay.in_high_image = 1000;
    vpcInMsg.overlay.in_width_text = 500;
    vpcInMsg.overlay.in_high_text = 390;
    vpcInMsg.overlay.image_width_stride = 2;
    vpcInMsg.overlay.image_high_stride = 2;
    vpcInMsg.overlay.text_width_stride = 2;
    vpcInMsg.overlay.text_high_stride = 2;
    vpcInMsg.overlay.in_buffer_image = NULL;
    vpcInMsg.overlay.in_buffer_text = NULL;
    vpcInMsg.overlay.auto_overlay_out_buffer = make_shared<AutoBuffer>();
    // Initializes the image buffer.
    if(NULL == (vpcInMsg.overlay.in_buffer_image =
(char*)malloc(vpcInMsg.overlay.in_width_image*vpcInMsg.overlay.in_high_image*3/2)))    {
        printf("in_buffer_image alloc failed!\n");
        return ;
    }
    // Opens the image and sends the data to the allocated buffer.
    char image_file[50] = "yuv_data_0";
    FILE * yuvimage = fopen(image_file,"rb");
    if(NULL == yuvimage)    {
        printf("VPC_TEST: yuv image open faild!");
        return ;
    }   else    {
        fread(vpcInMsg.overlay.in_buffer_image,
1,vpcInMsg.overlay.in_width_image*vpcInMsg.overlay.in_high_image*3/2,yuvimage);
        fclose(yuvimage);
        yuvimage = NULL;
}

    // Initializes the text buffer.
    if(NULL == (vpcInMsg.overlay.in_buffer_text =
(char*)malloc(vpcInMsg.overlay.in_width_text*vpcInMsg.overlay.in_high_text*3/2)))    {
        printf("in_buffer_text alloc failed!");
        free(vpcInMsg.overlay.in_buffer_image);
        return ;
    }

    // Opens the text-based image and sends the data to the allocated memory.
    char text_file[50] = "text_0";
    FILE * yuvtext = fopen(text_file,"rb");
    if(NULL == yuvtext)    {
        printf("VPC_TEST: yuv text image open faild!");
        free(vpcInMsg.overlay.in_buffer_image);
        return ;
    }   else    {
```

```
        fread(vpcInMsg.overlay.in_buffer_text,
1,vpcInMsg.overlay.in_width_text*vpcInMsg.overlay.in_high_text*3/2, yuvtext);
        fclose(yuvtext);
        yuvtext = NULL;
    }

    dvppApiCtlMsg.in = (void*)(&vpcInMsg);
    dvppApiCtlMsg.in_size = sizeof(vpc_in_msg);
    IDVPPAPI *pidvppapi = NULL;
    CreateDvppApi(pidvppapi);
    if(pidvppapi!=NULL)    {
        if(DvppCtl(pidvppapi,DVPP_CTL_VPC_PROC,&dvppApiCtlMsg)!= 0)        {
            printf("call dvppctl process faild!\n");
            DestroyDvppApi(pidvppapi);
            free(vpcInMsg.overlay.in_buffer_image);
            free(vpcInMsg.overlay.in_buffer_text);
            return ;
        }
        DestroyDvppApi(pidvppapi);
        FILE * fp   = fopen("./out_overlay_image_share","wb+");
        fwrite(vpcInMsg.overlay.auto_overlay_out_buffer->getBuffer(),
1,vpcInMsg.overlay.in_width_image*vpcInMsg.overlay.in_high_image*3/2,fp);
        fflush(fp);
        fclose(fp);
        fp=NULL;
        return ;
    }   else    {
        printf("pidvppapi is null!\n");
        return ;
    }
    if(NULL != vpcInMsg.overlay.in_buffer_image)    {
        free(vpcInMsg.overlay.in_buffer_image);
        vpcInMsg.overlay.in_buffer_image = NULL;
    }
    if(NULL != vpcInMsg.overlay.in_buffer_text)    {
        free(vpcInMsg.overlay.in_buffer_text);
        vpcInMsg.overlay.in_buffer_text = NULL;
    }
```

- Example of calling basic raw data stitching functions:

```
    dvppapi_ctl_msg dvppApiCtlMsg;
    vpc_in_msg vpcInMsg;
     // Configures raw data stitching parameters.
    vpcInMsg.use_flag = 3;
    vpcInMsg.collage.image_type = 0;
    vpcInMsg.collage.image_rank_type = 0;
    vpcInMsg.collage.bit_width = 8;
    vpcInMsg.collage.in_width = 1000;
    vpcInMsg.collage.in_high = 1000;
    vpcInMsg.collage.width_stride = 2;
    vpcInMsg.collage.high_stride = 2;
    vpcInMsg.collage.collage_type = 0;
    vpcInMsg.collage.auto_out_buffer = make_shared<AutoBuffer>();
    char image_file[4][50] = {"yuv_data_0","yuv_data_1","yuv_data_2","yuv_data_3"};
    // Initializes the image buffer (without stride data)**********begin**********.
    for(int i=0; i<4; i++) {
        if(NULL == (vpcInMsg.collage.in_buffer[i] =
(char*)malloc(vpcInMsg.collage.in_width*                                    vpcInMsg.collage.in_high*3/2)))
        {
            printf("in_buffer_image alloc failed!\n");
            for(int j=0; j<i; j++) {
                free(vpcInMsg.collage.in_buffer[j]);
            }
            return ;
        }
       // Opens the image and sends the data to the allocated buffer.
        FILE * yuvimage = fopen(image_file[i],"rb");
        if(NULL == yuvimage)        {
            printf("PVC_TEST: yuv image open faild!");
            return ;
```



```
        }       else       {
            fread(vpcInMsg.collage.in_buffer[i], 1,vpcInMsg.collage.in_width*
                vpcInMsg.collage.in_high*3/2,yuvimage);
            fclose(yuvimage);
            yuvimage = NULL;
        }
    }

    dvppApiCtlMsg.in = (void*)(&vpcInMsg);
    dvppApiCtlMsg.in_size = sizeof(vpc_in_msg);
    IDVPPAPI *pidvppapi = NULL;
    CreateDvppApi(pidvppapi);
    if(pidvppapi!=NULL)    {
        if(DvppCtl(pidvppapi,DVPP_CTL_VPC_PROC,&dvppApiCtlMsg)!= 0)       {
            printf("call dvppctl process faild!\n");
            DestroyDvppApi(pidvppapi);
            for(int i=0; i<4; i++) {
                free(vpcInMsg.collage.in_buffer[i]);
            }
            return ;
        }
        DestroyDvppApi(pidvppapi);
        FILE * fp   = fopen("./out_collage_image_share","wb+");
        fwrite(vpcInMsg.collage.auto_out_buffer->getBuffer(),
1,vpcInMsg.collage.in_width*vpcInMsg.collage.in_high*6,fp);
        fflush(fp);
        fclose(fp);
        fp=NULL;
        return ;
    }   else    {
        printf("pidvppapi is null!\n");
        return ;
    }
    for(int i=0; i<4; i++) {
        if(NULL != vpcInMsg.collage.in_buffer[i])       {
            free(vpcInMsg.collage.in_buffer[i]);
            vpcInMsg.collage.in_buffer[i] = NULL;
        }
    }
```

- Example of calling the VPC API:

```
IDVPPAPI *pidvppapi = NULL;
CreateDvppApi(pidvppapi);
if(pidvppapi!=NULL)
{
 if(0 != DvppCtl(pidvppapi,DVPP_CTL_VPC_PROC,&dvppApiCtlMsg))
 {
  printf("call dvppctl process faild!\n");
  DestroyDvppApi(pidvppapi);
  return -1;
 }
}
else
printf("pidvppapi is null!\n");
DestroyDvppApi(pidvppapi);
```

- Example of calling functions for configuring and obtaining VPC resizing parameters:

  g*Xxxxx* is a configuration parameter.

```
dvppapi_ctl_msg dvppApiCtlMsg;
vpc_in_msg vpcInMsg;
resize_param_in_msg resize_in_param;
resize_param_out_msg resize_out_param;
resize_in_param.src_width = gWidth;
resize_in_param.src_high = gHigh;
resize_in_param.hmax = gHmax;
resize_in_param.hmin = gHmin;
resize_in_param.vmax = gVmax;
resize_in_param.vmin = gVmin;
```



```
resize_in_param.dest_width = floor(gHinc * (gHmax - gHmin + 1) + 0.5);
resize_in_param.dest_high = floor(gVinc * (gVmax - gVmin + 1) + 0.5);
dvppApiCtlMsg.in = (void *)(&resize_in_param);
dvppApiCtlMsg.out = (void *)(&resize_out_param);
IDVPPAPI *pidvppapi = NULL;
CreateDvppApi(pidvppapi);
if (0 != DvppCtl(pidvppapi, DVPP_CTL_TOOL_CASE_GET_RESIZE_PARAM, &dvppApiCtlMsg))
{
  printf("call dvppctl process faild!\n");
  DestroyDvppApi(pidvppapi);
  return;
}
int bufferSize = 0;
vpcInMsg.format = gFormat;
vpcInMsg.cvdr_or_rdma = gcvdr_or_rdma;
vpcInMsg.bitwidth = gBitwidth;
vpcInMsg.rank = gRank;
vpcInMsg.width = gWidth;
vpcInMsg.high = gHigh;
vpcInMsg.stride = gStride;
vpcInMsg.hmax = resize_out_param.hmax;
vpcInMsg.hmin = resize_out_param.hmin;
vpcInMsg.vmax = resize_out_param.vmax;
vpcInMsg.vmin = resize_out_param.vmin;
vpcInMsg.vinc = resize_out_param.vinc;
vpcInMsg.hinc = resize_out_param.hinc;
```

5.3 CMDLIST Functions and Parameters

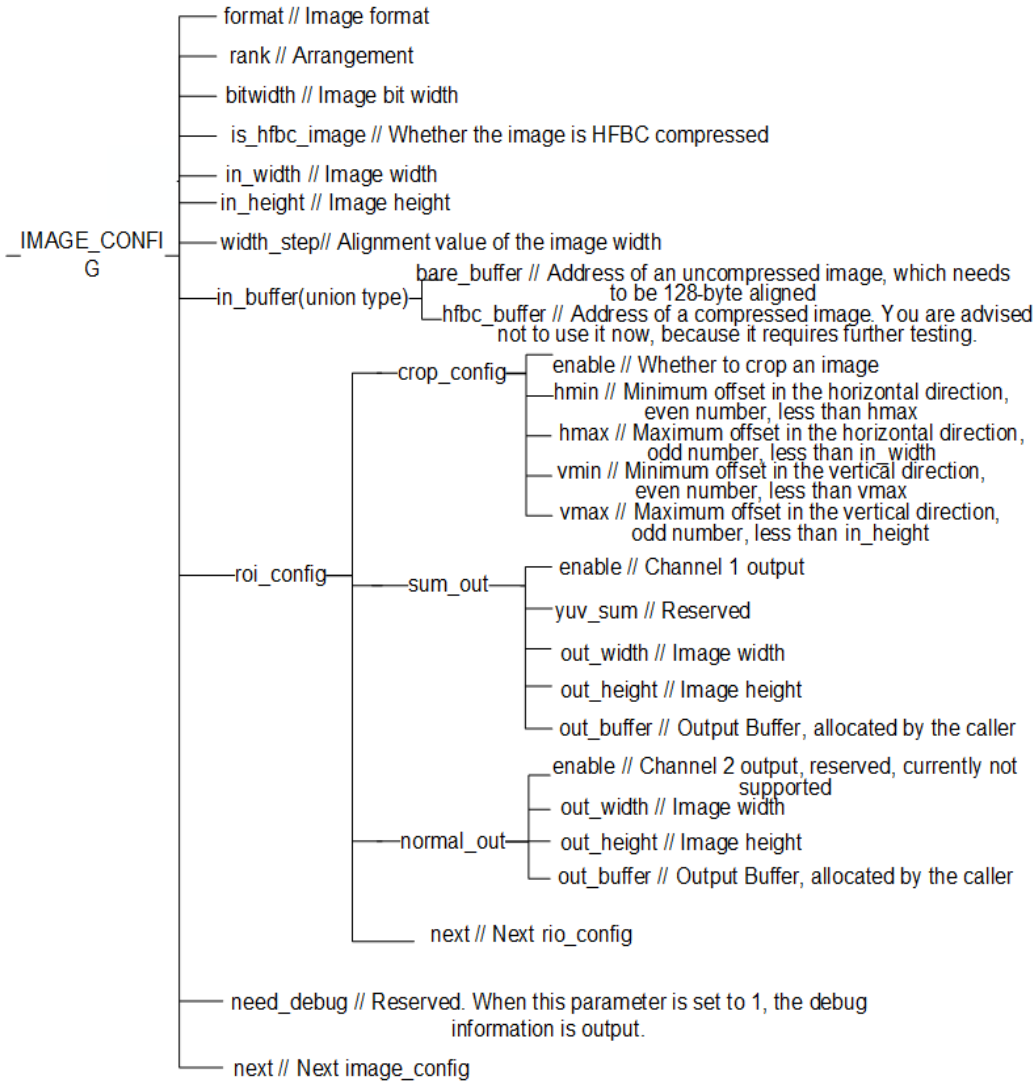
You are not advised to use the CMDLIST function, which will be removed in later versions. The VPC and CMDLIST functions in the earlier version have been combined into the VPC function. For details, see **2.2.2 VPC Parameters**.

The CMDLIST is an extended function of the VPC. It combines functions that need to start the VPC for multiple times into one startup-complete-interrupt-return process. CMDLIST applies to scenarios that do not require low latency and process a large number of low-resolution images.

Input Parameter

The CMDLIST input is a structure. For details about the parameters, see **Figure 5-1** and **Table 5-2**.

Figure 5-1 CMDLIST structures



For details about the description and value ranges, see **Table 5-2**.

Table 5-2 Structure description

Member Variable	Description	Value Range
unsigned int format	Input image type	typedef enum { yuv420_semi_plannar =0,//0 yuv422_semi_plannar,//1 yuv444_semi_plannar,//2 yuv422_packed,//3 yuv444_packed,//4 rgb888_packed,//5 xrgb8888_packed,//6 yuv400_semi_plannar,//7 invalid_image_type,//20 }Img_Type;
unsigned int rank	Output image format (NV12 or NV21)	For details, see **Table 5-1**.
unsigned int bitwidth	Bit depth, which is usually 8 bits. The 10-bit depth is used only when the image format is YUV/YVU420 Semi-Planar and HFBC compression is used.	**8**: 8 bits **10**: 10 bits
int is_hfbc_image	Input image channel. Generally, this parameter is set to **1** and the CVDR channel is used. The RDMA channel is used only when the HFBC data output by the VDEC is used as the input.	**0**: RDMA **1**: CVDR
unsigned int in_width	Width of the input image, which must be 128-pixel aligned	128~4096
unsigned int in_height	Height of the input image, which must be 16-pixel aligned	16~4096

Member Variable	Description	Value Range
unsigned int width_step	Image stride	YUV400SP, YUV420SP, YUV422SP, and YUV444SP: Align the width to a 128-pixel boundary. YUV422 Packed: Align (width x 2) to a 128-pixel boundary. YUV444 Packed and RGB888: Align (width x 3) to a 128-pixel boundary. XRGB8888: Align (width x 4) to a 128-pixel boundary.
unsigned int need_debug	Reserved for internal debugging. Generally, this parameter is set to **0**.	**0**: normal mode **1**: debug mode

CMDLIST_IN_BUFFER in_buffer	Pointer to the address of the input image buffer, which is a union. For common uncompressed images, use **bare_buffer**, which must be 128-byte aligned. For HFBC-compressed images, use **hfbc_buffer**.	union CMDLIST_IN_BUFFER
		{
		char* bare_buffer;
		RDMACHANNEL* hfbc_buffer;
		};
		RDMACHANNEL in this syntax is the same as that in **vpc_in_msg**.
		For details, see **7.5.1 RDMACHANNEL Structure**.

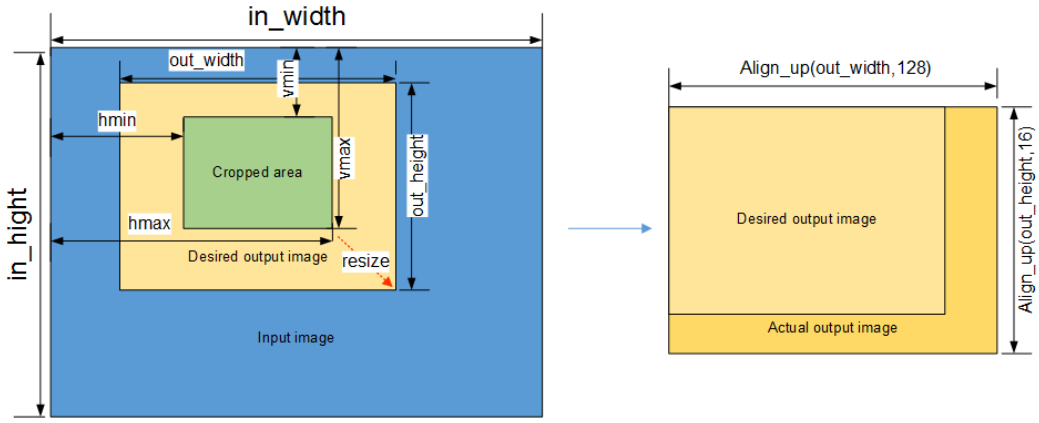
ROI_CONFIG roi_config	Structure of output parameter configuration	For details, see **ROI_CONFIG structure**.
IMAGE_CONFIG* next	Pointer to the next **IMAGE_CONFIG** structure. Set this parameter when multiple images are used. Otherwise, set this parameter to **NULL**.	--

NOTE

For details about the structure syntax, see **include/inc/dvpp/dvpp_config.h** in the DDK page.

Figure 5-2 shows the information about the input parameters

Figure 5-2 Example of input parameters



This figure shows the normalization of the cropping and resizing operations.

- When only cropping is performed, the resizing coefficient is 1.
- When only resizing is performed, the cropping size is the original image.

NOTE

- The width of the input image is 128-pixel aligned, the height of the input image is 16-pixel aligned, and the maximum resolution of the input image and that of the output image are both 4 KB.
- The following interfaces are required for buffer address alignment:

  The buffer addresses of the input and output images must be both 128-byte aligned. The addresses must be in the same 4 GB space. **HIAI_DVPP_DMalloc** is required for applying for huge page memory.

  The memory application API is **HIAI_DVPP_DMalloc(size)**
- The resizing ratio calculated based on the output image size and the input image size must be within the range of [0.03125, 4].
- The output buffer size must be calculated based on the output image resolution after the width is 128-pixel aligned and the height is 16-pixel aligned.
- The CMDLIST API supports a maximum of 32 images and each image supports a maximum of 256 ROIs.

Output Parameter

None. The output image is in **out_buffer** of **sum_out**. You can obtain the image by reading the buffer.

Table 5-2 shows the rank configuration table of the input and output image formats.

Calling Example

```
/**************************************************************************************/
/***** Example description: ***********************************************************
This example uses a 1920 x 1088 YUV420 NV12 image as the input.
File name of the input image: "file1_1920x1088_nv12.yuv"
Operations on the input image: Crop five sub-images and resize them to 224 x 224.
 ***************************************************************************************
 **************************************************************************************/
int main()
{
```



```
    int ret = 0;
   // Reads the input file.
    char image_file[128] = "file1_1920x1088_nv12.yuv";
    ifstream in_stream(image_file);
    if (!in_stream.is_open()) {
        printf("can not open %s.\n", image_file);
        return -1;
    }
    in_stream.seekg(0, ios::end);
    int file_len = in_stream.tellg();
    char* in_buffer = (char *)HIAI_DVPP_DMalloc(file_len);
    in_stream.seekg(0, ios::beg);
    in_stream.read(in_buffer, file_len);
    in_stream.close();
    // Starts to add the configuration of the first image.
    IMAGE_CONFIG* image_config = (IMAGE_CONFIG*)malloc(sizeof(IMAGE_CONFIG));
    image_config->in_buffer.bare_buffer = in_buffer; // Currently, all used images are uncompressed.
    image_config->format = 0;
    image_config->rank = 1;// When the input is in NV12 format, the output is also in NV12 format. Set
rank to 1.
    image_config->bitwidth = 8;// Currently, only 8-bit units are used.
    image_config->in_width = 1920;
    image_config->in_height = 1088;
    image_config->width_step = 1920;// The value is the same as the width.
    // Start to add the configuration of the first cropped image. In this example, the cropped area is enclosed
by (0,0) and (511,511).
    ROI_CONFIG* roi_config = &image_config->roi_config;
   // Starts to configure the cropping parameters.
    roi_config->crop_config.enable = 1; // Note: If only resizing is required, set this parameter to 0.
    roi_config->crop_config.hmin = 0;
    roi_config->crop_config.hmax = 511;
    roi_config->crop_config.vmin = 0;
    roi_config->crop_config.vmax = 511;
   // Starts to configure output channel parameters.
    roi_config->sum_out.enable = 1; // Enables the output of the first channel.
    roi_config->sum_out.out_width = 224;
    roi_config->sum_out.out_height = 224;
    int out_buffer_size = AlignUp(224,128)*AlignUp(224,16)*3/2;
    roi_config->sum_out.out_buffer = (char*)HIAI_DVPP_DMalloc(out_buffer_size);
    ROI_CONFIG* last_roi = roi_config;
   // Starts to add the configurations of the second to fifth cropped images.
    for (int i = 0 ; i < 4; i++) {
        ROI_CONFIG* roi_config = (ROI_CONFIG*)malloc(sizeof(ROI_CONFIG));
        // Starts to configure the cropping parameters.
        roi_config->crop_config.enable = 1;
        roi_config->crop_config.hmin = 100*i;
        roi_config->crop_config.hmax = 299 + 100*i;
        roi_config->crop_config.vmin = 100*i;
        roi_config->crop_config.vmax = 299 + 100*i;
        // Starts to configure output channel parameters.
        roi_config->sum_out.enable = 1;
        roi_config->sum_out.out_width = 224;
        roi_config->sum_out.out_height = 224;
        out_buffer_size = AlignUp(224,128)*AlignUp(224,16)*3/2;
        roi_config->sum_out.out_buffer = (char*)HIAI_DVPP_DMalloc(out_buffer_size);
        roi_config->next = nullptr;
        last_roi->next = roi_config;
        last_roi = roi_config;
    }
    // Starts to call the CMDLIST API of the DVPP.
    IDVPPAPI *pidvppapi = NULL;
    // Calls the createdvppapi API only once, regardless of the subsequent callings.
    ret = CreateDvppApi(pidvppapi);
    if (ret != 0) {
        printf("creat dvpp api faild!\n");
        return -1;
    }
    dvppapi_ctl_msg dvppApiCtlMsg;
    dvppApiCtlMsg.in = (void *)(image_config);
```



```
    dvppApiCtlMsg.in_size = sizeof(IMAGE_CONFIG);
    ret = DvppCtl(pidvppapi, DVPP_CTL_CMDLIST_PROC, &dvppApiCtlMsg);
    if (0 != ret) {
        printf("call cmdlist dvppctl process faild!\n");
    } else {
        printf("cmdlist success.\n");
    }
    ret += DestroyDvppApi(pidvppapi);
    roi_config = image_config->roi_config.next;
    while (roi_config != nullptr) {
        ROI_CONFIG* next_roi = roi_config->next;
        UNMAP(roi_config->sum_out.out_buffer, out_buffer_size);
        free(roi_config);
        roi_config = next_roi;
    }
    UNMAP(image_config->roi_config.sum_out.out_buffer, out_buffer_size);
    free(image_config);
    UNMAP(in_buffer, file_len);
    return ret;
}
```

5.4 VENC Functions and Parameters

You are not advised to use the VENC function of the earlier version, which will be removed in later versions. The VENC function in the earlier version has been migrated to new VENC APIs. For details, see **4 VENC Function Interfaces**.

Function

The VENC module is used to encode YUV420 and YVU420 image data.

- The VENC supports the following input formats:
  YUV420 semi-planner NV12/NV21-8bit
- The VENC supports the following output formats:
  H264 BP/MP/HP
  H265 MP

VENC Performance Specifications

Scenario	Total Frame Rate
1080p, 1 channel (multi-channel is not supported)	30fps

Input Parameter: venc_in_msg

Member Variable	Description	Value Range
Int width	Image width	The value is an even number ranging from 128 to 1920.

Member Variable	Description	Value Range
Int height	Image height	The value is an even number ranging from 128 to 1920.
Int coding_type	Video encoding protocol	0~3 • **0**: H.265 MP • **1**: H.264 BP • **2**: H.264 MP • **3**: H.264 HP
Int YUV_store_type	YUV image storage format	The value is **0** or **1**. • **0**: YUV420 Semi-Planar • **1**: YVU420 Semi-Planar
char* input_data	Address of the input image data	The value cannot be null.
Int input_data_size	Size of the input image data	The value is a positive number.
shared_ptr<AutoBuffer> output_data_queue	Address for output encoded stream	The buffer needs to be configured. This parameter is a smart pointer transferred into the DVPP, used for the caller to allocate the output buffer. You can read this buffer to obtain the output image. This parameter is mandatory.

Output Parameter

None

Calling Example

```
void TEST_3() //venc demo
{
    int read_file_size;
    int unit_file_size;
    FILE *fp = fopen(in_file_name, "rb");
    if (fp == NULL) {
        printf("open file: %s failed.\n", in_file_name);
        return;
    }
    printf("open yuv success \n");
    fseek(fp, 0L, SEEK_END);
    int file_size = ftell(fp);
```

```
    fseek(fp, 0L, SEEK_SET);

    venc_in_msg venc_msg;
    venc_msg.width = gWidth;
    venc_msg.height = gHigh;
    venc_msg.coding_type = gFormat;
    venc_msg.YUV_store_type = gBitwidth;
    venc_msg.output_data_queue = make_shared<AutoBuffer>();
    unit_file_size = gWidth * gHigh * 3 / 2 * MAX_FRAME_NUM_VENC; // The size of a single file is 16 frames.
    dvppapi_ctl_msg dvppApiCtlMsg;
    dvppApiCtlMsg.in = (void *)(&venc_msg);
    dvppApiCtlMsg.in_size = sizeof(venc_in_msg);
    IDVPPAPI *pidvppapi = NULL;
    CreateDvppApi(pidvppapi);
    char out_filename[] = "venc.bin";
    FILE *outputBufferFile;
    outputBufferFile = fopen (out_filename, "wb+");

    do{
       read_file_size = file_size>unit_file_size? unit_file_size:file_size; // The size of the file to be read each
time cannot exceed 16 frames.
        venc_msg.input_data = (char *)malloc(read_file_size);
        int read_len = fread(venc_msg.input_data, 1, read_file_size, fp);
        printf("file size is %d,read len is %d.\n", read_file_size, read_len);
        venc_msg.input_data_size = read_len;

        if (pidvppapi != NULL) {
            if (DvppCtl(pidvppapi, DVPP_CTL_VENC_PROC, &dvppApiCtlMsg) != 0) {
                printf("call dvppctl process faild!\n");
                DestroyDvppApi(pidvppapi);
                fclose(fp);
                fclose(outputBufferFile);
                return;
            }
            if (venc_msg.output_data_queue->getBufferSize() > 100) {// This is used to ensure that the
encoding result does not contain only the header information.
                char* out_buf = venc_msg.output_data_queue->getBuffer();
                int out_buf_size = venc_msg.output_data_queue->getBufferSize();
                int write_size = fwrite(out_buf, 1, out_buf_size, outputBufferFile);
                fflush(outputBufferFile);
            } else {
                printf("venc output data is too small : %d \n", venc_msg.output_data_queue->getBufferSize());
            }

        } else {
            printf("pidvppapi is null!\n");
        }

        if (venc_msg.input_data != NULL) {
            free(venc_msg.input_data);
            venc_msg.input_data = NULL;
        }

        file_size = file_size - unit_file_size;
    } while (file_size > 0);

    DestroyDvppApi(pidvppapi);
    fclose(outputBufferFile);
    fclose(fp);
    return;
}
```

6 Calling Example

6.1 Implementing the VPC Function

Concepts

For details about the concepts such as cropping, resizing, overlaying, top offset, bottom offset, left offset, and right offset, see **1.3 VPC Function**.

Example 1: Resizing of the Original Image

Set key parameters as follows:

- The width and height of the cropping area are the same as the actual width and height of the input image. The offset values of the cropped area are set as follows:
  - leftOffset = 0
  - upOffset = 0
  - rightOffset – leftOffset + 1 = Actual width of the input image
  - downOffset – upOffset + 1 = Actual height of the input image
- The width and height of the overwritten area are the width and height of the resized image. You can specify the position of the overwritten area. For example:

  If the specified overwritten area is in the upper left corner of the output image, the offset values of the overwritten area are set as follows:

  - leftOffset = 0
  - upOffset = 0
  - rightOffset – leftOffset + 1 = Width of the resized image
  - downOffset – upOffset + 1 = Height of the resized image
- For original 8K image resizing, **inputFormat** and **outputFormat** must be set to **INPUT_YUV420_SEMI_PLANNER_UV** and **OUTPUT_YUV420SP_UV**, respectively, or **INPUT_YUV420_SEMI_PLANNER_VU** and **OUTPUT_YUV420SP_VU**, respectively. For original non-8K image resizing, set **inputFormat** and **outputFormat** by referring to **Table 2-1**.

Sample Code:

In this sample, a YUV420SP image is scaled from 1080p to 720p.

```
void NewVpcTest1()
{
    uint32_t inWidthStride = 1920;
    uint32_t inHeightStride = 1080;
    uint32_t outWidthStride = 1280;
    uint32_t outHeightStride = 720;
    uint32_t inBufferSize = inWidthStride * inHeightStride * 3 / 2; // 1080P yuv420sp Image
    uint32_t outBufferSize = outWidthStride * outHeightStride * 3 / 2; // 720P yuv420sp bImage
    uint8_t* inBuffer  = static_cast<uint8_t*>(HIAI_DVPP_DMalloc(inBufferSize)); // Construct an input picture.
    if (inBuffer == nullptr) {
        HIAI_ENGINE_LOG(HIAI_VPC_CTL_ERROR, "can not alloc input buffer");
        return;
    }
    uint8_t* outBuffer = static_cast<uint8_t*>(HIAI_DVPP_DMalloc(outBufferSize)); // Construct an output
picture.
    if (outBuffer == nullptr) {
        HIAI_ENGINE_LOG(HIAI_VPC_CTL_ERROR, "can not alloc output buffer");
        HIAI_DVPP_DFree(inBuffer);
        inBuffer = nullptr;
        return;
    }

    FILE* fp = fopen("dvpp_vpc_1920x1080_nv12.yuv", "rb+");
    if (fp == nullptr) {
        HIAI_ENGINE_LOG(HIAI_ERROR, "fopen file failed");
        DFreeInOutBuffer(inBuffer, outBuffer);
        return;
    }

    fread(inBuffer, 1, inBufferSize, fp);
    fclose(fp);
    fp = nullptr;
    // Construct the input picture configuration.
    std::shared_ptr<VpcUserImageConfigure> imageConfigure(new VpcUserImageConfigure);
    imageConfigure->bareDataAddr = inBuffer;
    imageConfigure->bareDataBufferSize = inBufferSize;
    imageConfigure->widthStride = inWidthStride;
    imageConfigure->heightStride = inHeightStride;
    imageConfigure->inputFormat = INPUT_YUV420_SEMI_PLANNER_UV;
    imageConfigure->outputFormat = OUTPUT_YUV420SP_UV;
    imageConfigure->yuvSumEnable = false;
    imageConfigure->cmdListBufferAddr = nullptr;
    imageConfigure->cmdListBufferSize = 0;
    std::shared_ptr<VpcUserRoiConfigure> roiConfigure(new VpcUserRoiConfigure);
    roiConfigure->next = nullptr;
    VpcUserRoiInputConfigure* inputConfigure = &roiConfigure->inputConfigure;
    // Set the drawing area, [0,0] in the upper left corner of the area,
    // and [1919,1079] in the lower right corner.
    inputConfigure->cropArea.leftOffset  = 0;
    inputConfigure->cropArea.rightOffset = inWidthStride - 1;
    inputConfigure->cropArea.upOffset    = 0;
```



```
    inputConfigure->cropArea.downOffset  = inHeightStride - 1;
    VpcUserRoiOutputConfigure* outputConfigure = &roiConfigure->outputConfigure;
    outputConfigure->addr = outBuffer;
    outputConfigure->bufferSize = outBufferSize;
    outputConfigure->widthStride = outWidthStride;
    outputConfigure->heightStride = outHeightStride;
    // Set the map area, coordinate [0,0] in the upper left corner of the map area,
    // and [1279,719] in the lower right corner.
    outputConfigure->outputArea.leftOffset  = 0;
    outputConfigure->outputArea.rightOffset = outWidthStride - 1;
    outputConfigure->outputArea.upOffset    = 0;
    outputConfigure->outputArea.downOffset  = outHeightStride - 1;

    imageConfigure->roiConfigure = roiConfigure.get();

    IDVPPAPI *pidvppapi = nullptr;
    int32_t ret = CreateDvppApi(pidvppapi);
    if (ret != 0) {
        HIAI_ENGINE_LOG(HIAI_CREATE_DVPP_ERROR, "create dvpp api fail.");
        DFreeInOutBuffer(inBuffer, outBuffer);
        return;
    }
    dvppapi_ctl_msg dvppApiCtlMsg;
    dvppApiCtlMsg.in = static_cast<void*>(imageConfigure.get());
    dvppApiCtlMsg.in_size = sizeof(VpcUserImageConfigure);
    ret = DvppCtl(pidvppapi, DVPP_CTL_VPC_PROC, &dvppApiCtlMsg);
    if (ret != 0) {
        HIAI_ENGINE_LOG(HIAI_VPC_CTL_ERROR, "call vpc dvppctl process faild!");
        ret = DestroyDvppApi(pidvppapi);
        DFreeInOutBuffer(inBuffer, outBuffer);
        return;
    } else {
        HIAI_ENGINE_LOG(HIAI_OK, "NewVpcTest1::call vpc dvppctl process success!");
    }

    FILE* outImageFp = fopen("NewVpcTest1Out.yuv", "wb+");
    if (outImageFp == nullptr) {
        HIAI_ENGINE_LOG(HIAI_VPC_CTL_ERROR, "open NewVpcTest1Out.yuv faild!");
        ret = DestroyDvppApi(pidvppapi);
        DFreeInOutBuffer(inBuffer, outBuffer);
        return;
    }

    fwrite(outBuffer, 1, outBufferSize, outImageFp);

    ret = DestroyDvppApi(pidvppapi);
    DFreeInOutBuffer(inBuffer, outBuffer);
    fclose(outImageFp);
    outImageFp = nullptr;
    return;
}
```

Example 2: Cropping and Resizing of a Single Image

Set key parameters as follows:

- The width and height of the cropped area are the width and height of the image before resizing. You can specify the position of the cropped area. For example:

  If the specified cropped area is in the middle of the input image, the offset values of the cropped area are set as follows:

  - leftOffset = 100
  - rightOffset = 499
  - upOffset = 100

  - downOffset = 399
  - rightOffset – leftOffset + 1 = Width of the cropped area
  - downOffset – upOffset + 1 = Height of the cropped area
- The width and height of the overwritten area are the width and height of the resized image. You can specify the position of the overwritten area. For example:

  If the specified overwritten area is in the middle of the output image, the offset values of the overwritten area are set as follows:
  - leftOffset = 256
  - rightOffset = 399
  - upOffset = 200
  - downOffset = 399
  - rightOffset – leftOffset + 1 = Width of the resized image
  - downOffset – upOffset + 1 = Height of the resized image

Sample Code:

In this sample, an image is cropped out of a YUV420SP 1080p image and resized. Then, it overwrites a 720p image (which is a blank image generated by an empty output buffer allocated by the user.) If you want to use an existing image as the output image, read the image to the output buffer after it is allocated. For details about the code example, see **Example 5: Overlaying**.

```
void NewVpcTest2()
{
    uint32_t inWidthStride = 1920;
    uint32_t inHeightStride = 1080;
    uint32_t outWidthStride = 1280;
    uint32_t outHeightStride = 720;
    uint32_t inBufferSize = inWidthStride * inHeightStride * 3 / 2; // 1080P yuv420sp Image
    uint32_t outBufferSize = outWidthStride * outHeightStride * 3 / 2; // 720P yuv420sp Image
    uint8_t* inBuffer  = static_cast<uint8_t*>(HIAI_DVPP_DMalloc(inBufferSize)); //  Construct an input
picture.
    if (inBuffer == nullptr) {
        HIAI_ENGINE_LOG(HIAI_VPC_CTL_ERROR, "can not alloc input buffer");
        return;
    }
    uint8_t* outBuffer = static_cast<uint8_t*>(HIAI_DVPP_DMalloc(outBufferSize)); //  Construct an output
picture.
    if (outBuffer == nullptr) {
        HIAI_ENGINE_LOG(HIAI_VPC_CTL_ERROR, "can not alloc output buffer");
        HIAI_DVPP_DFree(inBuffer);
        inBuffer = nullptr;
        return;
    }
    (void)memset_s(outBuffer, outBufferSize, 0x80, outBufferSize);

    FILE* fp = fopen("dvpp_vpc_1920x1080_nv12.yuv", "rb+");
    if (fp == nullptr) {
        HIAI_ENGINE_LOG(HIAI_ERROR, "fopen file failed");
        DFreeInOutBuffer(inBuffer, outBuffer);
        return;
    }

    fread(inBuffer, 1, inBufferSize, fp);
    fclose(fp);
    fp = nullptr;
    // Construct the input picture configuration
    std::shared_ptr<VpcUserImageConfigure> imageConfigure(new VpcUserImageConfigure);
    imageConfigure->bareDataAddr = inBuffer;
```



```
    imageConfigure->bareDataBufferSize = inBufferSize;
    imageConfigure->widthStride = inWidthStride;
    imageConfigure->heightStride = inHeightStride;
    imageConfigure->inputFormat = INPUT_YUV420_SEMI_PLANNER_UV;
    imageConfigure->outputFormat = OUTPUT_YUV420SP_UV;
    imageConfigure->yuvSumEnable = false;
    imageConfigure->cmdListBufferAddr = nullptr;
    imageConfigure->cmdListBufferSize = 0;
    std::shared_ptr<VpcUserRoiConfigure> roiConfigure(new VpcUserRoiConfigure);
    roiConfigure->next = nullptr;
    VpcUserRoiInputConfigure* inputConfigure = &roiConfigure->inputConfigure;
    // Set the drawing area, [100,100] in the upper left corner of the area, and [499,499] in the lower right
corner.
    inputConfigure->cropArea.leftOffset  = 100;
    inputConfigure->cropArea.rightOffset = 499;
    inputConfigure->cropArea.upOffset    = 100;
    inputConfigure->cropArea.downOffset  = 499;
    VpcUserRoiOutputConfigure* outputConfigure = &roiConfigure->outputConfigure;
    outputConfigure->addr = outBuffer;
    outputConfigure->bufferSize = outBufferSize;
    outputConfigure->widthStride = outWidthStride;
    outputConfigure->heightStride = outHeightStride;
    // Set the map area, [256,200] in the upper left corner of the map area, and [399,399] in the lower right
corner.
    outputConfigure->outputArea.leftOffset  = 256; // The offset value must be 16-pixel-aligned.
    outputConfigure->outputArea.rightOffset = 399;
    outputConfigure->outputArea.upOffset    = 200;
    outputConfigure->outputArea.downOffset  = 399;

    imageConfigure->roiConfigure = roiConfigure.get();

    IDVPPAPI *pidvppapi = nullptr;
    int32_t ret = CreateDvppApi(pidvppapi);
    if (ret != 0) {
        HIAI_ENGINE_LOG(HIAI_CREATE_DVPP_ERROR, "create dvpp api fail.");
        DFreeInOutBuffer(inBuffer, outBuffer);
        return;
    }
    dvppapi_ctl_msg dvppApiCtlMsg;
    dvppApiCtlMsg.in = static_cast<void*>(imageConfigure.get());
    dvppApiCtlMsg.in_size = sizeof(VpcUserImageConfigure);
    ret = DvppCtl(pidvppapi, DVPP_CTL_VPC_PROC, &dvppApiCtlMsg);
    if (ret != 0) {
        HIAI_ENGINE_LOG(HIAI_VPC_CTL_ERROR, "call vpc dvppctl process faild!");
        ret = DestroyDvppApi(pidvppapi);
        DFreeInOutBuffer(inBuffer, outBuffer);
        return;
    } else {
        HIAI_ENGINE_LOG(HIAI_OK, "NewVpcTest2::call vpc dvppctl process success!");
    }

    FILE* outImageFp = fopen("NewVpcTest2Out.yuv", "wb+");
    if (outImageFp == nullptr) {
        HIAI_ENGINE_LOG(HIAI_ERROR, "open NewVpcTest2Out.yuv failed");
        DestroyDvppApi(pidvppapi);
        DFreeInOutBuffer(inBuffer, outBuffer);
        return;
    }
    fwrite(outBuffer, 1, outBufferSize, outImageFp);

    ret = DestroyDvppApi(pidvppapi);
    DFreeInOutBuffer(inBuffer, outBuffer);
    fclose(outImageFp);
    outImageFp = nullptr;
    return;
}
```


Example 3: Cropping, Resizing, and Stitching of Multiple Images

Set key parameters as follows: For details about how to configure the top offset, bottom offset, left offset, and right offset of the cropped area or overwritten area, see **Example 2: Cropping and Resizing of a Single Image**.

Sample code: In this sample, multiple images are cropped out of a YUV420SP 1080p image. Then, the images are resized and stitched onto a 720p output image (which is a blank image generated by the empty output buffer allocated by the user). The position of the overwritten area is determined by the top offset, bottom offset, left offset, and right offset. If you want to use an existing image as the output image, read the image to the output buffer after it is allocated. For details about the code example, see **Example 5: Overlaying**.

```
void NewVpcTest3()
{
    uint32_t inWidthStride = 1920;
    uint32_t inHeightStride = 1080;
    uint32_t outWidthStride = 1280;
    uint32_t outHeightStride = 720;
    uint32_t inBufferSize = inWidthStride * inHeightStride * 3 / 2; // 1080P yuv420sp Image
    uint32_t outBufferSize = outWidthStride * outHeightStride * 3 / 2; // 720P yuv420sp Image
    uint8_t* inBuffer = static_cast<uint8_t*>(HIAI_DVPP_DMalloc(inBufferSize)); // Construct an input picture.
    if (inBuffer == nullptr) {
        HIAI_ENGINE_LOG(HIAI_VPC_CTL_ERROR, "can not alloc input buffer");
        return;
    }

    FILE* fp = fopen("dvpp_vpc_1920x1080_nv12.yuv", "rb+");
    if (fp == nullptr) {
        HIAI_ENGINE_LOG(HIAI_ERROR, "fopen file failed");
        HIAI_DVPP_DFree(inBuffer);
        inBuffer = nullptr;
        return;
    }

    fread(inBuffer, 1, inBufferSize, fp);
    fclose(fp);
    fp == nullptr;
    // Construct the input picture configuration.
    std::shared_ptr<VpcUserImageConfigure> imageConfigure(new VpcUserImageConfigure);
    imageConfigure->bareDataAddr = inBuffer;
    imageConfigure->bareDataBufferSize = inBufferSize;
    imageConfigure->widthStride = inWidthStride;
    imageConfigure->heightStride = inHeightStride;
    imageConfigure->inputFormat = INPUT_YUV420_SEMI_PLANNER_UV;
    imageConfigure->outputFormat = OUTPUT_YUV420SP_UV;
    imageConfigure->yuvSumEnable = false;
    imageConfigure->cmdListBufferAddr = nullptr;
    imageConfigure->cmdListBufferSize = 0;
    std::shared_ptr<VpcUserRoiConfigure> lastRoi;
    std::vector<std::shared_ptr<VpcUserRoiConfigure>> roiVector;
    for (uint32_t i = 0; i < 5; i++) {
        std::shared_ptr<VpcUserRoiConfigure> roiConfigure(new VpcUserRoiConfigure);
        roiVector.push_back(roiConfigure);
        roiConfigure->next = nullptr;
        VpcUserRoiInputConfigure* inputConfigure = &roiConfigure->inputConfigure;
        // Set the drawing area.
        inputConfigure->cropArea.leftOffset  = 100 + i * 16;
        inputConfigure->cropArea.rightOffset = 499 + i * 16;
        inputConfigure->cropArea.upOffset    = 100 + i * 16;
        inputConfigure->cropArea.downOffset  = 499 + i * 16;
        VpcUserRoiOutputConfigure* outputConfigure = &roiConfigure->outputConfigure;
        uint8_t* outBuffer = static_cast<uint8_t*>(HIAI_DVPP_DMalloc(outBufferSize)); // Construct an input
picture
        if (outBuffer == nullptr) {
            HIAI_ENGINE_LOG(HIAI_VPC_CTL_ERROR, "can not alloc output buffer");
```



```
        HIAI_DVPP_DFree(inBuffer);
        inBuffer = nullptr;
        FreeMultipleRio(imageConfigure->roiConfigure);
        return;
    }
    (void)memset_s(outBuffer, outBufferSize, 0x80, outBufferSize);
    outputConfigure->addr = outBuffer;
    outputConfigure->bufferSize = outBufferSize;
    outputConfigure->widthStride = outWidthStride;
    outputConfigure->heightStride = outHeightStride;
    // Set the map area.
    outputConfigure->outputArea.leftOffset  = 256 + i * 16; // The offset value must be 16-pixel-aligned.
    outputConfigure->outputArea.rightOffset = 399 + i * 16;
    outputConfigure->outputArea.upOffset    = 256 + i * 16;
    outputConfigure->outputArea.downOffset  = 399 + i * 16;
    if (i == 0) {
        imageConfigure->roiConfigure = roiConfigure.get();
        lastRoi = roiConfigure;
    } else {
        lastRoi->next = roiConfigure.get();
        lastRoi = roiConfigure;
    }
}

IDVPPAPI *pidvppapi = nullptr;
int32_t ret = CreateDvppApi(pidvppapi);
if (ret != 0) {
    HIAI_ENGINE_LOG(HIAI_CREATE_DVPP_ERROR, "create dvpp api fail.");
    HIAI_DVPP_DFree(inBuffer);
    inBuffer = nullptr;
    FreeMultipleRio(imageConfigure->roiConfigure);
    return;
}
dvppapi_ctl_msg dvppApiCtlMsg;
dvppApiCtlMsg.in = static_cast<void*>(imageConfigure.get());
dvppApiCtlMsg.in_size = sizeof(VpcUserImageConfigure);
ret = DvppCtl(pidvppapi, DVPP_CTL_VPC_PROC, &dvppApiCtlMsg);
if (ret != 0) {
    HIAI_ENGINE_LOG(HIAI_VPC_CTL_ERROR, "call vpc dvppctl process faild!");
    ret = DestroyDvppApi(pidvppapi);
    HIAI_DVPP_DFree(inBuffer);
    inBuffer = nullptr;
    FreeMultipleRio(imageConfigure->roiConfigure);
    return;
} else {
    HIAI_ENGINE_LOG(HIAI_OK, "NewVpcTest3::call vpc dvppctl process success!");
}
FILE* outImageFp = nullptr;
uint32_t imageCount = 0;
char fileName[50] = {0};

while (imageConfigure->roiConfigure != nullptr) {
    int32_t safeFuncRet = sprintf_s(fileName, sizeof(fileName), "NewVpcTest3_%dOut.yuv", imageCount);
    if (safeFuncRet == -1) {
        HIAI_ENGINE_LOG(HIAI_ERROR, "sprintf_s fail, ret = %d", safeFuncRet);
        DestroyDvppApi(pidvppapi);
        HIAI_DVPP_DFree(inBuffer);
        inBuffer = nullptr;
        FreeMultipleRio(imageConfigure->roiConfigure);
        return;
    }
    outImageFp = fopen(fileName, "wb+");
    if (outImageFp == nullptr) {
        HIAI_ENGINE_LOG(HIAI_ERROR, "open %s faild!", fileName);
        DestroyDvppApi(pidvppapi);
        HIAI_DVPP_DFree(inBuffer);
        inBuffer = nullptr;
        FreeMultipleRio(imageConfigure->roiConfigure);
        return;
```



```
        }
        fwrite(imageConfigure->roiConfigure->outputConfigure.addr, 1,
            imageConfigure->roiConfigure->outputConfigure.bufferSize, outImageFp);
        fclose(outImageFp);
        outImageFp = nullptr;
        imageConfigure->roiConfigure = imageConfigure->roiConfigure->next;
        imageCount++;
    }
    ret = DestroyDvppApi(pidvppapi);
    HIAI_DVPP_DFree(inBuffer);
    inBuffer = nullptr;
    FreeMultipleRio(imageConfigure->roiConfigure);
    return;
}
```

Example 4: 8K Image Resizing

8K image resizing: Resizing is supported, the format conversion between YUV420SP NV12 and YUV420SP NV21 is supported, but cropping is not supported.

Sample code: In this sample, a YUV420SP image is scaled from 8129 x 8192 to 4000 x 4000.

```
void NewVpcTest4()
{
    uint32_t inWidthStride = 8192; // No need for 128 byte alignment
    uint32_t inHeightStride = 8192; // No need for 16 byte alignment
    uint32_t outWidthStride = 4000; // No need for 128 byte alignment
    uint32_t outHeightStride = 4000; // No need for 16 byte alignment
    uint32_t inBufferSize = inWidthStride * inHeightStride * 3 / 2;
    uint32_t outBufferSize = outWidthStride * outHeightStride * 3 / 2; // Construct dummy data
    uint8_t* inBuffer = (uint8_t*)HIAI_DVPP_DMalloc(inBufferSize); // Construct input image
    if (inBuffer == nullptr) {
        HIAI_ENGINE_LOG(HIAI_VPC_CTL_ERROR, "can not alloc roiOutput buffer");
        return;
    }
    uint8_t* outBuffer = (uint8_t*)HIAI_DVPP_DMalloc(outBufferSize); // Construct output image
    if (outBuffer == nullptr) {
        HIAI_ENGINE_LOG(HIAI_VPC_CTL_ERROR, "can not alloc roiOutput buffer");
        HIAI_DVPP_DFree(inBuffer);
        inBuffer = nullptr;
        return;
    }

    FILE* fp = fopen("dvpp_vpc_8192x8192_nv12.yuv", "rb+");
    if (fp == nullptr) {
        HIAI_ENGINE_LOG(HIAI_ERROR, "fopen file failed");
        DFreeInOutBuffer(inBuffer, outBuffer);
        return;
    }
    fread(inBuffer, 1, inBufferSize, fp);
    fclose(fp);
    fp = nullptr;
    std::shared_ptr<VpcUserImageConfigure> imageConfigure(new VpcUserImageConfigure);
    imageConfigure->bareDataAddr = inBuffer;
    imageConfigure->bareDataBufferSize = inBufferSize;
    imageConfigure->isCompressData = false;
    imageConfigure->widthStride = inWidthStride;
    imageConfigure->heightStride = inHeightStride;
    imageConfigure->inputFormat = INPUT_YUV420_SEMI_PLANNER_UV;
    imageConfigure->outputFormat = OUTPUT_YUV420SP_UV;
    imageConfigure->yuvSumEnable = false;
    imageConfigure->cmdListBufferAddr = nullptr;
    imageConfigure->cmdListBufferSize = 0;
    std::shared_ptr<VpcUserRoiConfigure> roiConfigure(new VpcUserRoiConfigure);
    roiConfigure->next = nullptr;
    VpcUserRoiInputConfigure* inputConfigure = &roiConfigure->inputConfigure; // Set the roi area
    inputConfigure->cropArea.leftOffset  = 0;
```



```
    inputConfigure->cropArea.rightOffset = inWidthStride - 1;
    inputConfigure->cropArea.upOffset = 0;
    inputConfigure->cropArea.downOffset  = inHeightStride - 1;
    VpcUserRoiOutputConfigure* outputConfigure = &roiConfigure->outputConfigure;
    outputConfigure->addr = outBuffer;
    outputConfigure->bufferSize = outBufferSize;
    outputConfigure->widthStride = outWidthStride;
    outputConfigure->heightStride = outHeightStride; // Set the map area
    outputConfigure->outputArea.leftOffset  = 0;
    outputConfigure->outputArea.rightOffset = outWidthStride - 1;
    outputConfigure->outputArea.upOffset = 0;
    outputConfigure->outputArea.downOffset  = outHeightStride - 1;
    imageConfigure->roiConfigure = roiConfigure.get();
    IDVPPAPI *pidvppapi = nullptr;
    int32_t ret = CreateDvppApi(pidvppapi);
    if (ret != 0) {
        HIAI_ENGINE_LOG(HIAI_CREATE_DVPP_ERROR, "create dvpp api fail.");
        DFreeInOutBuffer(inBuffer, outBuffer);
        return;
    }
    dvppapi_ctl_msg dvppApiCtlMsg;
    dvppApiCtlMsg.in = static_cast<void*>(imageConfigure.get());
    dvppApiCtlMsg.in_size = sizeof(VpcUserImageConfigure);
    ret = DvppCtl(pidvppapi, DVPP_CTL_VPC_PROC, &dvppApiCtlMsg);
    if (ret != 0) {
        HIAI_ENGINE_LOG(HIAI_VPC_CTL_ERROR, "call vpc dvppctl process faild!");
        ret = DestroyDvppApi(pidvppapi);
        DFreeInOutBuffer(inBuffer, outBuffer);
        return;
    } else {
        HIAI_ENGINE_LOG(HIAI_OK, "NewVpcTest4::call vpc dvppctl process success!");
    }

    FILE* outImageFp = fopen("NewVpcTest4Out.yuv", "wb+");
    if (outImageFp == nullptr) {
        HIAI_ENGINE_LOG(HIAI_VPC_CTL_ERROR, "open NewVpcTest4Out.yuv faild!");
        ret = DestroyDvppApi(pidvppapi);
        DFreeInOutBuffer(inBuffer, outBuffer);
        return;
    }
    fwrite(outBuffer, 1, outBufferSize, outImageFp);
    fclose(outImageFp);
    outImageFp = nullptr;
    ret = DestroyDvppApi(pidvppapi);
    DFreeInOutBuffer(inBuffer, outBuffer);
    return;
}
```

Example 5: Overlaying

To read an existing image into the buffer for storing the output image, overlay the overwritten area on the output image. This requires you to write the code logic to read the image to the buffer. After allocating the output buffer by using code **uint8_t* outBuffer = static_cast<uint8_t*>(HIAI_DVPP_DMalloc(outBufferSize))**, add the following code:

```
FILE* fpOut = fopen("vpcOut.yuv", "rb+");
if  (fpOut == nullptr) {
    HIAI_ENGINE_LOG(HIAI_ERROR, "fopen file  failed.");
    fclose(fpOut);
    return;
}
fread(outBuffer, 1,  outBufferSize, fpOut);
fclose(fpOut);
```


6.2 Implementing the JPEGE Function

- The memory size can be obtained by calling **DvppGetOutParameter**. The output memory can be specified and freed by the user. The calling example is as follows:

```
void TEST_JPEGE_CUSTOM_MEMORY()
{
    sJpegeIn  inData;
    sJpegeOut outData;

    inData.width         = g_width;
    inData.height        = g_high;
    inData.heightAligned = g_high; // no need to align
    inData.format        = (eEncodeFormat)g_format;
    inData.level         = 100;

    inData.stride  = ALIGN_UP(inData.width * 2, 16);
    inData.bufSize = inData.stride * inData.heightAligned;
    if (JPGENC_FORMAT_YUV420 == (inData.format & JPGENC_FORMAT_BIT)) {
        inData.stride  = ALIGN_UP(inData.width, 16);
        inData.bufSize = inData.stride * inData.heightAligned * 3 / 2;
    }

    void* addrOrig = HIAI_DVPP_DMalloc(inData.bufSize);
    if (addrOrig == nullptr) {
        HIAI_ENGINE_LOG(HIAI_JPEGE_CTL_ERROR, "can not alloc input buffer");
        return;
    }
    inData.buf = reinterpret_cast<unsigned char*>(addrOrig);

    unsigned char* tmpAddr = nullptr;

    do {
        // load img file
        FILE* fpIn = fopen(g_inFileName, "rb");
        if (nullptr == fpIn) {
            HIAI_ENGINE_LOG(HIAI_OPEN_FILE_ERROR, "can not open input file");
            break;
        }
        // only copy valid image data, other part is pending
        if (JPGENC_FORMAT_YUV420 == (inData.format & JPGENC_FORMAT_BIT)) {
            // for yuv420semi-planar format, like nv12 / nv21
            HIAI_ENGINE_LOG("input yuv 420 data");
            // for y data
            for (uint32_t j = 0; j < inData.height; j++) {
                fread(inData.buf + j * inData.stride, 1, inData.width, fpIn);
            }
            // for uv data
            for (uint32_t j = inData.heightAligned; j < inData.heightAligned + inData.height / 2; j++) {
                fread(inData.buf + j * inData.stride, 1, inData.width, fpIn);
            }
        } else {
            // for yuv422packed format, like uyvy / vyuy / yuyv / yvyu
            HIAI_ENGINE_LOG("input yuv 422 data");
            for (uint32_t j = 0; j < inData.height; j++) {
                fread(inData.buf + j * inData.stride, 1, inData.width * 2, fpIn);
            }
        }
        fclose(fpIn);
        fpIn = nullptr;

        int32_t ret = DvppGetOutParameter((void*)(&inData), (void*)(&outData),
GET_JPEGE_OUT_PARAMETER);
        if (ret != 0) {
            HIAI_ENGINE_LOG(HIAI_JPEGE_CTL_ERROR, "call DvppGetOutParameter process failed");
            break;
```



```
        }
        // The obtained jpgSize is an estimated value. The actual data length may be less than the
estimated value.
        // After the DvppCtl API is called, the jpgSize value is the actual value after encoding.
        HIAI_ENGINE_LOG("outdata size is %d", outData.jpgSize);
        tmpAddr = (unsigned char*)HIAI_DVPP_DMalloc(outData.jpgSize);
        if (tmpAddr == nullptr) {
            HIAI_ENGINE_LOG(HIAI_JPEGE_CTL_ERROR, "can not alloc output buffer");
            break;
        }
        uint32_t size = outData.jpgSize;

        // call jpege process
        dvppapi_ctl_msg dvppApiCtlMsg;
        dvppApiCtlMsg.in = (void *)&inData;
        dvppApiCtlMsg.in_size = sizeof(inData);
        dvppApiCtlMsg.out = (void *)&outData;
        dvppApiCtlMsg.out_size = sizeof(outData);

        if(!IsHandleJpegeCtlSucc(dvppApiCtlMsg, tmpAddr, size, &outData)) {
            break;
        }
    } while (0); // for resource inData.buf
    if (addrOrig != nullptr) {
        HIAI_DVPP_DFree(addrOrig);
        addrOrig = nullptr;
    }
    if (tmpAddr != nullptr) {// Note: tmpAddr is used to free the buffer, because address offset occurs
after outData.jpgData is processed by the JPEGE.
        HIAI_DVPP_DFree(tmpAddr);
        tmpAddr = nullptr;
    }
}
```

- If the output buffer is not specified by the user, DVPP allocates the buffer internally. The **cbFree()** callback function should be called to free the buffer. The calling example is as follows:

```
void JpegeProcess()
{
    sJpegeIn  inData;
    sJpegeOut outData;

    inData.width         = g_width;
    inData.height        = g_high;
    inData.heightAligned = g_high; // no need to align
    inData.format        = (eEncodeFormat)g_format;
    inData.level         = 100;

    if (JPGENC_FORMAT_YUV420 == (inData.format & JPGENC_FORMAT_BIT)) {
        inData.stride  = ALIGN_UP(inData.width, 16);
        inData.bufSize = inData.stride * inData.heightAligned * 3 / 2;
    } else {
        inData.stride  = ALIGN_UP(inData.width * 2, 16);
        inData.bufSize = inData.stride * inData.heightAligned;
    }

    void* addrOrig = HIAI_DVPP_DMalloc(inData.bufSize);

    if (addrOrig == nullptr) {
        HIAI_ENGINE_LOG(HIAI_JPEGE_CTL_ERROR, "can not alloc input buffer");
        return;
    }
    inData.buf = reinterpret_cast<unsigned char*>(addrOrig);

    do {
        // load img file
        FILE* fpIn = fopen(g_inFileName, "rb");
        if (fpIn == nullptr) {
            HIAI_ENGINE_LOG(HIAI_OPEN_FILE_ERROR, "can not open input file");
```



```
            break;
        }
        // only copy valid image data, other part is pending
        if (JPGENC_FORMAT_YUV420 == (inData.format & JPGENC_FORMAT_BIT)) {
            // for yuv420semi-planar format, like nv12 / nv21
            HIAI_ENGINE_LOG("input yuv 420 data");
            // for y data
            for (uint32_t j = 0; j < inData.height; j++) {
                fread(inData.buf + j * inData.stride, 1, inData.width, fpIn);
            }
            // for uv data
            for (uint32_t j = inData.heightAligned; j < inData.heightAligned + inData.height / 2; j++) {
                fread(inData.buf + j * inData.stride, 1, inData.width, fpIn);
            }
        } else {
            // for yuv422packed format, like uyvy / vyuy / yuyv / yvyu
            HIAI_ENGINE_LOG("input yuv 422 data");
            for (uint32_t j = 0; j < inData.height; j++) {
                fread(inData.buf + j * inData.stride, 1, inData.width * 2, fpIn );
            }

        }
        fclose(fpIn);
        fpIn = nullptr;

        // call jpege process
        dvppapi_ctl_msg dvppApiCtlMsg;
        dvppApiCtlMsg.in = (void*)&inData;
        dvppApiCtlMsg.in_size = sizeof(inData);
        dvppApiCtlMsg.out = (void*)&outData;
        dvppApiCtlMsg.out_size = sizeof(outData);

        IDVPPAPI *pidvppapi = nullptr;
        CreateDvppApi(pidvppapi);
        if (pidvppapi == nullptr) {
            HIAI_ENGINE_LOG(HIAI_JPEGE_CTL_ERROR, "can not open dvppapi engine");
            break;
        }

        JpegeProcessBranch(pidvppapi, dvppApiCtlMsg, outData);

        DestroyDvppApi(pidvppapi);

    } while (0); // for resource inData.buf
    if (addrOrig != nullptr) {
        HIAI_DVPP_DFree(addrOrig);
        addrOrig = nullptr;
    }
}

/*
 * only handle jpege process.
 */
void JpegeProcessBranch(IDVPPAPI*& pidvppapi, dvppapi_ctl_msg& dvppApiCtlMsg, sJpegeOut&
outData)
{
    do {
        for (int i = 0;i < g_loop; i++) { // same picture loop test
            if (DvppCtl(pidvppapi, DVPP_CTL_JPEGE_PROC, &dvppApiCtlMsg)) {
                HIAI_ENGINE_LOG(HIAI_JPEGE_CTL_ERROR, "call jpeg encoder fail");
                break;
            }
            if (i < g_loop - 1) {
                outData.cbFree();
                outData.jpgData = nullptr;
            }
        }
        stringstream outFile;
        outFile << g_outFileName << "_t" << std::this_thread::get_id() << ".jpg";
```

```
        FILE* fpOut = fopen(outFile.str().c_str(), "wb");
        if (fpOut != nullptr) {
            fwrite(outData.jpgData, 1, outData.jpgSize, fpOut);
            fflush(fpOut);
            fclose(fpOut);
            fpOut = nullptr;
        } else {
            HIAI_ENGINE_LOG(HIAI_JPEGE_CTL_ERROR, "call not save result file %s ", outFile.str().c_str());
        }

        outData.cbFree();
        outData.jpgData = nullptr;
        HIAI_ENGINE_LOG(HIAI_OK, "jpeg encode process completed");
    } while (0); // for resource pddvppapi
}
```

6.3 Implementing the JPEGD Function

- The memory size can be obtained by calling **DvppGetOutParameter**. The output memory can be specified and freed by the user. The calling example is as follows:

```
void TEST_JPEGD_CUSTOM_MEMORY()
{
    struct JpegdIn jpegdInData;
    struct JpegdOut jpegdOutData;

    if (g_rank) {
        jpegdInData.isYUV420Need = false;
    }

    FILE *fpIn = fopen(g_inFileName, "rb");
    if (fpIn == nullptr) {
        HIAI_ENGINE_LOG(HIAI_OPEN_FILE_ERROR, "can not open file %s.", g_inFileName);
        return;
    }

    do { // for resource fpIn
        fseek(fpIn, 0, SEEK_END);
        // the buf len should 8 byte larger, the driver asked
        uint32_t fileLen   = ftell(fpIn);
        jpegdInData.jpegDataSize = fileLen + 8;
        fseek(fpIn, 0, SEEK_SET);

        void* addrOrig = HIAI_DVPP_DMalloc(jpegdInData.jpegDataSize);
        if (addrOrig == nullptr) {
            HIAI_ENGINE_LOG(HIAI_JPEGD_CTL_ERROR, "can not alloc input buffer");
            fclose(fpIn);
            fpIn = nullptr;
            break;
        }

        jpegdInData.jpegData = reinterpret_cast<unsigned char*>(addrOrig);
        do { // for resource inBuf
            fread(jpegdInData.jpegData, 1, fileLen, fpIn);
            fclose(fpIn);
            fpIn = nullptr;
            int32_t ret = DvppGetOutParameter((void*)(&jpegdInData), (void*)(&jpegdOutData),
GET_JPEGD_OUT_PARAMETER);
            if (ret != 0) {
                HIAI_ENGINE_LOG(HIAI_JPEGD_CTL_ERROR, "call DvppGetOutParameter process failed");
                break;
            }
            HIAI_ENGINE_LOG("jpegd out size is %d", jpegdOutData.yuvDataSize);
            jpegdOutData.yuvData = (unsigned char*)HIAI_DVPP_DMalloc(jpegdOutData.yuvDataSize);
            if (jpegdOutData.yuvData == nullptr) {
                HIAI_ENGINE_LOG(HIAI_JPEGD_CTL_ERROR, "can not alloc output buffer");
```



```
            break;
        }
        HIAI_ENGINE_LOG("jpegdOutData.yuvData is %p", jpegdOutData.yuvData);

        dvppapi_ctl_msg dvppApiCtlMsg;
        dvppApiCtlMsg.in = (void *)&jpegdInData;
        dvppApiCtlMsg.in_size = sizeof(jpegdInData);
        dvppApiCtlMsg.out = (void *)&jpegdOutData;
        dvppApiCtlMsg.out_size = sizeof(jpegdOutData);

        IDVPPAPI *pidvppapi = nullptr;
        CreateDvppApi(pidvppapi);

        if (pidvppapi != nullptr) {
            for (int i = 0;i < g_loop; i++) {
                if (0 != DvppCtl(pidvppapi, DVPP_CTL_JPEGD_PROC, &dvppApiCtlMsg)) {
                    HIAI_ENGINE_LOG(HIAI_JPEGD_CTL_ERROR, "call dvppctl process failed");
                    break;
                }
            }
            DestroyDvppApi(pidvppapi);
        } else {
            HIAI_ENGINE_LOG(HIAI_CREATE_DVPP_ERROR, "can not create dvpp api");
            break;
        }
        WriteTestJpegdResultToFile(&jpegdOutData);
    } while (0); // for resource inBuf

    if (addrOrig != nullptr) {
        HIAI_DVPP_DFree(addrOrig);
        addrOrig = nullptr;
    }
    if (jpegdOutData.yuvData != nullptr) {
        HIAI_DVPP_DFree(jpegdOutData.yuvData);
        jpegdOutData.yuvData = nullptr;
    }
} while (0); // for resource fpIn
}
```

- If the output buffer is not specified by the user, DVPP allocates the buffer internally. The **cbFree()** callback function should be called to free the buffer. The calling example is as follows:

```
void JpegdProcess()
{
    struct jpegd_raw_data_info jpegdInData;
    struct jpegd_yuv_data_info jpegdOutData;

    if (g_rank) {
        jpegdInData.IsYUV420Need = false;
    }

    FILE *fpIn = fopen(g_inFileName, "rb");
    if (fpIn == nullptr) {
        HIAI_ENGINE_LOG(HIAI_OPEN_FILE_ERROR, "can not open file %s.", g_inFileName);
        return;
    }

    do { // for resource fpIn
        fseek(fpIn, 0, SEEK_END);
        // the buf len should 8 byte larger, the driver asked
        uint32_t fileLen   = ftell(fpIn);
        jpegdInData.jpeg_data_size = fileLen + 8;
        fseek(fpIn, 0, SEEK_SET);

        void* addrOrig = HIAI_DVPP_DMalloc(jpegdInData.jpeg_data_size);
        if (addrOrig == nullptr) {
            HIAI_ENGINE_LOG(HIAI_JPEGD_CTL_ERROR, "can not alloc input buffer");
            break;
        }
```



```
    jpegdInData.jpeg_data = reinterpret_cast<unsigned char*>(addrOrig);

    do { // for resource inBuf
        fread(jpegdInData.jpeg_data, 1, fileLen, fpIn);
        dvppapi_ctl_msg dvppApiCtlMsg;
        dvppApiCtlMsg.in = (void*)&jpegdInData;
        dvppApiCtlMsg.in_size = sizeof(jpegdInData);
        dvppApiCtlMsg.out = (void*)&jpegdOutData;
        dvppApiCtlMsg.out_size = sizeof(jpegdOutData);

        IDVPPAPI *pidvppapi = nullptr;
        CreateDvppApi(pidvppapi);

        if (pidvppapi != nullptr) {
            for (int i = 0;i < g_loop; i++) { // same picture loop test
                if (DvppCtl(pidvppapi, DVPP_CTL_JPEGD_PROC, &dvppApiCtlMsg) != 0) {
                    HIAI_ENGINE_LOG(HIAI_JPEGD_CTL_ERROR, "call dvppctl process failed");
                    break;
                }
                if (i < g_loop - 1) {
                    jpegdOutData.cbFree();
                    jpegdOutData.yuv_data = nullptr;
                }
            }
            DestroyDvppApi(pidvppapi);
        } else {
            HIAI_ENGINE_LOG(HIAI_CREATE_DVPP_ERROR, "can not create dvpp api");
            break;
        }

        WriteJpegdProcessResultToFile(&jpegdOutData);
        jpegdOutData.cbFree();
        jpegdOutData.yuv_data = nullptr;

    } while (0); // for resource inBuf

    if (addrOrig != nullptr) {
        HIAI_DVPP_DFree(addrOrig);
        addrOrig = nullptr;
    }

  } while (0); // for resource fpIn
  fclose(fpIn);
  fpIn = nullptr;
}
```

6.4 Implementing the PNGD Function

The memory size can be obtained by calling **DvppGetOutParameter**. The output memory can be specified and freed by the user. The calling example is as follows:

```
void TEST_PNGD_CUSTOM_MEMORY()
{
    FILE* fpIn = fopen(g_inFileName, "rb");
    if (nullptr == fpIn) {
        HIAI_ENGINE_LOG(HIAI_OPEN_FILE_ERROR, "can not open file %s.", g_inFileName);
        return;
    }

    fseek(fpIn, 0, SEEK_END);
    uint32_t fileLen = ftell(fpIn);
    fseek(fpIn, 0, SEEK_SET);
    void* inbuf = HIAI_DVPP_DMalloc(fileLen);
    if (inbuf == nullptr) {
        HIAI_ENGINE_LOG(HIAI_PNGD_CTL_ERROR, "can not alloc input buffer");
        fclose(fpIn);
```



```
    fpIn = nullptr;
    return;
}

// prepare msg
struct PngInputInfoAPI inputPngData; // input data
inputPngData.inputData = inbuf; // input png data
inputPngData.inputSize = fileLen; // the size of png data

HIAI_ENGINE_LOG("inputPngData.address: %p", inputPngData.address);
HIAI_ENGINE_LOG("inputPngData.size: %u", inputPngData.size);
HIAI_ENGINE_LOG("inputPngData.inputData: %p", inputPngData.inputData);
HIAI_ENGINE_LOG("inputPngData.inputSize: %u", inputPngData.inputSize);

fread(inputPngData.inputData, 1, fileLen, fpIn);
fclose(fpIn);
fpIn = nullptr;

if (g_transform == 1) { // Whether format conversion
    inputPngData.transformFlag = 1; // RGBA -> RGB
} else {
    inputPngData.transformFlag = 0;
}

struct PngOutputInfoAPI outputPngData; // output data
int32_t ret = DvppGetOutParameter((void*)(&inputPngData), (void*)(&outputPngData),
GET_PNGD_OUT_PARAMETER);
if (ret != 0) {
    HIAI_ENGINE_LOG(HIAI_PNGD_CTL_ERROR, "call DvppGetOutParameter process failed");
    HIAI_DVPP_DFree(inbuf);
    inbuf = nullptr;
    return;
}
HIAI_ENGINE_LOG("pngd out size is  %d", outputPngData.size);
outputPngData.address = HIAI_DVPP_DMalloc(outputPngData.size);
if (outputPngData.address == nullptr) {
    HIAI_ENGINE_LOG(HIAI_PNGD_CTL_ERROR, "can not alloc output buffer");
    HIAI_DVPP_DFree(inbuf);
    inbuf = nullptr;
    return;
}

dvppapi_ctl_msg dvppApiCtlMsg; // call the interface msg
dvppApiCtlMsg.in = (void*)(&inputPngData);
dvppApiCtlMsg.in_size = sizeof(struct PngInputInfoAPI);
dvppApiCtlMsg.out = (void*)(&outputPngData);
dvppApiCtlMsg.out_size = sizeof(struct PngOutputInfoAPI);

// use interface
IDVPPAPI *pidvppapi = nullptr;
CreateDvppApi(pidvppapi); // create dvppapi

if (pidvppapi != nullptr) { // use DvppCtl interface to handle DVPP_CTL_PNGD_PROC
    for (int i = 0; i < g_loop; i++) {
        if (-1 == DvppCtl(pidvppapi, DVPP_CTL_PNGD_PROC, &dvppApiCtlMsg)) {
            HIAI_ENGINE_LOG(HIAI_PNGD_CTL_ERROR, "call dvppctl process failed");
            HIAI_DVPP_DFree(inbuf);
            inbuf = nullptr;
            HIAI_DVPP_DFree(outputPngData.address);
            outputPngData.address = nullptr;
            DestroyDvppApi(pidvppapi); // destory dvppapi
            return;
        }
    }
} else {
    HIAI_ENGINE_LOG(HIAI_PNGD_CTL_ERROR, "can not get dvpp api");
}

DestroyDvppApi(pidvppapi);
```



```
    HIAI_ENGINE_LOG("output result");

    char* pAddr = (char*)outputPngData.outputData;
    FILE* fpOut = fopen(g_outFileName, "wb");
    if (fpOut == nullptr) {
        HIAI_ENGINE_LOG(HIAI_OPEN_FILE_ERROR, "can not open file %s.", g_outFileName);
        HIAI_DVPP_DFree(inbuf);
        inbuf = nullptr;
        HIAI_DVPP_DFree(outputPngData.address);
        outputPngData.address = nullptr;
        return;
    }

    int size = outputPngData.width * 4;
    if (outputPngData.format == PNGD_RGB_FORMAT_NUM) {
        size = outputPngData.width * 3;
    } else if (outputPngData.format == PNGD_RGBA_FORMAT_NUM) {
        size = outputPngData.width * 4;
    }
    // copy valid image data from every line.
    for (int i = 0; i < outputPngData.high; i++) {
        fwrite(pAddr + (int)(i * outputPngData.widthAlign), size, 1, fpOut);
    }

    fclose(fpOut);
    fpOut = nullptr;
    HIAI_DVPP_DFree(inbuf);
    inbuf = nullptr;
    HIAI_DVPP_DFree(outputPngData.address);
    outputPngData.address = nullptr;
    return;
}
```

6.5 Implementing the VDEC Function

For a video stream, after you create an instance by calling **CreateVdecApi**, you must use the same instance to call **VdecCtl** for video decoding, and then call **DestroyVdecApi** to release the instance.

During video decoding, if you need to switch from a video stream to another video stream, you must release the instance of the previous stream by calling **DestroyVdecApi**, and then create an instance by calling **CreateVdecApi** to process the new video stream.

This example reads the H.264 stream file named **test_file**, calls the VDEC function, and saves the decoding result to the **output_dir** directory.

```
// User-defined sub-class
class HIAI_DATA_SP_SON: public HIAI_DATA_SP {
public:
    ~ HIAI_DATA_SP_SON ()
    {
        //destruct here;
    }
    // User-defined member functions. The following is only an example.
    uint8_t GetTotalFrameNum()
    {
        return info_.totalFrameNum;
    }

private:
    // User-defined member variables. The INFO structure is only an example.
    struct INFO {
        uint8_t totalFrameNum;
        uint8_t frameRate;
```



```
    }info_;
};

IDVPPAPI * pidvppapi_vpc = NULL;
IDVPPAPI * pidvppapi_vdec = NULL;

// The callback function is used as an example. The caller needs to redefine the callback function as
required.
void FrameReturn(FRAME* frame, void* hiaiData)
{
    static int32_t imageCount = 0;
    imageCount++;
    HIAI_ENGINE_LOG("call save frame number:[%d], width:[%d], height:[%d]", imageCount, frame->width,
frame->height);
    // The image directly output by vdec is an image of hfbc compression format, which cannot be directly
displayed.
    // It is necessary to call vpc to convert hfbc to an image of uncompressed format to display.
    HIAI_ENGINE_LOG("start call vpc interface to translate hfbc.");
    IDVPPAPI* dvppHandle = nullptr;
    int32_t ret = CreateDvppApi(dvppHandle);
    if (ret != 0) {
        HIAI_ENGINE_LOG(HIAI_CREATE_DVPP_ERROR, "creat dvpp api fail.");
        return;
    }

    // Construct vpc input configuration.
    std::shared_ptr<VpcUserImageConfigure> userImage(new VpcUserImageConfigure);
    // bareDataAddr should be null which the image is hfbc.
    userImage->bareDataAddr = nullptr;
    userImage->bareDataBufferSize = 0;
    userImage->widthStride = frame->width;
    userImage->heightStride = frame->height;
    // Configuration input format
    string imageFormat(frame->image_format);
    if (frame->bitdepth == 8) {
        if (imageFormat == "nv12") {
            userImage->inputFormat = INPUT_YUV420_SEMI_PLANNER_UV;
        } else {
            userImage->inputFormat = INPUT_YUV420_SEMI_PLANNER_VU;
        }
    } else {
        if (imageFormat == "nv12") {
            userImage->inputFormat = INPUT_YUV420_SEMI_PLANNER_UV_10BIT;
        } else {
            userImage->inputFormat = INPUT_YUV420_SEMI_PLANNER_VU_10BIT;
        }
    }
    userImage->outputFormat = OUTPUT_YUV420SP_UV;
    userImage->isCompressData = true;
    // Configure hfbc input address
    VpcCompressDataConfigure* compressDataConfigure = &userImage->compressDataConfigure;
    uint64_t baseAddr = (uint64_t)frame->buffer;
    compressDataConfigure->lumaHeadAddr = baseAddr + frame->offset_head_y;
    compressDataConfigure->chromaHeadAddr = baseAddr + frame->offset_head_c;
    compressDataConfigure->lumaPayloadAddr = baseAddr + frame->offset_payload_y;
    compressDataConfigure->chromaPayloadAddr = baseAddr + frame->offset_payload_c;
    compressDataConfigure->lumaHeadStride = frame->stride_head;
    compressDataConfigure->chromaHeadStride = frame->stride_head;
    compressDataConfigure->lumaPayloadStride = frame->stride_payload;
    compressDataConfigure->chromaPayloadStride = frame->stride_payload;

    userImage->yuvSumEnable = false;
    userImage->cmdListBufferAddr = nullptr;
    userImage->cmdListBufferSize = 0;
    // Configure the roi area and output area
    std::shared_ptr<VpcUserRoiConfigure> roiConfigure(new VpcUserRoiConfigure);
    roiConfigure->next = nullptr;
    userImage->roiConfigure = roiConfigure.get();
    VpcUserRoiInputConfigure* roiInput = &roiConfigure->inputConfigure;
```



```
    roiInput->cropArea.leftOffset = 0;
    roiInput->cropArea.rightOffset = frame->width - 1;
    roiInput->cropArea.upOffset = 0;
    roiInput->cropArea.downOffset = frame->height - 1;
    VpcUserRoiOutputConfigure* roiOutput = &roiConfigure->outputConfigure;
    roiOutput->outputArea.leftOffset = 0;
    roiOutput->outputArea.rightOffset = frame->width - 1;
    roiOutput->outputArea.upOffset = 0;
    roiOutput->outputArea.downOffset = frame->height - 1;
    roiOutput->bufferSize = ALIGN_UP(frame->width, 16) * ALIGN_UP(frame->height, 2) * 3 / 2;
    roiOutput->addr = (uint8_t*)HIAI_DVPP_DMalloc(roiOutput->bufferSize);
    if (roiOutput->addr == nullptr) {
        HIAI_ENGINE_LOG(HIAI_VPC_CTL_ERROR, "can not alloc roiOutput buffer");
        DestroyDvppApi(dvppHandle);
        return;
    }
    roiOutput->widthStride = ALIGN_UP(frame->width, 16);
    roiOutput->heightStride = ALIGN_UP(frame->height, 2);

    dvppapi_ctl_msg dvppApiCtlMsg;
    dvppApiCtlMsg.in = (void*)(userImage.get());
    dvppApiCtlMsg.in_size = sizeof(VpcUserImageConfigure);
    ret = DvppCtl(dvppHandle, DVPP_CTL_VPC_PROC, &dvppApiCtlMsg);
    if (ret != 0) {
        HIAI_ENGINE_LOG(HIAI_ERROR, "call dvppctl fail");
        HIAI_DVPP_DFree(roiOutput->addr);
        roiOutput->addr = nullptr;
        DestroyDvppApi(dvppHandle);
        return;
    }
    ret = FrameReturnSaveVpcResult(frame, roiOutput->addr, imageCount);
    if (ret != 0) {
        HIAI_ENGINE_LOG(HIAI_ERROR, "save vpc result fail");
    }
    HIAI_DVPP_DFree(roiOutput->addr);
    roiOutput->addr = nullptr;
    DestroyDvppApi(dvppHandle);
    return;
}

/*
*Error reporting callback function. If you do not need error information, you do not need to define this
function.
*/
void ErrReport(VDECERR* vdecErr)
{
    if (vdecErr != nullptr) {
        HIAI_ENGINE_LOG(HIAI_CREATE_DVPP_ERROR, "vdec error code is %d, channelId = %u\n", vdecErr-
>errType, vdecErr->channelId);
    }
}

// Main entry of the test function
void TEST_VDEC()
{
    char outputDir[20] = {0};
    int32_t safeFuncRet = memset_s(outputDir, sizeof(outputDir), 0, 20);
    if (safeFuncRet != EOK) {
        HIAI_ENGINE_LOG(HIAI_ERROR, "memset_s fail");
        return;
    }
    safeFuncRet = strncpy_s(outputDir, sizeof(outputDir), "output_dir", strlen("output_dir"));
    if (safeFuncRet != EOK) {
        HIAI_ENGINE_LOG(HIAI_ERROR, "strncpy_s fail");
        return;
    }
    if (access(outputDir, F_OK) == -1) {
        if (mkdir(outputDir, 0700) < 0) { // directory authority: 0700
            HIAI_ENGINE_LOG(HIAI_VDEC_CTL_ERROR, "create dir failed.");
```



```
        return;
    }
  } else if (access(outputDir, R_OK | W_OK | X_OK) == -1) {
    HIAI_ENGINE_LOG(HIAI_VDEC_CTL_ERROR, "output directory authority is not correct, use 700
instead.");
    return;
  }
  IDVPPAPI *pidvppapi = nullptr;
  int32_t ret = CreateVdecApi(pidvppapi, 0);
  if (ret != 0) {
    HIAI_ENGINE_LOG(HIAI_CREATE_DVPP_ERROR, "create dvpp api fail.");
    return;
  }
  if (pidvppapi != nullptr) {
    FILE* fp = fopen(g_inFileName, "rb");
    if (fp == nullptr) {
      HIAI_ENGINE_LOG(HIAI_OPEN_FILE_ERROR, "open file: %s failed.", g_inFileName);
      DestroyVdecApi(pidvppapi, 0);
      return;
    }
    fseek(fp, 0L, SEEK_END);
    int file_size = ftell(fp);
    fseek(fp, 0L, SEEK_SET);
    int rest_len = file_size;
    int len = file_size;

    vdec_in_msg vdec_msg;
    vdec_msg.call_back = FrameReturn;
    vdec_msg.err_report = ErrReport; // can be unassigned, if user does not need error info
    vdec_msg.hiai_data = nullptr;
    int32_t safeFuncRet = 0;
    if (g_format == 0) {
      safeFuncRet = memcpy_s(vdec_msg.video_format, sizeof(vdec_msg.video_format), "h264", 4);
      if (safeFuncRet != EOK) {
        HIAI_ENGINE_LOG(HIAI_ERROR, "memcpy_s fail");
        DestroyVdecApi(pidvppapi, 0);
        fclose(fp);
        fp = nullptr;
        return;
      }
    } else {
      safeFuncRet = memcpy_s(vdec_msg.video_format, sizeof(vdec_msg.video_format), "h265", 4);
      if (safeFuncRet != EOK) {
        HIAI_ENGINE_LOG(HIAI_ERROR, "memcpy_s fail");
        DestroyVdecApi(pidvppapi, 0);
        fclose(fp);
        fp = nullptr;
        return;
      }
    }
    vdec_msg.in_buffer = (char*)malloc(len);
    if (vdec_msg.in_buffer == nullptr) {
      HIAI_ENGINE_LOG(HIAI_ERROR, "malloc fail");
      fclose(fp);
      fp = nullptr;
      DestroyVdecApi(pidvppapi, 0);
      return;
    }

    dvppapi_ctl_msg dvppApiCtlMsg;
    dvppApiCtlMsg.in = (void*)(&vdec_msg);
    dvppApiCtlMsg.in_size = sizeof(vdec_in_msg);

    while (rest_len > 0) {
      int read_len = fread(vdec_msg.in_buffer, 1, len, fp);
      HIAI_ENGINE_LOG("rest_len is %d,read len is %d.", rest_len, read_len);
      vdec_msg.in_buffer_size = read_len;
      fclose(fp);
      fp = nullptr;
```

```
            if (VdecCtl(pidvppapi, DVPP_CTL_VDEC_PROC, &dvppApiCtlMsg, 0) != 0) {
                HIAI_ENGINE_LOG(HIAI_VDEC_CTL_ERROR, "call dvppctl process faild!");
                free(vdec_msg.in_buffer);
                vdec_msg.in_buffer = nullptr;
                DestroyVdecApi(pidvppapi, 0);
                return;
            }
            HIAI_ENGINE_LOG(HIAI_OK, "call vdec process success!");
            rest_len = rest_len - len;
        }
        free(vdec_msg.in_buffer);
        vdec_msg.in_buffer = nullptr;
    } else {
        HIAI_ENGINE_LOG(HIAI_VDEC_CTL_ERROR, "pidvppapi is null!");
    }
    DestroyVdecApi(pidvppapi, 0);
    return;
}
```

6.6 Implementing the VENC Function

To encode multiple images into a video, call **RunVenc** to encode the video by using the same instance after creating an instance by calling **CreateVenc**, and then call **DestroyVenc** to release the instance.

In this example, the VENC API is called to encode YUV images into H.265 or H.264 streams.

```
std::string vencOutFileName("venc.bin");
std::shared_ptr<FILE> vencOutFile(nullptr);
void VencCallBackDumpFile(struct VencOutMsg* vencOutMsg, void* userData)
{
    if (vencOutFile.get() == nullptr) {
        HIAI_ENGINE_LOG(HIAI_VENC_CTL_ERROR, "get venc out file fail!");
        return;
    }
    fwrite(vencOutMsg->outputData, 1, vencOutMsg->outputDataSize, vencOutFile.get());
    fflush(vencOutFile.get());
}

/*
 * venc new interface to achieve venc basic functions.
 */
void TEST_VENC()
{
    std::shared_ptr<FILE> fpIn(fopen(g_inFileName, "rb"), fclose);
    vencOutFile.reset(fopen(vencOutFileName.c_str(), "wb"), fclose);

    if (fpIn.get() == nullptr || vencOutFile.get() == nullptr) {
        HIAI_ENGINE_LOG(HIAI_OPEN_FILE_ERROR, "open open venc in/out file failed.");
        return;
    }

    fseek(fpIn.get(), 0, SEEK_END);
    uint32_t fileLen = ftell(fpIn.get());
    fseek(fpIn.get(), 0, SEEK_SET);

    struct VencConfig vencConfig;
    vencConfig.width = g_width;
    vencConfig.height = g_high;
    vencConfig.codingType = g_format;
    vencConfig.yuvStoreType = g_yuvStoreType;
    vencConfig.keyFrameInterval = 16;
    vencConfig.vencOutMsgCallBack = VencCallBackDumpFile;
    vencConfig.userData = nullptr;

    int32_t vencHandle = CreateVenc(&vencConfig);
```



```
    if (vencHandle == -1) {
        HIAI_ENGINE_LOG(HIAI_VENC_CTL_ERROR, "CreateVenc fail!");
        return;
    }

    // input 16 frames once
    uint32_t inDataLenMaxOnce = g_width * g_high * 3 / 2;
    std::shared_ptr<char> inBuffer(static_cast<char*>(malloc(inDataLenMaxOnce)), free);

    if (inBuffer.get() == nullptr) {
        HIAI_ENGINE_LOG(HIAI_OPEN_FILE_ERROR, "alloc input buffer failed");
        DestroyVenc(vencHandle);
        return;
    }

    uint32_t inDataUnhandledLen = fileLen;

    auto start = std::chrono::system_clock::now();
    uint32_t frameCount = 0;

    while (inDataUnhandledLen > 0) {
        uint32_t inDataLen      = inDataUnhandledLen;
        if (inDataUnhandledLen > inDataLenMaxOnce) {
            inDataLen           = inDataLenMaxOnce;
        }
        inDataUnhandledLen    -= inDataLen;
        uint32_t readLen        = fread(inBuffer.get(), 1, inDataLen, fpIn.get());
        if (readLen != inDataLen) {
            HIAI_ENGINE_LOG(HIAI_OPEN_FILE_ERROR, "error in read input file");
            DestroyVenc(vencHandle);
            return;
        }

        struct VencInMsg vencInMsg;
        vencInMsg.inputData = inBuffer.get();
        vencInMsg.inputDataSize = inDataLen;
        vencInMsg.keyFrameInterval = 16;
        vencInMsg.forceIFrame = 0;
        vencInMsg.eos = 0;

        if (RunVenc(vencHandle, &vencInMsg) == -1) {
            HIAI_ENGINE_LOG(HIAI_VENC_CTL_ERROR, "call video encode fail");
            break;
        }
        ++frameCount;
    }

    auto end = std::chrono::system_clock::now();
    auto duration = std::chrono::duration_cast<std::chrono::microseconds>(end - start);
    size_t timeCount = static_cast<size_t>(duration.count());
    HIAI_ENGINE_LOG("Total frame count: %u", frameCount);
    HIAI_ENGINE_LOG("Time cost: %lu us", timeCount);
    HIAI_ENGINE_LOG("FPS: %lf", frameCount * 1.0 / (timeCount / 1000000.0));

    DestroyVenc(vencHandle);
    return;
}
```


7 Data Types

7.1 Structures in VpcUserImageConfigure

The structures are defined in the **ddk/include/inc/dvpp/Vpc.h** file in the DDK installation directory.

- **VpcUserRoiConfigure** structure

Member Variable	Description
VpcUserRoiInputConfigure inputConfigure	User ROI input configuration. For details, see **VpcUserRoiInputConfigure structure**. If the 8K image scaling function is implemented, this parameter does not need to be set.
VpcUserRoiOutputConfigure outputConfigure	User ROI output configuration. For details, see **VpcUserRoiOutputConfigure structure**.
VpcUserRoiConfigure* next	Next user ROI configuration. If the single-image multi-ROI mode is used, configure this parameter. Otherwise, set this parameter to **NULL**. The default value is **NULL**.
uint64_t reserve1	Reserved

- **VpcCompressDataConfigure** structure

Member Variable	Description
uint64_t lumaHeadAddr	Header address of the Y component
uint64_t chromaHeadAddr	Header address of the U and V components
uint32_t lumaHeadStride	Header stride of the Y component, which must be the same as the value of **stride_head** in the **FRAME structure**.
uint32_t chromaHeadStride	Header stride of the U and V components, which must be the same as the value of **stride_head** in the **FRAME structure**.
uint64_t lumaPayloadAddr	Payload address of the Y component
uint64_t chromaPayloadAddr	Payload address of the U and V components
uint32_t lumaPayloadStride	Payload stride of the Y component, which must be the same as the value of **stride_payload** in the **FRAME structure**.
uint32_t chromaPayloadStride	Payload stride of the U and V components, which must be the same as the value of **stride_payload** in the **FRAME structure**.

- **VpcUserYuvSum** structure

Member Variable	Description
uint32_t ySum	Y component sum
uint32_t uSum	U component sum
uint32_t vSum	V component sum
uint64_t reserve1	Reserved

- **VpcUserPerformanceTuningParameter** structure

Member Variable	Description
uint64_t reserve1	Reserved parameter 1
uint64_t reserve2	Reserved parameter 2
uint64_t reserve3	Reserved parameter 3
uint64_t reserve4	Reserved parameter 4

Member Variable	Description
uint64_t reserve5	Reserved parameter 5

- **VpcUserRoiInputConfigure** structure

Member Variable	Description
VpcUserCropConfigure cropArea	Input data configuration of the cropped area. For details, see **VpcUserCropConfigure structure**.
uint64_t reserve1	Reserved

- **VpcUserRoiOutputConfigure** structure

Member Variable	Description
uint8_t* addr	Start address of the output image You are advised to allocate the buffer by calling **HIAI_DVPP_DMalloc** provided by Matrix. The allocated buffer must meet the DVPP requirements, that is, the buffer addresses of the input and output images must be in the same 4 GB space, and the start addresses must be 16-pixel aligned. For details about **HIAI_DVPP_DMalloc**, see *Matrix API Reference*.
uint32_t bufferSize	Output buffer size calculated based on the YUV420SP format
uint32_t widthStride	Width stride of the output image, which must be 16-pixel aligned. The minimum width stride is 32, and the maximum width stride is 4096.
uint32_t heightStride	Height stride of the output image, which must be 2-pixel aligned. The minimum height stride is 6, and the maximum height stride is 4096. If the output is a YUV420SP image, you need to calculate the start addresses of the U and V data based on the value of **heightStride**.

Member Variable	Description
VpcUserCropConfigure outputArea	Coordinates of the output area specified by the user. For details, see **VpcUserCropConfigure structure**. If the 8K image scaling function is implemented, this parameter does not need to be set.
uint64_t reserve1	Reserved

- **VpcUserCropConfigure** structure

  For details about the top offset, bottom offset, left offset, and right offset, see **Table 1-2**.

Member Variable	Description
uint32_t leftOffset	Left offset. The value must be an even number. The left offset of the overwritten area relative to the output image is 16-pixel aligned.
uint32_t rightOffset	Right offset. The value must be an odd number.
uint32_t upOffset	Top offset. The value must be an even number.
uint32_t downOffset	Bottom offset. The value must be an odd number.
uint64_t reserve1	Reserved

7.2 Structures and Classes in vdec_in_msg

The structures are defined in the **ddk/include/inc/dvpp/Vdec.h** file in the DDK installation directory.

- **FRAME** structure

Member Variable	Description
int height	Height of the output image after alignment. For the H.264 format, the value is 16-pixel aligned. For the H.265 format, the value is 64-pixel aligned.

Member Variable	Description
int width	Width of the output image after alignment. For the H.264 format, the value is 16-pixel aligned. For the H.265 format, the value is 64-pixel aligned.
int realHeight	Actual image height
int realWidth	Actual image width
unsigned char* buffer	Buffer address of the output image
int buffer_size	Buffer size of the output image
unsigned int offset_payload_y	Y component offset of the output image payload. Y component address of the payload = Buffer + **offset_payload_y**
unsigned int offset_payload_c	C component offset of the output image payload. C component address of the payload = Buffer + **offset_payload_c**
unsigned int offset_head_y	Y component offset of the output image header. Y component address of the header = Buffer + **offset_head_y**
unsigned int offset_head_c	C component offset of the output image header. C component address of the header = Buffer + **offset_head_c**
unsigned int stride_payload	Payload stride of the output image
unsigned int stride_head	Header stride of the output image
unsigned short bitdepth	Bit depth of the output image
char video_format[10]	Video format. The format can be **h264** or **h265**.
char image_format[10]	Output image format. The format can be **nv12** or **nv21**.

- **HIAI_DATA_SP** class

Member Variable or Function	Description
unsigned long long frameIndex	Frame sequence number
void * frameBuffer	Buffer for storing output frames
unsigned int frameSize	Size of the buffer for storing output frames

Member Variable or Function	Description
void setFrameIndex(unsigned long long index)	Function for configuring the frame sequence number
unsigned long long getFrameIndex()	Function for obtaining the frame sequence number
void setFrameBuffer(void * frameBuff)	Function for setting the frame buffer address
void * getFrameBuffer()	Function for obtaining the frame buffer address
void setFrameSize(unsigned int size)	Function for setting the frame size
unsigned int getFrameSize()	Function for obtaining the frame size

- **VDECERR** structure

Member Variable	Description

ERRTYPE errType	Error type
	enum ERRTYPE{
	// An error occurs when the VDEC state is abnormal. You need to destroy the VDEC instance and then create a new one.
	ERR_INVALID_STATE = 0x10001,
	// A hardware error occurs. For example, the VDEC is started, executed, or stopped abnormally. You need to destroy the VDEC instance and then create a new one.
	ERR_HARDWARE,
	// An error occurs when the video stream is divided into multiple frames of images. You need to check whether the input video stream data is correct.
	ERR_SCD_CUT_FAIL,
	// An error occurs when a frame is decoded. You need to check whether the input video stream data is correct.
	ERR_VDM_DECODE_FAIL,
	// Reserved
	ERR_ALLOC_MEM_FAIL,
	// An error occurs when the input video resolution exceeds the range or the internal dynamic memory allocation fails. You need to check the resolution of the input video or available memory of the system as required.
	ERR_ALLOC_DYNAMIC_MEM_FAIL,
	// An error occurs when the input and output buffers of the VDEC are allocated. You need to check whether the system has available memory.
	ERR_ALLOC_IN_OR_OUT_PORT_MEM_FAIL,
	// A stream error occurs. This structure is reserved.
	ERR_BITSTREAM,
	// A video format error occurs. This structure is reserved.
	ERR_VIDEO_FORMAT,
	// An error occurs in the output format configuration. This structure is reserved.
	ERR_IMAGE_FORMAT,
	// A null error occurs in the callback function. This structure is reserved.
	ERR_CALLBACK,
	// A null error occurs in the input buffer. This structure is reserved.

Member Variable	Description

	ERR_INPUT_BUFFER,
	// An error occurs when the size of the input buffer is less than or equal to 0. This structure is reserved.
	ERR_INBUF_SIZE,
	// A thread error occurs when the system returns the decoding result through the callback function. You need to check whether the resources (such as threads and memory) in the system are available.
	ERR_THREAD_CREATE_FBD_FAIL,
	// An error occurs when the VDEC instance fails to be created. You need to create a VDEC instance again.
	ERR_CREATE_INSTANCE_FAIL,
	// An error occurs when the VDEC fails to be initialized. For example, the number of VDEC instances exceeds 16, which is the upper limit. In this case, you need to release some VDEC instances and create new ones.
	ERR_INIT_DECODER_FAIL,
	// An error occurs when the VDEC handle to a video stream fails to be obtained. You need to create a VDEC instance again.
	ERR_GET_CHANNEL_HANDLE_FAIL,
	// An error occurs when a VENC instance is set abnormally in the system. You need to check whether the input parameter values of the VENC are correct, such as the input video format (**video_format**) and output frame format (**image_format**).
	ERR_COMPONENT_SET_FAIL,
	// An error occurs when a VENC instance name is set abnormally in the system. You need to check whether the input parameter values of the VENC are correct, such as the input video format (**video_format**) and output frame format (**image_format**).
	ERR_COMPARE_NAME_FAIL,
	// Other errors occur.
	ERR_OTHER
	};

unsigned short channelId	Channel for a decoding error

7.3 Structures in IMAGE_CONFIG

- **ROI_CONFIG** structure

Member Variable	Description	Value Range
CROP_CONFIG crop_config	Structure for cropping parameter configuration	For details, see **CROP_CONFIG structure**.
YUV_SUM_OUT_CONFIG sum_out	Structure 1 of output parameter configuration	For details, see **YUV_SUM_OUT_CONFIG structure**.
NORMAL_OUT_CONFIG normal_out	Structure 2 of output parameter configuration, which is reserved and not supported currently	For details, see **NORMAL_OUT_CONFIG structure**.
ROI_CONFIG* next	Pointer to the next **ROI_CONFIG** structure	Set this parameter when multiple ROIs are used. Otherwise, set this parameter to **NULL**.

- **CROP_CONFIG** structure

Member Variable	Description	Value Range
int enable	Whether to perform cropping	**0**: disabled **1**: enabled
unsigned int hmin	Minimum horizontal offset	The value is an even number and less than **hmax**.
unsigned int hmax	Maximum horizontal offset	The value is an odd number and less than the input width (**in_width**).
unsigned int vmin	Minimum vertical offset	The value is an even number and less than **vmax**.
unsigned int vmax	Maximum vertical offset	The value is an odd number and less than the input height (**in_height**).

- **YUV_SUM_OUT_CONFIG** structure

Member Variable	Description	Value Range
int enable	Whether to enable the channel for output	**0**: disabled **1**: enabled
unsigned int out_width	Output image width	The value is an even number. The value range is [128, 4096] and the resizing ratio range is [0.03125, 4].
unsigned int out_height	Output image height	The value is an even number. The value range is [16, 4096] and the resizing ratio range is [0.03125, 4].
char* out_buffer	Pointer to the address of the output image buffer	The output buffer size must be calculated based on the output image resolution after the width is 128-pixel aligned and the height is 16-pixel aligned.
unsigned int yuv_sum	Sum of all YUV values of the output image	Reserved (not supported yet)

- **NORMAL_OUT_CONFIG** structure

Member Variable	Description	Value Range
int enable	Whether to enable the channel for output	**0**: disabled **1**: enabled
unsigned int out_width	Output image width	The value is an even number. The value range is [128, 4096] and the resizing ratio range is [0.03125, 4].
unsigned int out_height	Output image height	The value is an even number. The value range is [16, 4096] and the resizing ratio range is [0.03125, 4].
char* out_buffer	Pointer to the address of the output image buffer	The output buffer size must be calculated based on the output image resolution after the width is 128-pixel aligned and the height is 16-pixel aligned.

7.4 Structures in dvpp_engine_capability_stru

The structures are defined in the **ddk/include/inc/dvpp/DvppCapability.h** file in the DDK installation directory.

- **dvpp_resolution_stru** structure

Member Variable	Description	Value Range
uint32_t resolution_high;	Height resolution	Maximum value: VDEC: 4096 JPEGD: 8192 PNGD: 4096 JPEGE: 8192 VPC: 4096 VENC: 1920 Minimum value: VDEC: 128 JPEGD: 32 PNGD: 32 JPEGE: 32 VPC: 16 VENC: 128
uint32_t resolution_width;	Width resolution	Maximum value: VDEC: 4096 JPEGD: 8192 PNGD: 4096 JPEGE: 8192 VPC: 4096 VENC: 1920 Minimum value: VDEC: 128 JPEGD: 32 PNGD: 32 JPEGE: 32 VPC: 16 VENC: 128

- **dvpp_format_unit_stru** structure

Member Variable	Description	Value Range

enum dvpp_color_format color_format;	Supported image format	*enum* dvpp_color_format {
		//YUV444 in different ordering of YUV Semi-Planar/Packed, 8-Bit, and Linear.
		DVPP_COLOR_YUV444_YUV_P_8BIT_LIN,
		DVPP_COLOR_YUV444_YVU_P_8BIT_LIN,
		DVPP_COLOR_YUV444_UYV_P_8BIT_LIN,
		DVPP_COLOR_YUV444_UVY_P_8BIT_LIN,
		DVPP_COLOR_YUV444_VYU_P_8BIT_LIN,
		DVPP_COLOR_YUV444_VUY_P_8BIT_LIN,
		DVPP_COLOR_YUV444_UV_SP_8BIT_LIN,
		DVPP_COLOR_YUV444_VU_SP_8BIT_LIN,
		/*422*/
		DVPP_COLOR_YUYV422_YUYV_P_8BIT_LIN,
		DVPP_COLOR_YUYV422_YVYU_P_8BIT_LIN,
		DVPP_COLOR_YUYV422_UYVY_P_8BIT_LIN,
		DVPP_COLOR_YUYV422_VYUY_P_8BIT_LIN,
		DVPP_COLOR_YUV422_UV_SP_8BIT_LIN,
		DVPP_COLOR_YUV422_VU_SP_8BIT_LIN,
		/*420*/
		DVPP_COLOR_YUV420_SP_8BIT_LIN,
		DVPP_COLOR_YVU420_SP_8BIT_LIN,
		DVPP_COLOR_YUV420_SP_8BIT_HFBC,
		DVPP_COLOR_YVU420_SP_8BIT_HFBC,
		DVPP_COLOR_YUV420_SP_10BIT_HFBC,
		DVPP_COLOR_YVU420_SP_10BIT_HFBC,
		DVPP_COLOR_YUV420_P_8BIT_LIN,

Member Variable	Description	Value Range
		/*400*/ DVPP_COLOR_YVU400_SP_8BIT, /*rgb888*/ DVPP_COLOR_RGB888_RGB_P_8BIT_LIN, DVPP_COLOR_RGB888_RBG_P_8BIT_LIN, DVPP_COLOR_RGB888_GBR_P_8BIT_LIN, DVPP_COLOR_RGB888_GRB_P_8BIT_LIN, DVPP_COLOR_RGB888_BRG_P_8BIT_LIN, DVPP_COLOR_RGB888_BGR_P_8BIT_LIN, /*argb888*/ DVPP_COLOR_ARGB8888_ARGB_P_8BIT_LIN, DVPP_COLOR_ARGB8888_ARBG_P_8BIT_LIN, DVPP_COLOR_ARGB8888_AGBR_P_8BIT_LIN, DVPP_COLOR_ARGB8888_AGRB_P_8BIT_LIN, DVPP_COLOR_ARGB8888_ABRG_P_8BIT_LIN, DVPP_COLOR_ARGB8888_ABGR_P_8BIT_LIN, DVPP_COLOR_ARGB8888_RAGB_P_8BIT_LIN, DVPP_COLOR_ARGB8888_RABG_P_8BIT_LIN, DVPP_COLOR_ARGB8888_RGBA_P_8BIT_LIN, DVPP_COLOR_ARGB8888_RGAB_P_8BIT_LIN, DVPP_COLOR_ARGB8888_RBAG_P_8BIT_LIN, DVPP_COLOR_ARGB8888_RBGA_P_8BIT_LIN, DVPP_COLOR_ARGB8888_BRGA_P_8BIT_LIN, DVPP_COLOR_ARGB8888_BRAG_P_8BIT_LIN, DVPP_COLOR_ARGB8888_BGAR_P_8BIT_LIN, DVPP_COLOR_ARGB8888_BGRA_P_8BIT_LIN, DVPP_COLOR_ARGB8888_BAR

Member Variable	Description	Value Range
		G_P_8BIT_LIN, DVPP_COLOR_ARGB8888_BAGR_P_8BIT_LIN, DVPP_COLOR_ARGB8888_GRAG_P_8BIT_LIN, DVPP_COLOR_ARGB8888_GRBA_P_8BIT_LIN, DVPP_COLOR_ARGB8888_GABR_P_8BIT_LIN, DVPP_COLOR_ARGB8888_GARB_P_8BIT_LIN, DVPP_COLOR_ARGB8888_GBRA_P_8BIT_LIN, DVPP_COLOR_ARGB8888_GBAR_P_8BIT_LIN, PIC_JPEG, PIC_PNG, VIO_H265, VIO_H264 };

uint32_t compress_type;	Compression type	enum dvpp_compress_type {
		arithmetic_code =0,
		huffman_code
		};

uint32_t stride_size;	Stride size	VDEC: 128 JPEGD: 128 PNGD: 128 JPEGE: 0 VPC: 128 VENC: 0

enum dvpp_high_align_type high_alignment;	Height alignment type	enum dvpp_high_align_type {
		pix_random = 0,
		two_pix_alignment = 2,
		four_pix_alignment = 4,
		eight_pix_alignment = 8,
		sixteen_pix_alignment = 16
		};

Member Variable	Description	Value Range

enum dvpp_high_align_type width_alignment;	Width alignment type	enum dvpp_high_align_type {
		pix_random = 0,
		two_pix_alignment = 2,
		four_pix_alignment = 4,
		eight_pix_alignment = 8,
		sixteen_pix_alignment = 16
		};

uint32_t out_mem_alignment;	Output buffer alignment	-

- **dvpp_perfomance_unit_stru** structure

Member Variable	Description	Value Range
uint32_t resolution_high;	Height resolution	VDEC: 1920 JPEGD: 1920 PNGD: 1920 JPEGE: 1920 VPC: 3840 VENC: 1920
uint32_t resolution_width;	Width resolution	VDEC: 1080 JPEGD: 1080 PNGD: 1080 JPEGE: 1080 VPC: 2160 VENC: 1080
uint32_t stream_num;	Stream size	VDEC: 16 JPEGD: 0 PNGD: 0 JPEGE: 0 VPC: 0 VENC: 1
unsigned long fps;	Frame rate	VDEC: 30 JPEGD: 256 PNGD: 24 JPEGE: 64 VPC: 90 VENC: 30

- **dvpp_pre_contraction_stru** structure

Member Variable	Description	Value Range

enum dvpp_support_type is_support;	Whether pre-contraction is supported	VPC: supported
		Others: not supported
		The options are as follows:
		enum dvpp_support_type {
		no_support =0, //no support
		do_support //support
		};

uint32_t contraction_types;	Contraction type	VPC: 3 Others: 0
uint32_t contraction_size[DVPP_PRE_CONTRATION_TYPE_MAX];	Fixed ratio of pre-contraction	VPC: **2**, **4**, or **8** Others: **0**
enum dvpp_support_type is_horizontal_support;	Whether horizontal pre-contraction is supported	VPC: supported Others: not supported
enum dvpp_support_type is_vertical_support;	Whether vertical pre-contraction is supported	VPC: supported Others: not supported

- **dvpp_pos_scale_stru** structure

Member Variable	Description	Value Range
enum dvpp_support_type is_support;	Whether post-scaling is supported	VPC: supported Others: not supported
uint32_t min_scale;	Minimum scaling coefficient	VPC: **1** Others: **1**
uint32_t max_scale;	Maximum scaling coefficient	VPC: **4** Others: **1**
enum dvpp_support_type is_horizontal_support;	Whether horizontal post-scaling is supported	VPC: supported Others: not supported
enum dvpp_support_type is_vertical_support;	Whether vertical post-scaling is supported	VPC: supported Others: not supported

- **dvpp_vpc_data_spec_stru** structure

Member Variable	Description	Value Range
uint32_t input_type;	Type of the input format	The options are as follows: • 0: HFBC • 1: YUV or RGB
struct dvpp_resolution_stru min_resolution;	Minimum resolution	For details, see **struct dvpp_resolution_...**.
struct dvpp_resolution_stru max_resolution;	Maximum resolution	For details, see **struct dvpp_resolution_...**.

enum dvpp_align_type high_alignment;	Height alignment type	enum dvpp_high_align_type {
		//1-pixel alignment
		pix_random = 0,
		//2-pixel alignment
		two_pix_alignment = 2,
		//4-pixel alignment
		four_pix_alignment = 4,
		//8-pixel alignment
		eight_pix_alignment = 8,
		//16-pixel alignment
		sixteen_pix_alignment = 16
		};

enum dvpp_align_type width_alignment;	Width alignment type	enum dvpp_high_align_type {
		//1-pixel alignment
		pix_random = 0,
		//2-pixel alignment
		two_pix_alignment = 2,
		//4-pixel alignment
		four_pix_alignment = 4,
		//8-pixel alignment
		eight_pix_alignment = 8,
		//16-pixel alignment
		sixteen_pix_alignment = 16
		};

7.5 Structures in vpc_in_msg

7.5.1 RDMACHANNEL Structure

Member Variable	Description	Value Range
Long luma_head_addr	Header address of the Y component	-
Long chroma_head_addr	Header address of the U and V components	-
unsigned int luma_head_stride	Header stride of the Y component	-
nsigned int chroma_head_stride;	Header stride of the U and V components	-
long luma_payload_addr	Address of the Y component data	-
long chroma_payload_addr	Address of the U and V component data	-
unsigned int luma_payload_stride	Header stride of the Y component data	-
unsigned int chroma_payload_stride;	Header stride of the U and V component data	-

7.5.2 VpcTurningReverse Structure

Member Variable	Description	Value Range
unsigned int reverse1	Reserved interface 1 for internal tuning	--
unsigned int reverse2	Reserved interface 2 for internal tuning	--
unsigned int reverse3	Reserved interface 3 for internal tuning	--
unsigned int reverse4	Reserved interface 4 for internal tuning	--

8 Appendix

8.1 Auxiliary Function APIs

- JpegCalBackFree class auxiliary API:

class JpegCalBackFree:

JpegCalBackFree() :go

~JpegCalBackFree()

JpegCalBackFree(JpegCalBackFree& others)

void setBuf(void* addr4Free)

void setBuf(void* addr4Free, size_t size4Free)

void operator() ()

const JpegCalBackFree& operator= (JpegCalBackFree& others)

- AutoBuffer class auxiliary API:

class AutoBuffer:

AutoBuffer()

~AutoBuffer()

void Reset()

char* allocBuffer(int size)

char* getBuffer()

int getBufferSize()

AutoBuffer(const AutoBuffer&)

const AutoBuffer& operator= (const AutoBuffer&)

- HIAI_DATA_SP class auxiliary API:

class HIAI_DATA_SP:

HIAI_DATA_SP()

virtual ~HIAI_DATA_SP()

void setFrameIndex(unsigned long long index)

unsigned long long getFrameIndex()

void setFrameBuffer(void * frameBuff)

void * getFrameBuffer()

void setFrameSize(unsigned int size)

unsigned int getFrameSize()

8.2 List of APIs that Are Not Recommended

- class IDVPPCAPABILITY:

virtual ~IDVPPCAPABILITY(void)

virtual int process(dvppapi_ctl_msg *MSG) = 0

virtual int init() = 0

virtual int start() = 0

virtual int stop() = 0

- class IJPEGDAPI:

virtual ~IJPEGDAPI(void)

virtual int process(dvppapi_ctl_msg* MSG) = 0

virtual int init() = 0

virtual int start() = 0

virtual int stop() = 0

- class IJPEGEAPI:

virtual ~IJPEGEAPI(){}

virtual int process(dvppapi_ctl_msg* MSG) = 0

virtual int init() = 0

virtual int start() = 0

virtual int stop() = 0

- class IVDECAPI:

virtual ~IVDECAPI(void)

virtual int process(dvppapi_ctl_msg* MSG) = 0

virtual int init() = 0

virtual int start() = 0

virtual int stop() = 0

- class IVENCAPI:

virtual ~IVENCAPI(void)

virtual int process(dvppapi_ctl_msg* MSG) = 0

virtual int init() = 0

virtual int start() = 0

- class IVPCAPI:

virtual ~IVPCAPI(void)

virtual int process(dvppapi_ctl_msg* MSG) = 0

virtual int init() = 0

virtual int start() = 0

virtual int stop() = 0

8.3 Sample Usage of the DVPP Executor

The DVPP executor integrates the basic functions of six DVPP modules: VPC, JPEGE, JPEGD, VENC, VDEC, and PNGD.

8.3.1 Environment Preparation

Step 1 Copy the **sample_dvpp_hlt_ddk** executable file generated after compilation to a directory on the device side.

NOTE

The dependent .so files are as follows: **lib/device/libc_sec.so**, **libDvpp_api.so**, **libDvpp_jpeg_decoder.so**, **libDvpp_jpeg_encoder.so**, **libDvpp_png_decoder.so**, **libDvpp_vpc.so**, **libOMX_hisi_video_decoder.so**, **libOMX_hisi_video_encoder.so**, and **libslog.so**

By default, the .so files are in the **/usr/lib64** directory on the device side. You do not need to copy them separately.

Step 2 Save the data file to the directory where the executable file is located, and then run the following command. In the command, **paramList** is the parameter described in **8.3.2 Sample Input Parameters of the DVPP Executor**.

./sample_dvpp_hlt_ddk paramList

----End

8.3.2 Sample Input Parameters of the DVPP Executor

Parameter	Description
img_width	Input image width
img_height	Input image height

Parameter	Description
in_format	• For the VPC, the input image format is as follows: INPUT_YUV400, // 0 INPUT_YUV420_SEMI_PLANNER_UV, // 1 INPUT_YUV420_SEMI_PLANNER_VU, // 2 INPUT_YUV422_SEMI_PLANNER_UV, // 3 INPUT_YUV422_SEMI_PLANNER_VU, // 4 INPUT_YUV444_SEMI_PLANNER_UV, // 5 INPUT_YUV444_SEMI_PLANNER_VU, // 6 INPUT_YUV422_PACKED_YUYV, // 7 INPUT_YUV422_PACKED_UYVY, // 8 INPUT_YUV422_PACKED_YVYU, // 9 INPUT_YUV422_PACKED_VYUY, // 10 INPUT_YUV444_PACKED_YUV, // 11 INPUT_RGB, // 12, input image format: RGB888 INPUT_BGR, // 13, input image format: BGR888 INPUT_ARGB, // 14. The component sequence of an input image in this format during storage is similar to RGB888. **A** indicates transparency. INPUT_ABGR, // 15. The component sequence of an input image in this format during storage is similar to BGR888. **A** indicates transparency. INPUT_RGBA, // 16. The component sequence of an input image in this format during storage is similar to RGB888. **A** indicates transparency. INPUT_BGRA, // 17. The component sequence of an input image in this format during storage is similar to BGR888. **A** indicates transparency. INPUT_YUV420_SEMI_PLANNER_UV_10BIT, // 18 INPUT_YUV420_SEMI_PLANNER_VU_10BIT, // 19 • For the VDEC, the input stream format is as follows: – **0**: H.264 stream – **1**: H.265 stream • For the VENC, the output stream format is as follows: – **0**: H.265 Main Profile Level (Only slice streams are supported.) – **1**: H.264 Baseline Profile Level – **2**: H.264 Main Profile Level – **3**: H.264 High Profile Level • For the JPEGE, the input image formats are as follows: – **0**: JPGENC_FORMAT_UYVY – **1**: JPGENC_FORMAT_VYUY

Parameter	Description
	– **2**: JPGENC_FORMAT_YVYU – **3**: JPGENC_FORMAT_YUYV – **16**: JPGENC_FORMAT_NV12 – **17**: JPGENC_FORMAT_NV21
out_format	Output image format of the VPC • **0**: OUTPUT_YUV420SP_UV • **1**: OUTPUT_YUV420SP_VU
yuvStoreType	Storage format of the input YUV data of the VENC • **0**: YUV420 semi-planar • **1**: YVU420 semi-planar
inWidthStride	Width stride of the VPC input image. For details, see **Input Parameter: VpcUserImageConfigure**.
inHighStride	Height stride of the VPC input image. For details, see **Input Parameter: VpcUserImageConfigure**.
outWidthStride	Width stride of the VPC output image, which is 16-pixel aligned
outHighStride	Height stride of the VPC output image, which is 2-pixel aligned
hmin	Left offset of the cropped area specified by the VPC user. The value must be an even number.
hmax	Right offset of the cropped area specified by the VPC user. The value must be an odd number.
vmin	Top offset of the cropped area specified by the VPC user. The value must be an even number.
vmax	Bottom offset of the cropped area specified by the VPC user. The value must be an odd number.
outLeftOffset	Left offset of the output area specified by the VPC user. The value must be an even number.
outRightOffset	Right offset of the output area specified by the VPC user. The value must be an odd number.
outUpOffset	Top offset of the output area specified by the VPC user. The value must be an even number.
outDownOffset	Bottom offset of the output area specified by the VPC user. The value must be an odd number.
out_width	Width of the output image
out_high	Height of the output image

Parameter	Description
in_image_file	Path of the input image file, which includes the file name
out_image_file	Path of the output image file, which includes the file name
rank	• **0**: The JPEGD outputs YUV420 semi-planar data. • **1**: The JPEGD outputs original YUV format data.
transform	Transform flag when the PNGD processes images • **1**: RGBA is converted to RGB. • **0**: The original format is retained.
test_type	The options are as follows: • **1**: VPC basic function 1 (image cropping and resizing) • **2**: VPC basic function 2 (original image resizing, cropped image resizing, single-image multi-ROI cropping, and 8K image resizing) • **3**: VDEC • **4**: VENC • **5**: JPEGE • **6**: The JPEGE user specifies the output buffer. • **7**: JPEGD • **8**: The JPEGD user specifies the output buffer. • **9**: PNGD • **10**: The PNGD user specifies the output buffer. • **12**: JPEGD+VPC+JPEGE

8.3.3 VPC Instructions

8.3.3.1 VPC Basic Function 1

```
./sample_dvpp_hlt_ddk --in_image_file dvpp_vpc_1920x1080_nv12.yuv --out_image_file vpc_out.yuv --inWidthStride 1920 --inHighStride 1080 --in_format 1 --out_format 0 --hmax 1919 --hmin 0 --vmax 999 --vmin 0 --outLeftOffset 0 --outRightOffset 719 --outUpOffset 0 --outDownOffset 719 --outWidthStride 720 --outHighStride 720 --test_type 1
```

Call the VPC to crop and resize an image.

- Input image:
  - An image whose width is 1920 pixels, height is 1080 pixels, format is YUV420SP, and file name is **dvpp_vpc_1920x1080_nv12.yuv**.
  - The input image is in the same directory as the executable file.
- Output image:
  - An image whose width is 720 pixels, height is 720 pixels, format is YUV420SP (NV12), and file name is **vpc_out.yuv**.

  - The output image is in the same directory as the executable file.
- You can open the **vpc_out.yuv** image by using image display software to verify the correctness.

8.3.3.2 VPC Basic Function 2

```
./sample_dvpp_hlt_ddk --test_type 2
```

This command calls the VPC to resize the original image, resize the cropped image, perform single-image multi-ROI cropping, and resize an 8K image.

- Input images:
  - An image whose width is 1920 pixels, height is 1080 pixels, format is YUV420SP NV12, and file name is **dvpp_vpc_1920x1080_nv12.yuv**.
  - An image whose width is 8192 pixels, height is 8192 pixels, format is YUV420SP NV12, and file name is **dvpp_vpc_8192x8192_nv12.yuv**.
  - The input images are in the same directory as the executable file.
- Output images:
  - Call the **NewVpcTest1** function to resize the 1080p YUV420SP image to 1280 x 720 YUV420SP (NV12). The output file name is **NewVpcTest1Out.yuv**.
  - Call the **NewVpcTest2** function to crop a sub-image from the 1080p YUV420SP image, and attach the sub-images to a specified location in the output image. The output file name is **NewVpcTest2Out.yuv**, the output image size is 1280 x 720, and the output image format is YUV420SP.
  - Call the **NewVpcTest3** function to crop five sub-images from the 1080p YUV420SP image, and attach the su-images to specified locations in the output image. The output file name is **NewVpcTest3_*Out.yuv**. The output image size is 1280 x 720 and the output image format is YUV420SP.
  - Call the **NewVpcTest4** function to resize the 8192 x 8192 YUV420SP image to a 4000 x 4000 YUV420SP image. The output file name is **NewVpcTest4Out.yuv**.
  - The generated images and the executable file are in the same directory.
- You can open the images by using image display software to verify the correctness.

8.3.4 VDEC Usage

```
./sample_dvpp_hlt_ddk --in_image_file dvpp_vdec_h264_1frame_bp_51_1920x1080.h264 --out_image_file h264_image --in_format 0 --test_type 3
./sample_dvpp_hlt_ddk --in_image_file dvpp_vdec_h265_1frame_mp_51_1920x1080.h265 --out_image_file h265_image --in_format 1 --test_type 3
```

Sample description: The preceding commands call the VDEC to decode H.264 and H.265 streams and call the VPC to generate YUV images.

- Input video:
  - The stream has the width of 1920 pixels, height of 1080 pixels, and file name of **dvpp_vdec_h264_1frame_bp_51_1920x1080.h264**.
  - The stream has the width of 1920 pixels, height of 1080 pixels, and file name of **dvpp_vdec_h265_1frame_mp_51_1920x1080.h265**.

  - The video files are in the same directory as the executable file.
- Output image:
  - The output image is in YUV420SP format and has the width of 1920 pixels and height of 1080 pixels.
  - For the decoding of H.264 streams, an image named **h264_image*** is generated in the **output_dir** directory. Only one image is output because the current video contains only one frame. If the video contains multiple frames, multiple images will be output.
  - For the decoding of H.265 streams, an image named **h265_image*** is generated in the **output_dir** directory. Only one image is output because the current video contains only one frame. If the video contains multiple frames, multiple images will be output.
- You can open the **h264_image*** and **h265_image*** files by using image display software to verify the correctness.

8.3.5 Usage of the VENC

```
./sample_dvpp_hlt_ddk --in_image_file dvpp_venc_128x128_nv12.yuv --img_width 128 --img_height 128 --in_format 0 --yuvStoreType 0 --test_type 4
```

This command calls the VENC to encode a YUV image into a H.265 stream.

- Input image:
  - An image whose width is 128 pixels, height is 128 pixels, format is YUV420SP NV12, and file name is **dvpp_venc_128x128_nv12.yuv**.
  - The input image file is in the same directory as the executable file.
  - **in_format** corresponds to **coding_type** in **Input parameter: VencConfig**.
  - **yuvStoreType** corresponds to **YUV_store_type** in **Input parameter: VencConfig**.
- Output video:
  - The output stream file is **venc.bin**.
  - The output stream file is in the same directory as the executable file.
- You can run the **ffmpeg.exe -i venc.bin venc.jpg** command to decode the **venc.bin** file by using FFmpeg, and then open the image by using image display software to check the correctness. The official FFmpeg website is **http://ffmpeg.org/**.

8.3.6 Usage of the JPEGE

```
./sample_dvpp_hlt_ddk --in_image_file dvpp_jpege_444_1920x1080_1920_1080_nv12.yuv --img_width 1920 --img_height 1080 --in_format 16 --test_type 5
./sample_dvpp_hlt_ddk --in_image_file dvpp_jpege_444_1920x1080_1920_1080_nv12.yuv --img_width 1920 --img_height 1080 --in_format 16 --test_type 6
```

Sample description: In the first command (**test_type 5**), the DVPP allocates memory. In the second command (**test_type 6**), the user allocates memory. Both commands can be used to encode a YUV image into a JPG image.

- Input image:
  - The input image has the width of 1920 pixels, height of 1080 pixels, format of YUV420SP NV12, and file name of **dvpp_jpege_444_1920x1080_1920_1080_nv12.yuv**.

  - The input image is in the same directory as the executable file.
- Output image:
  - The output image has the width of 1920 pixels, height of 1080 pixels, and file name of **outfile_t*X*.jpg** (*X* indicates the thread ID).
  - The output image is in the same directory as the executable file.

8.3.7 Usage of the JPEGD

```
./sample_dvpp_hlt_ddk --in_image_file dvpp_jpegd_decode_1920x1080.jpg --test_type 7 --rank 0
./sample_dvpp_hlt_ddk --in_image_file dvpp_jpegd_decode_1920x1080.jpg --test_type 8 --rank 0
```

In the first command (**test_type 7**), the DVPP allocates memory. In the second command (**test_type 8**), the user allocates memory. Both commands decode a JPG image into a YUV image.

- Input image:
  - An image whose width is 1920 pixels, height is 1080 pixels, and file name is **dvpp_jpegd_decode_1920x1080.jpg**.
  - The input image is in the same directory as the executable file.
  - If **--rank** is set to **0** or not set, the output data format is **NV21**. If **-- rank** is not **0**, the output data is the semi-planar data of the original format.
- Output image:
  - An image whose width is 1920 pixels, height is 1080 pixels, and file name is **dvppapi_jpegd_wxx_hxx_fx_tx.yuv**.
  - The output image is in the same directory as the executable file.
- You can open the output YUV image by using image display software to verify the correctness.

8.3.8 Usage of PNGD

```
./sample_dvpp_hlt_ddk --in_image_file dvpp_pngd_1920x1080_RGBA.png --out_image_file
pngd_out_RGBAtoRGB_1.rgb --test_type 9  --transform 1
./sample_dvpp_hlt_ddk --in_image_file dvpp_pngd_1920x1080_RGBA.png --out_image_file
pngd_out_RGBAtoRGB_2.rgb --test_type 10 --transform 1
```

In the first command (**test_type 9**), the DVPP allocates memory. In the second command (**test_type 10**), the user allocates memory. Both commands decode a PNG image into an RGB888 image.

- Input picture:
  - An image whose width is 1920 pixels, height is 1080 pixels, and file name is **dvpp_pngd_1920x1080_RGBA.png**.
  - The input image is in the same directory as the executable file.
  - If **--transform** is set to **1**, the RGBA PNG image is converted to the RGB888 format.
- Output image:
  - Two 1920 x 1080 images. The file name of the first output image (**test_type 9**) is **pngd_out_RGBAtoRGB_1.rgb**, and the file name of the second output image (**test_type 10**) is **pngd_out_RGBAtoRGB_2.rgb**.
  - The output images are in the same directory as the executable file.

- You can open the **pngd_out_RGBAtoRGB_1.rgb** and **pngd_out_RGBAtoRGB_2.rgb** files by using image display software to verify the correctness.

8.3.9 JPEGD+VPC+JPEGE Cascading

```
./sample_dvpp_hlt_ddk --in_image_file dvpp_jpegd_vpc_jpege.jpg --out_image_file
out_jpegd_vpc_jpege_new.jpg --hmax 1919 --hmin 0 --vmax 999 --vmin 0 --outLeftOffset 0 --
outRightOffset 719 --outUpOffset 0 --outDownOffset 719 --outWidthStride 720 --outHighStride 720 --
test_type 12
```

Sample description: This command calls the JPEGD, VPC, and JPEGE in sequence. **test_type 12** indicates that the user allocates memory. This command calls the JPEGD to decode a JPG image as a YUV420SP image, sends the image to the VPC for cropping and resizing, and calls the JPEGE to encode the VPC output to generate a JPG image.

- Input image:
  - The input image has the width of 1920 pixels, height of 1080 pixels, and file name of **dvpp_jpegd_vpc_jpege.jpg**.
  - The input image is in the same directory as the executable file.
- Output image:
  - The output image has the width of 720 pixels, height of 720 pixels, and file name of **out_jpegd_vpc_jpege_new.jpg**.
  - The output image is in the same directory as the executable file.

8.4 Exception Handling

If the caller fails to call **DvppCtl** or **DvppGetOutParameter** (the return value is **–1**), you can view the log in the **Log** window of Mind Studio. Select **DVPP** from the **Module Name** drop-down list box, and click **Search** to query the log. You can view the latest log based on the **Time** column and rectify the calling error accordingly.

For example, when the caller uses the VPC function, if the width and height of the input image are not 128 x 16 aligned, the following information is displayed in the **Content** column:

```
The input image width should be divided by 128, high should be divided by 16, now width 512, high 456.
```

NOTE

For details about how to view a log, see section 2.3.1 "Viewing a Log" in *Ascend 310 Mind Studio Auxiliary Tools*.

8.5 Acronyms

Table 8-1 Acronyms

Acronym	Full Name
VDEC	video decoder

Acronym	Full Name
VENC	video encoder
JPEGD	JPEG decoder
JPEGE	JPEG encoder
PNGD	PNG decoder
VPC	vision preprocessing core
DVPP	digital vision pre-processing
HFBC	HiSilicon frame buffer compression

8.6 Change History

Release Date	Description
2020-05-30	This issue is the first official release.

