



HiMPP IPC V2.0 Media Processing Software

Development Reference

Issue	12
Date	2016-11-24

Copyright © HiSilicon Technologies Co., Ltd. 2014-2016. All rights reserved.

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of HiSilicon Technologies Co., Ltd.

Trademarks and Permissions



, **HISILICON**, and other HiSilicon icons are trademarks of HiSilicon 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 HiSilicon 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.

HiSilicon Technologies Co., Ltd.

Address: Huawei Industrial Base
 Bantian, Longgang
 Shenzhen 518129
 People's Republic of China

Website: <http://www.hisilicon.com>

Email: support@hisilicon.com



Contents

About This Document.....	i
--------------------------	---



About This Document

Purpose

This document provides reference information for the programmers that develop products or solutions by using the HiSilicon media processing platform (HiMPP). The information includes application programming interfaces (APIs), header files, and error codes.

This document also describes the method of calling each API, and the definitions of data structures and error codes.

NOTE

- This document uses the Hi3516A as an example. Unless otherwise specified, the contents of the Hi3516A also apply to the Hi3516D and Hi3518E V200.
- Unless otherwise specified, the contents of Hi3518E V200 also apply to Hi3518E V201 and Hi3516C V200.

Related Versions

The following table lists the product versions related to this document.

Product Name	Version
Hi3516A	V100
Hi3516D	V100
Hi3518E	V200
Hi3518E	V201
Hi3516C	V200
Hi3519	V100

Intended Audience

This document is intended for:

- Technical support engineers
- Software development engineers



Conventions

Symbol Conventions

The symbols that may be found in this document are defined as follows:

Symbol	Description
DANGER	Indicates a hazard with a high level of risk which, if not avoided, will result in death or serious injury.
WARNING	Indicates a hazard with a medium or low level of risk that, if not avoided, could result in minor or moderate injury.
CAUTION	Indicates a potentially hazardous situation, which if not avoided, could result in equipment damage, data loss, performance degradation, or unexpected results.
TIP	Indicates a tip that may help you solve a problem or save time.
NOTE	Provides additional information to emphasize or supplement important points of the main text.

General Conventions

The general conventions that may be found in this document are defined as follows:

Convention	Description
Times New Roman	Normal paragraphs are in Times New Roman.
Boldface	Names of files, directories, folders, and users are in boldface . For example, log in as user root .
<i>Italic</i>	Book titles are in <i>italics</i> .
Courier New	Examples of information displayed on the screen are in Courier New.

Command Conventions

The command conventions that may be found in this document are defined as follows:

Convention	Description
Boldface	The keywords of a command line are in boldface .
<i>Italic</i>	Command arguments are in <i>italics</i> .



Convention	Description
[]	Items (keywords or arguments) in square brackets ([]) are optional.
{ x y ... }	Optional items are grouped in braces and separated by vertical bars. One item is selected.
[x y ...]	Optional items are grouped in brackets and separated by vertical bars. One item is selected or no item is selected.
{ x y ... } *	Optional items are grouped in braces and separated by vertical bars. A minimum of one item or a maximum of all items can be selected.
[x y ...] *	Optional items are grouped in brackets and separated by vertical bars. Several items or no item can be selected.

Numerical System

The expressions of data capacity, frequency, and data rate are described as follows:

Type	Symbol	Value
Data capacity (such as the RAM capacity)	K	1024
	M	1,048,576
	G	1,073,741,824
Frequency, data rate	k	1000
	M	1,000,000
	G	1,000,000,000

The expressions of addresses and data are described as follows:

Symbol	Example	Description
0x	0xFE04, 0x18	Address or data in hexadecimal
0b	0b000, 0b00 00000000	Data or sequence in binary (register description is excluded.)
X	00X, 1XX	In data expression, X indicates 0 or 1. For example, 00X indicates 000 or 001 and 1XX indicates 100, 101, 110, or 111.



Change History

Changes between document issues are cumulative. Therefore, the latest document issue contains all changes made in previous issues.

Issue 12 (2016-11-24)

This issue is the twelfth official release, which incorporates the following changes:

Chapter 2 System Control

In section 2.3, HI_MPI_SYS_SetTimeZone and HI_MPI_SYS_GetTimeZone are added.

Chapter 3 VI

In section 3.4, HI_MPI_VI_Query and HI_MPI_VI_SetCSCAttr are modified.

In section 3.5, VIU_DEV_COMP_MASK_NUM to VIU_CSC_COEF_NUM are added. VI_DEV_ATTR_S, VI_DEV_ATTR_EX_S, VI_CHN_STAT_S, LDC_VIEW_TYPE_E, VI_CSC_TYPE_E, VI_EXT_CHN_ATTR_S, and VI_DCI_PARAM_S are modified. VI_CSC_MATRIX_S is added.

Chapter 4 VO

In section 4.3, the description in the **Note** field of HI_MPI_VO_SetChnReceiveThreshold is modified.

In section 4.4, the description in the **Note** field of VO_VIDEO_LAYER_ATTR_S is modified.

Chapter 5 VPSS

In section 5.2.3, table 5-2 is modified.

In section 5.4, VPSS_EXTCHN_MAX_IMAGE_WIDTH, VPSS_EXTCHN_MAX_IMAGE_HEIGHT, LDC_VIEW_TYPE_E, VPSS_CHN_ATTR_S, and VPSS_LDC_ATTR_S are modified.

Chapter 6 VENC

Section 6.2.14 is modified.

In section 6.4, VENC_PARAM_H264_CBR_S and VENC_PARAM_H265_CBR_S are modified.

Chapter 8 Region Management

Section 8.2 and section 8.3 are modified.

In section 8.4, RGN_CANVAS_INFO_S is modified.

Chapter 9 Audio

In section 9.3.1, HI_MPI_AI_QueryFileStatus is added.

In section 9.3.2, HI_MPI_AO_QueryFileStatus is added.

In section 9.3.3, HI_MPI_AENC_QueryFileStatus is added.

In section 9.4.1, VQE_PEQ_BAND_NUM and AUDIO_FILE_STATUS_S are added.

Chapter 10 VGS



In section 10.2.2, table 10-1 and table 10-3 are modified.

Chapter 12 Proc Debugging Information

Sections 12.9, 12.12, 12.16, and 12.17 are modified.

Chapter 12 Proc2 Debugging Information

Section 12.4 is modified.

Issue 11 (2016-06-30)

This issue is the eleventh official release, which incorporates the following changes:

Chapter 2 System Control

In section 2.4.1, the common data structures are modified.

In section 2.4.3, the description in the **Note** field of VIDEO_SUPPLEMENT_S is modified.

Chapter 3 VI

Section 3.2 is modified.

In section 3.4, the descriptions in the **Note** fields of HI_MPI_VI_SetChnAttr, HI_MPI_VI_SetRotate, HI_MPI_VI_SetDevDumpAttr, HI_MPI_VI_GetFrame, and HI_MPI_VI_SetDISConfig are modified. The descriptions in the **Parameter** fields of HI_MPI_VI_SetFisheyeDevConfig and HI_MPI_VI_GetFisheyeDevConfig are modified. HI_MPI_VI_SendBayerDataEx is added.

In section 3.5, the description in the **Difference** field of VIU_DEV_MAX_HEIGHT is modified. The descriptions in the **Description**, **Member**, **Note**, and **See Also** fields of VI_DUMP_ATTR_S are modified. The descriptions in the **Member** and **Note** fields of VI_DIS_CONFIG_S are modified. The description in the **Member** field of VI_DIS_ATTR_S is modified. The descriptions in the **Description** and **Member** fields of VI_DIS_MOTION_TYPE_E and VI_DIS_GYRO_DATA_S are modified. The description in the **Note** field of VI_DIS_CONFIG_S is modified.

Chapter 4 VO

In section 4.4, the description in the **Note** field of VO_ZOOM_ATTR_S is modified.

Chapter 5 VPSS

In section 5.3, the descriptions in the **Note** fields of HI_MPI_VPSS_GetFisheyeConfig, HI_MPI_VPSS_GetFisheyeAttr, HI_MPI_VPSS_SetGrpNRSPParam, and HI_MPI_VPSS_SetGrpNRBParam are modified. The description in the **Difference** field of HI_MPI_VPSS_GetFisheyeAttr is modified.

In section 5.4, the description in the **Note** field of VPSS_CROP_INFO_S is modified. The descriptions in the **Description**, **Member**, and **Note** fields of VPSS_GRP_NRS_PARAM_S and VPSS_GRP_NRB_PARAM_S are modified.

Chapter 6 VENC

Section 6.2.14 is modified.

In section 6.3, the description in the **Note** field of HI_MPI_VENC_CreateChn is modified.

Chapter 6 VENC2

The contents related to Hi3516C V300 are added.



Section 6.2.15 is added.

Section 6.2.10 and section 6.2.16 are modified.

In section 6.3, the descriptions in the **Note** fields of HI_MPI_VENC_CreateChn, HI_MPI_VENC_GetStream, HI_MPI_VENC_SetFrameRate, HI_MPI_VENC_SetIntraRefresh, HI_MPI_VENC_EnableAdvSmartP, HI_MPI_VENC_SetRcParam, and HI_MPI_VENC_SetFrameRatem are modified. The descriptions in the **Description** and **Note** fields of HI_MPI_VENC_SendFrameEx are modified.

In section 6.4, the description in the **Note** field of VENC_ATTR_H264_CBR_S is modified. The descriptions in the **Member** fields of VENC_PARAM_INTRA_REFRESH_S and VENC_PARAM_MOD_VENC_S are modified. The descriptions in the **Description** and **Member** fields of VENC_PARAM_H264_CBR_S, VENC_PARAM_H265_CBR_S, and VENC_RC_PARAM_S are modified. The descriptions of VENC_PARAM_H264_VBR_S, VENC_PARAM_H264_CBR_S, VENC_PARAM_H265_VBR_S, VENC_PARAM_H265_CBR_S, VENC_RC_PARAM_S, and VENC_PARAM_MOD_VENC_S are modified.

Chapter 8 Region Management

In section 8.4, the description in the **Note** field of OVERLAY_CHN_ATTR_S is modified.

Chapter 9 Audio

Section 9.2.1.5 is modified.

In section 9.3, the descriptions in the **Requirement** fields of HI_MPI_AI_SetHiFiVqeAttr, HI_MPI_AI_GetHiFiVqeAttr, HI_MPI_AI_EnableVqe, and HI_MPI_AI_DisableVqe are modified.

In section 9.4, AI_HIFIVQE_MASK_AVC is replaced with AI_HIFIVQE_MASK_DRC, and the description of this data structure is modified. AI_AVG_CONFIG_S is replaced with AI_DRC_CONFIG_S, and the description of this data structure is modified. The descriptions in the **Description** and **Note** fields of AI_HIFIVQE_CONFIG_S are modified.

In section 9.4.1, VQE_DRC_SECNUM is added.

Chapter 10 VGS

In section 10.2.2, table 10-3 is modified.

Chapter 11 Fisheye Subsystem

In section 11.3, the description in the **Note** field of fisheye_mod_init is modified.

In section 11.4, the description in the **Note** field of FISHEYE_CONFIG_S is modified.

Chapter 12 Proc Debugging Information

Sections 12.12, 12.14, 12.16, and 12.21 are modified.

Chapter 12 Proc2 Debugging Information

Section 12.2, 12.4, and 12.5 is modified.

Issue 10 (2016-05-13)

This issue is the tenth official release, which incorporates the following changes:

Chapter 2 System Control



In section 2.3, HI_MPI_SYS_SetScaleCoefLevel and HI_MPI_SYS_GetScaleCoefLevel are added.

In section 2.4.2, SCALE_RANGE_E, OEFF_LEVEL_E, SCALE_RANGE_S, and SCALE_COEFF_LEVEL_S are added.

Chapter 3 VI

In section 3.4, HI_MPI_VI_SetLDCAttr and HI_MPI_VI_GetLDCAttr are modified.

In section 3.5, VI_DIS_CONFIG_S is modified and VI_DIS_ANGLE_TYPE_E is added.

Chapter 4 VO

In section 4.4, the description in the **Note** field of VO_PUB_ATTR_S is modified.

Chapter 5 VPSS

In section 5.3, the description in the **Note** field of HI_MPI_VPSS_GetGrpFrame is modified. HI_MPI_VPSS_SetChnParam, HI_MPI_VPSS_GetChnParam, and HI_MPI_VPSS_SetLDCAttr are modified.

In section 5.4, the description in the **Note** field of VPSS_LDC_ATTR_S is modified.

Chapter 6 VENC

Section 6.2.14 is added.

In section 6.3, HI_MPI_VENC_CreateChn is modified.

In section 6.4, the descriptions in the **Member** fields of VENC_ATTR_H264_S, VENC_ATTR_MJPEG_S, VENC_ATTR_MPEG4_S, and VENC_ATTR_H265_S are modified.

Chapter 8 Region Management

Section 8.2.1 is modified, and figure 8-2 is added.

Chapter 9 Audio

In section 9.4.1, AUDIO_AGC_CONFIG_S, AI_AEC_CONFIG_S, AUDIO_ANR_CONFIG_S, AI_RNR_CONFIG_S, AI_AVG_CONFIG_S, and AI_PEQ_CONFIG_S are modified.

Chapter 12 Proc Debugging Information

Section 12.2, section 12.7, section 12.12, section 12.14, and section 12.20 are modified.

Chapter 12 Proc2 Debugging Information

Section 12.2, section 12.3, section 12.4, section 12.5, and section 12.7 are modified.

Issue 09 (2016-01-29)

This issue is the ninth official release, which incorporates the following changes:

Chapter 2 System Control

In section 2.3, HI_MPI_SYS_MmapCache, HI_MPI_VB_SetSupplementConf, HI_MPI_VB_GetSupplementConf, HI_MPI_VB_GetSupplementAddr, and HI_MPI_SYS_MflushCache are added.

In section 2.4.1, common data structures are modified.



In section 2.4.3, the descriptions in the **Syntax** and **Member** fields of VIDEO_FRAME_S are modified. FRAME_FLASH_TYPE_E and VB_SUPPLEMENT_CONF_S are added.

Chapter 3 VI

Section 3.2 is modified.

In section 3.3, figure 3-3 is added and figure 3-4 is changed. The DIS software process is modified.

In section 3.4, the descriptions in the **Difference** and **Note** fields of HI_MPI_VI_SetChnAttr, HI_MPI_VI_SetLDCAttr, and HI_MPI_VI_SetExtChnAttr are modified. The descriptions in the **Note** fields of HI_MPI_VI_SetCSCAttr and HI_MPI_VI_SetExtChnCrop are modified.

HI_MPI_VI_SetDISConfig, HI_MPI_VI_GetDISConfig, HI_MPI_VI_DISInit, HI_MPI_VI_SetDISAttr, HI_MPI_VI_GetDISAttr, HI_MPI_VI_DISRun, HI_MPI_VI_DISExit, HI_MPI_VI_RegisterDISCallback, HI_MPI_VI_UnRegisterDISCallback, HI_MPI_VI_SetModParam, and HI_MPI_VI_GetModParam are added.

In section 3.5, VI_DIS_CONFIG_S, VI_DIS_ATTR_S, VI_DIS_ACCURACY_E, VI_DIS_CAMERA_MODE_E, VI_DIS_MOTION_TYPE_E, VI_DIS_GYRO_DATA_S, VI_DIS_CALLBACK_S, VIU_DEV_MIN_WIDTH, VIU_DEV_MIN_HEIGHT, VIU_DEV_MAX_WIDTH, VIU_DEV_MAX_HEIGHT, VIU_CHN_MIN_WIDTH, VIU_CHN_MIN_HEIGHT, VIU_CHN_MAX_WIDTH, VIU_CHN_MAX_HEIGHT, VIU_EXTCHN_MIN_WIDTH, VIU_EXTCHN_MIN_HEIGHT, VIU_EXTCHN_MAX_WIDTH, VIU_EXTCHN_MAX_HEIGHT, and VIU_EXTCHN_MINIFICATION are added.

The descriptions in the **Member** and **Difference** fields of VI_DEV_ATTR_S and VI_DEV_ATTR_EX_S are modified. The descriptions in the **Member**, **Note**, and **Difference** fields of VI_CHN_ATTR_S are modified. The descriptions in the **Member**, **Difference**, and **See Also** fields of VI_EXT_CHN_ATTR_S are modified.

The descriptions in the **Note** fields of LDC_ATTR_S and VI_DIS_MOTION_TYPE_E are modified.

The descriptions in the **Syntax** and **Member** fields of VI_CSC_ATTR_S are modified.

VI_MOD_PARAM_S is added.

Chapter 4 VO

In section 4.4, the description in the **Difference** field of VO_VIDEO_LAYER_ATTR_S is modified.

Chapter 5 VPSS

In section 5.2.1, the descriptions of FRC and Crop are modified.

In section 5.2.3, table 5-1 and figure 5-3 are modified.

In section 5.3, HI_MPI_VPSS_SetModParam, HI_MPI_VPSS_GetModParam, HI_MPI_VPSS_SetNRBParam, HI_MPI_VPSS_SetNRV3Param, HI_MPI_VPSS_GetNRV3Param, and HI_MPI_VPSS_GetNRBParam are added.

The descriptions in the **Difference** fields of HI_MPI_VPSS_SetLDCAttr, HI_MPI_VPSS_SetNRBParam, and HI_MPI_VPSS_GetNRBParam are modified.

The description in the **Note** field of HI_MPI_VPSS_SetRotate is modified.



In section 5.4, VPSS_MIN_IMAGE_WIDTH, VPSS_MIN_IMAGE_HEIGHT, VPSS_MAX_IMAGE_WIDTH, VPSS_MAX_IMAGE_HEIGHT, VPSS_OFFLINE_MAX_IMAGE_WIDTH, VPSS_ONLINE_MAX_IMAGE_WIDTH, VPSS_EXTCHN_MAX_IMAGE_WIDTH, VPSS_EXTCHN_MAX_IMAGE_HEIGHT, VPSS_MAX_ZOOMIN, VPSS_MAX_ZOOMOUT, VPSS_EXT_CHN_MAX_ZOOMIN, VPSS_EXT_CHN_MAX_ZOOMOUT, VPSS_PARAM_MOD_S, VPSS_GRP_VPPNRB19CORE_S, VPSS_GRP_VPPNRBCORE_S, VPSS_GRP_VPPNRBEX_S, and VPSS_GRP_VPPNRB_S are added.

The description in the **Member** field of VPSS_GRP_ATTR_S is modified.

The description in the **Note** field of VPSS_LDC_ATTR_S is modified.

The descriptions in the **Syntax**, **Member**, and **Note** fields of VPSS_GRP_VPPNRB19CORE_S and VPSS_GRP_VPPNRB_S are modified.

The descriptions in the **Syntax** and **Member** fields of VPSS_NR_PARAM_V1_S are modified.

Chapter 6 VENC

In section 6.3, the description in the **Note** field of HI_MPI_VENC_SendFrame is modified. HI_MPI_VENC_SetRefParamEx and HI_MPI_VENC_GetRefParamEx are added.

In section 6.4, VENC_PARAM_REF_EX_S is added. The descriptions in the **Member** fields of VENC_ROI_CFG_S, VENC_PARAM_REF_EX_S, and VENC_PARAM_INTRA_REFRESH_S are modified.

Chapter 6 VENC_Hi3519

In section 6.2.3, the content related to QPMAP is added.

In section 6.2.4.table 6-2 and figure 6-5 are modified.

Section 6.2.5 is modified.

In section 6.3, the description in the **Note** field of HI_MPI_VENC_SetRefParam, HI_MPI_VENC_GetChnAttr, HI_MPI_VENC_EnableIDR, HI_MPI_VENC_GetRoiBgFrameRate, HI_MPI_VENC_SetFrameLostStrategy, HI_MPI_VENC_GetFrameLostStrategy, and HI_MPI_VENC_GetIntraRefresh are modified. The description in the **Member** field of VENC_GOP_DUALP_S is modified. The descriptions in the **Syntax** and **Member** fields of VENC_ATTR_H264_QPMAP_S are modified. The description in the **Syntax** fields of VENC_PARAM_H265_DBLOCK_S and VENC_PARAM_H265_SAO_S are modified. HI_MPI_VENC_SendFrameEx, HI_MPI_VENC_SetModParam, and HI_MPI_VENC_GetModParam are added.

In section 6.4, H264E_NALU_TYPE_E, H264E_REF_TYPE_E, H265E_NALU_TYPE_E, VENC_RC_ATTR_S, and VENC_RC_MODE_E are modified.

VENC_RC_QPMAP_MODE_E, VENC_ATTR_H264_QPMAP_S, and VENC_ATTR_H265_QPMAP_S are added.

VENC_PARAM_MOD_S to USER_FRAME_INFO_S are added.

Chapter 8 Region Management

In section 8.4, RGN_CHN_ATTR_U is modified.

Chapter 9 Audio

Section 9.2.1.5 is modified.



In section 9.3.1, the MPIs from HI_MPI_AI_SetHiFiVqeAttr to HI_MPI_AI_GetTalkVqeAttr are added.

In section 9.4.1, AO_VQE_MASK_HPF to AO_VQE_MASK_EQ, AI_HIFIVQE_MASK_HPF, AI_HIFIVQE_MASK_AEC, AI_HIFIVQE_MASK_HDR, AI_HIFIVQE_MASK_EQ, AI_AVG_CONFIG_S, VQE_PEQ_BAND_NUM, AI_PEQ_CONFIG_S, AI_HIFIVQE_CONFIG_S, AI_TALKVQE_CONFIG_S, AI_TALKVQE_MASK_ANR, AI_AVG_CONFIG_S, VQE_PEQ_BAND_NUM, AI_PEQ_CONFIG_S, AI_HIFIVQE_CONFIG_S, AI_TALKVQE_CONFIG_S, MAX_AUDIO_FILE_PATH_LEN, and MAX_AUDIO_FILE_NAME_LEN are added. The descriptions of AI_HDR_CONFIG_S, AO_VQE_CONFIG_S, AI_VQE_CONFIG_S, AI_VQE_CONFIG_S, AI_HIFIVQE_CONFIG_S, AI_TALKVQE_CONFIG_S, AO_VQE_CONFIG_S, and AUDIO_SAVE_FILE_INFO_S are modified.

Chapter 11 Fisheye Subsystem

Section 11.2.1 and section 11.2.2 are modified.

In section 11.2.2, the HiPQ Fish Eye Calibration Tool is added.

In section 11.4, the descriptions in the **Syntax** and **Member** fields of FISHEYE_REGION_ATTR_S are modified. The description in the **Member** field of FISHEYE_ATTR_S is modified.

Chapter 12 Proc Debugging Information

Sections 12.3, 12.4, 12.5, 12.7, 12.10, 12.11, 12.12, 12.13, 12.14, 12.16, 12.17, and 12.20 are modified.

Chapter 12 Proc2 Debugging Information

Sections 12.2, 12.4, 12.5, 12.6, and 12.7 are modified.

Issue 08 (2015-12-18)

This issue is the eighth official release, which incorporates the following changes:

Chapter 2 System Control

In section 2.3, the descriptions in the **Syntax** fields of HI_MPI_SYS_Bind, HI_MPI_SYS_UnBind, and HI_MPI_SYS_GetBindbyDest are modified.

In section 2.4, the descriptions in the **Syntax** fields of MOD_ID_E, PAYLOAD_TYPE_E, and VIDEO_FRAME_S are modified.

Chapter 3 VI

In section 3.4, the descriptions in the **Description** and **Syntax** fields of HI_MPI_VI_GetFisheyeDevConfig are modified. The description in the **Description** field of HI_MPI_VI_GetFisheyeAttr is modified. The descriptions in the **Difference** and **Note** fields of HI_MPI_VI_SetDevDumpAttr are modified. The description in the **Difference** field of HI_MPI_VI_GetDevDumpAttr is modified. The description in the **Note** field of HI_MPI_VI_EnableBayerDump is modified. The description in the **Parameter** field of HI_MPI_VI_SendBayerData is modified.

In section 3.5, the description in the **Syntax** field of VI_DEV_ATTR_EX_S is modified. The descriptions in the **Syntax** and **Member** fields of VI_CHN_STAT_S and VI_DATA_PATH_E are modified. The descriptions in the **Syntax**, **Member**, and **See Also** fields of VI_DUMP_ATTR_S are modified.

Chapter 4 VO



In section 4.3, the descriptions in the **Syntax** and **Parameter** fields of HI_MPI_VO_GetScreenFrame, HI_MPI_VO_ReleaseScreenFrame, HI_MPI_VO_GetChnAttr, and HI_MPI_VO_GetChnReceiveThreshold are modified.

In section 4.4, the name of VIDEO_FIELD_E is changed into VO_DISPLAY_FIELD_E. The descriptions in the **Syntax** and **Member** fields of VO_DISPLAY_FIELD_E are modified. The description in the **Note** field of HI_MPI_VO_GetChnFrame is modified. VO_ZOOM_IN_E is added.

Chapter 5 VPSS

In section 5.2.3, table 5-2 is modified.

In section 5.3, the descriptions in the **Note** fields of HI_MPI_VPSS_SetDepth, HI_MPI_VPSS_SetRotate, and HI_MPI_VPSS_SetLowDelayAttr are modified.

In section 5.4, the descriptions in the **Description** fields of VPSS_DIE_MODE_E, VPSS_CROP_COORDINATE_E, LDC_VIEW_TYPE_E, VPSS_CROP_INFO_S, VPSS_GRP_ATTR_S, VPSS_CHN_ATTR_S, VPSS_CHN_MODE_E, VPSS_CHN_MODE_S, VPSS_CHN_PARAM_S, VPSS_FRAME_RATE_S, VPSS_EXT_CHN_ATTR, and VPSS_REGION_INFO_S are modified.

The descriptions in the **Note** fields of VPSS_REF_SEL_MODE_E and VPSS_EXT_CHN_ATTR are modified.

Chapter 6 VENC

Section 6.3 and section 6.4 are modified.

Chapter 6 VENC2

In section 6.3, the descriptions in the **Note** fields of HI_MPI_VENC_SetChnAttr and HI_MPI_VENC_GetChnAttr are modified.

In section 6.4, the descriptions in the **Member** fields of VENC_PARAM_H265_TIMING_S and VENC_SUPERFRAME_CFG_S are modified.

Chapter 9 Audio

Section 9.3.1, section 9.3.2, and section 9.4.1 are modified.

In section 9.4.2, AENC_ATTR_LPCM_S is added.

In section 9.4.3, ADEC_ATTR_LPCM_S is added.

In section 9.5, table 9-9 to table 9-12 are modified.

Chapter 11 Fisheye Subsystem

Section 11.2.1 is modified.

Chapter 12 Proc Debugging Information

Sections 12.8, 12.11, 12.12, and 12.13 are modified.

Chapter 12 Proc2 Debugging Information

Section 12.6 is modified.

Issue 07 (2015-11-20)

This issue is the seventh official release, which incorporates the following changes:



Chapter 2 System Control

In section 2.3, the description in the **Note** field of HI_MPI_VB_SetConf is modified.

In section 2.4.2, the description in the **Note** field of VB_CONF_S is modified.

Chapter 3 VI

In section 3.4, the descriptions in the **Note** fields of HI_MPI_VI_SetLDCAttr, HI_MPI_VI_SetChnAttr, HI_MPI_VI_DisableChn, HI_MPI_VI_SetUserPic, and HI_MPI_VI_SetRotate are modified. The descriptions in the **Note** and **Difference** fields of HI_MPI_VI_SetUserPic are modified.

In section 3.5, the description in the **Note** field of VI_DEV_ATTR_S is modified.

Chapter 4 VO

Section 4.2 is modified.

In section 4.3, the description in the **Note** field of HI_MPI_VO_SendFrame is modified.

In section 4.4, the descriptions in the **Note** fields of VO_VIDEO_LAYER_ATTR_S, VO_ZOOM_ATTR_S, and VO_PUB_ATTR_S are modified.

Chapter 5 VPSS

Section 5.2.2 and section 5.2.3 are modified.

In section 5.3, the descriptions in the **Note** fields of HI_MPI_VPSS_SetChnMode, HI_MPI_VPSS_SetExtChnAttr, and HI_MPI_VPSS_SetLowDelayAttr are modified.

In section 5.4, the descriptions in the **Note** fields of VPSS_CROP_INFO_S and VPSS_CHN_ATTR_S are modified. The description in the **Member** field of VPSS_CHN_ATTR_S is modified.

Chapter 6 VENC2_Hi3519

Section 6.2.11.1 is modified.

In section 6.2.4, table 6-2 and figure 6-4 are modified.

In section 6.3, the descriptions in the **Note** fields of HI_MPI_VENC_SetRoiBgFrameRate, HI_MPI_VENC_GetRoiBgFrameRate, and HI_MPI_VENC_SetRcParam are modified.

In section 6.3, table 6-3 and table 6-6 are modified.

In section 6.4, the descriptions in the **Member** fields of VENC_ATTR_MJPEG_S, VENC_ATTR_H264_CBR_S, VENC_ATTR_MJPEG_CBR_S, VENC_ATTR_H265_CBR_S, VENC_GOP_SMARTP_S, VENC_ATTR_H264_FIXQP_S, VENC_ATTR_H265_FIXQP_S, and VENC_PARAM_H265_VBR_S are modified. The description in the **Syntax** field of VENC_PARAM_MJPEG_VBR_S is modified.

Chapter 6 VENC1

In section 6.3, the description in the **Note** field of HI_MPI_VENC_SetRoiBgFrameRate is modified.

In section 6.4, the descriptions in the **Member** fields of VENC_ATTR_H264_S and VENC_ATTR_H265_S are modified.

Chapter 8 Region Management

In section 8.2.1, table 8-3 is modified.



In section 8.4, RGN_COVER_MIN_X, RGN_COVER_MIN_Y, OVERLAYEX_MAX_NUM_VPSS, RGN_OVERLAY_MIN_X to RGN_OVERLAY_MAX_HEIGHT are added. The descriptions in the **Syntax** fields of RGN_MAX_WIDTH and RGN_MAX_HEIGHT are modified. The descriptions in the **Note** fields of RGN_QUADRANGLE_S and OVERLAY_ATTR_S are modified. The descriptions in the **Member** and **Note** fields of COVER_CHN_ATTR_S and COVEREX_CHN_ATTR_S are modified.

Chapter 9 Audio

Section 9.3.5 is modified.

In section 9.4.4, the description in the **Note** field of AO_CHN_STATE_S is modified.

Chapter 10 VGS

In section 10.2, table 10-3 is modified.

Section 10.4 is modified.

Chapter 11 Fisheye Subsystem

In section 11.2.1, the contents related to the 180° panoramic correction mode under table 11-2 are modified.

Section 11.2.2 is modified.

In section 11.4, the descriptions in the **Description** and **Member** fields of FISHEYE_REGION_ATTR_S are modified. The descriptions in the **Member** fields of RECT_S and FISHEYE_ATTR_S are modified. The description in the **Note** field of FISHEYE_CONFIG_S is modified.

Chapter 12 Proc Debugging Information

Section 12.6 is modified.

Section 12.14 is modified.

Chapter 12 Proc2 Debugging Information

Section 12.2 is modified.

Section 12.4 is modified.

Issue 06 (2015-10-20)

This issue is the sixth official release, which incorporates the following changes:

Chapter 3 VI

In section 3.4, the descriptions in the **Note** fields of HI_MPI_VI_SetChnAttr and HI_MPI_VI_SetLDCAttr are modified. The description in the **Difference** field of HI_MPI_VI_SetExtChnAttr is modified.

Chapter 4 VO

In section 4.4, VO_PUB_ATTR_S is modified.

Chapter 5 VPSS

In section 5.3, the description in the **Note** field of HI_MPI_VPSS_CreateGrp is modified.

Chapter 6 VENC



In section 6.3, the description in the **Note** field of HI_MPI_VENC_SetIntraRefresh is modified.

In section 6.4, the descriptions in the **Member** fields of VENC_ATTR_MJPEG_CBR_S, VENC_ATTR_MJPEG_VBR_S, VENC_PARAM_MPEG4_CBR_S, VENC_FRAME_RATE_S, VENC_ATTR_H265_S, VENC_ATTR_JPEG_S, and VENC_ATTR_MJPEG_S are modified. The description in the **Note** field of VENC_PARAM_INTRA_REFRESH_S is modified.

The description in section 6.2.13 is modified, and table 6-7 is added.

Chapter 6 VENC_Hi3519

In section 6.4, the descriptions in the **Member** fields of VENC_ATTR_MJPEG_CBR_S, VENC_ATTR_MJPEG_VBR_S, and VENC_FRAME_RATE_S are modified.

Chapter 8 Region Management

In section 8.2.1, table 8-3 is modified.

In section 8.4, the description in the **Note** field of RGN_QUADRANGLE_S is modified.

Chapter 9 Audio

In section 9.2.4, table 9-6 is modified.

In section 9.3.5.1, ACODEC_ENABLE_BOOSTL, ACODEC_ENABLE_BOOSTR, ACODEC_SET_PD_LINEINL, and ACODEC_SET_PD_LINEINR are added.

In section 9.3.5.2, ACODEC_SET_ADC_CLK is added.

In section 9.4.4, ACODEC_MIXER_E is modified.

Chapter 10 VGS

Section 10.2.1 is modified.

In section 10.2.2, table 10-3 is modified.

Chapter 11 Fisheye Subsystem

Section 11.2.2 is modified.

Chapter 12 Proc Debugging Information

The descriptions in sections 12.7, 12.8, 12.9, 12.11, 12.12, 12.14, and 12.20 are modified.

Issue 05 (2015-09-20)

This issue is the fifth official release, which incorporates the following changes:

The content related to Hi3519V100 is added.

Chapter 3 VI

In section 3.4, HI_MPI_VI_SetChnAttr and HI_MPI_VI_SetExtChnAttr are modified.

In section 3.5, VI_DEV_ATTR_S and VI_DEV_ATTR_EX_S are modified.

Chapter 4 VO

In section 4.3, HI_MPI_VO_GetScreenFrame, HI_MPI_VO_GetChnFrame, and HI_MPI_VO_ReleaseChnFrame are modified.



Chapter 5 VPSS

Section 5.1 is modified.

In section 5.3, the descriptions in the **Note** fields of HI_MPI_VPSS_CreateGrp, HI_MPI_VPSS_GetRegionLuma, HI_MPI_VPSS_DisableBackupFrame, and HI_MPI_VPSS_SetRotate are modified.

In section 5.4, VPSS_FRAME_WORK_E is deleted. VPSS_MAX_GRP_NUM, VPSS_GRP_ATTR_S, LDC_VIEW_TYPE_E, VPSS_CHN_ATTR_S, and VPSS_EXT_CHN_ATTR_S are modified.

Chapter 6 VENC

Section 6.2.3 is modified.

In section 6.3, table 6-2 is modified. HI_MPI_VENC_SetH264InterPred and HI_MPI_VENC_SetJpegSnapMode are modified.

In section 6.4, VENC_PARAM_MJPEG_CBR_S, VENC_ATTR_MJPEG_CBR_S, and VENC_PARAM_MJPEG_VBR_S are modified.

Chapter 7 VDA

Section 7.2.3 is modified.

Chapter 8 Region Management

In section 8.2.1, table 8-4 is added.

In section 8.4, RGN_MAX_X, RGN_MAX_Y, RGN_MAX_WIDTH, and RGN_MAX_HEIGHT are added. COVER_CHN_ATTR_S and COVEREX_CHN_ATTR_S are modified.

Chapter 9 Audio

In section 9.3.1, HI_MPI_AI_SetPubAttr is modified. HI_MPI_AI_EnableAecRefFrame and HI_MPI_AI_DisableAecRefFrame are added.

In section 9.3.4, HI_MPI_ADEC_GetFrame and HI_MPI_ADEC_ReleaseFrame are added.

In section 9.3.5, the descriptions in the **Note** fields of ACODEC_SET_INPUT_VOL and ACODEC_SET_GAIN_MICL are modified.

In section 9.4.1, AI_AEC_CONFIG_S and AI_VQE_CONFIG_S are modified. AI_HDR_CONFIG_S is added.

Chapter 10 VGS

In section 10.4, the description in the **Member** field of VGS_ADD OSD_S is modified.

Chapter 12 Proc Debugging Information

The descriptions in sections 12.6, 12.17, 12.18, and 12.19 are modified.

Issue 04 (2015-08-20)

This issue is the fourth official release, which incorporates the following changes:

Chapter 3 VI

Section 3.1 and 3.3 are modified.



In section 3.4, HI_MPI_VI_CloseFd, HI_MPI_VI_SetDevDumpAttr, and HI_MPI_VI_GetDevDumpAttr are added.

In section 3.5, VI_DUMP_ATTR_S and VI_DUMP_TYPE_E are added.

Chapter 4 VO

In section 4.3, the description in the **Note** field of HI_MPI_VO_SetChnReceiveThreshold is modified.

In section 4.4, the description in the **Member** field of VO_PUB_ATTR_S is modified.

Chapter 5 VPSS

Section 5.2.1 is modified.

In section 5.3, HI_MPI_VPSS_SetNRParam and HI_MPI_VPSS_GetNRParam are added.

HI_MPI_VPSS_SetGrpCrop and HI_MPI_VPSS_SendFrame are modified.

In section 5.4, VPSS_CROP_INFO_S and VPSS_GRP_PARAM_S are modified.

VPSS_NR_PARAM_U and VPSS_NR_PARAM_V1_S are added.

Chapter 6 VENC

Section 6.2.1 is modified;

Sections 6.2.11, 6.2.12, and 6.2.13 are added.

In section 6.3, HI_MPI_VENC_CloseFd is added.

The descriptions in the **Note** fields of HI_MPI_VENC_DestroyChn, HI_MPI_VENC_SetChnAttr, and HI_MPI_VENC_GetStream are modified.

In section 6.4, the descriptions in the **Member** fields of VENC_RC_ATTR_S and VENC_RC_MODE_E are added.

Chapter 9 Audio

The description in section 9.2.1.4 is added.

In section 9.3.1, HI_MPI_AI_SetVqeAttr is modified.

In section 9.3.5, ACODEC_SET_MIXER_MIC and ACODEC_MIXER_E are modified.

Section 9.3.5.1 and section 9.3.5.2 are modified.

In section 9.4.1, AUDIO_AGC_CONFIG_S, AUDIO_ANR_CONFIG_S, AUDIO_EQ_CONFIG_S, and AI_VQE_CONFIG_S are modified. Table 9-6 and table 9-7 are added.

Chapter 10 VGS

In section 10.3, HI_MPI_VGS_AddDrawLineTaskArray, HI_MPI_VGS_AddCoverTaskArray, and HI_MPI_VGS_AddOsdTaskArray are added.

Chapter 11 Proc Debugging Information

The descriptions in sections 11.6, 11.13, and 11.14 are modified.

Issue 03 (2015-05-27)

This issue is the third official release, which incorporates the following changes:



The contents related to Hi3518E V200, Hi3518E V201, and Hi3516C V200 are added.

Chapter 3 VI

Section 3.2 is modified.

In section 3.5, the description of VI_DCI_PARAM_S is modified.

Chapter 5 VPSS

In section 5.2.3, table 5-1 is modified.

In section 5.3, HI_MPI_VPSS_SetGrpParamV2 and HI_MPI_VPSS_GetGrpParamV2 are added.

In section 5.4, VPSS_GRP_PARAM_V2_S is added.

Chapter 6 VENC

Section 6.2.3 and section 6.3 are modified.

Chapter 9 Audio

Section 9.2 is modified.

In section 9.3.1, HI_MPI_AI_SetVqeVolume and HI_MPI_AI_GetVqeVolume are added.

In section 9.3.2, the MPIs from HI_MPI_AO_SetVqeAttr to HI_MPI_AO_DisableVqe are added.

In section 9.3.5, VQE_WORKSTATE_E, VQE_EQ_BAND_NUM, AUDIO_EQ_CONFIG_S, and AO_VQE_CONFIG_S are added.

In section 9.4.1, the descriptions in the **Member** and **Note** fields of AUDIO_AGC_CONFIG_S, AI_AEC_CONFIG_S, AUDIO_ANR_CONFIG_S, and AUDIO_HPF_CONFIG_S are modified.

In section 9.4.4, the description of ACODEC_MIXER_E is modified.

Chapter 11 Proc Debugging Information

Sections 11.3, 11.8, 11.14, 11.16, and 11.17 are modified.

Issue 02 (2015-02-10)

This issue is the second official release, which incorporates the following changes:

Chapter 2 System Control

In section 2.3, HI_MPI_SYS_SetProfile is added.

In section 2.4.1, PROFILE_TYPE_E is added.

Chapter 3 VI

In section 3.4, HI_MPI_VI_SetExtChnCrop and HI_MPI_VI_GetExtChnCrop are added. The description in the **Note** field of HI_MPI_VI_SetCSCAttr is modified.

Chapter 4 VO

In section 4.4, the description in the **Note** field of VO_VIDEO_LAYER_ATTR_S is modified.

Chapter 5 VPSS



In section 5.3, HI_MPI_VPSS_SetExtChnCrop and HI_MPI_VPSS_GetExtChnCrop are added.

Chapter 6 VENC

In section 6.3, the description in the **Note** field of HI_MPI_VENC_GetStream is added.

In section 6.4, H264E_REF_TYPE_E is added.

Chapter 8 Region Management

In section 8.4, the descriptions of OVERLAY_ATTR_S and OVERLAY_CHN_ATTR_S are modified.

Chapter 9 Audio

The description in 9.2.1.4 is modified. Table 9-5 is updated.

In section 9.3.1, the description in the **Note** field of HI_MPI_AI_SetVqeAttr is modified.

In section 9.4.1, AUDIO_HPF_FREQ_E to AI_RNR_CONFIG_S are added. The descriptions in the **Syntax** and **Member** fields of AI_VQE_CONFIG_S are modified.

Chapter 11 Proc Debugging Information

Sections 11.11, 11.13, and 11.16 are updated.

Issue 01 (2014-12-20)

This issue is the first official release, which incorporates the following changes:

The contents related to the Hi3516D are added.

Chapter 3 VI

In section 3.4, HI_MPI_VI_SetChnMinorAttr, HI_MPI_VI_GetChnMinorAttr, HI_MPI_VI_ClearChnMinorAttr, HI_MPI_VI_EnableCascade, HI_MPI_VI_DisableCascade, HI_MPI_VI_EnableCascadeChn, and HI_MPI_VI_DisableCascadeChn are deleted.

In section 3.5, SUBCHN, VI_CAS_CHN_1, VI_CAS_CHN_2, VI_RAW_READ_MODE_E, and VI_RAW_FRAME_INFO_S are deleted. The description of VI_RAW_DATA_INFO_S is modified.

Chapter 4 VO

In section 4.3, HI_MPI_VO_BindVideoLayer, HI_MPI_VO_UnBindVideoLayer, HI_MPI_VO_SetChnDispPos, HI_MPI_VO_GetChnDispPos, HI_MPI_VO_BindGraphicLayer, and HI_MPI_VO_UnBindGraphicLayer are deleted.

Chapter 6 VENC

In section 6.3, the descriptions of HI_MPI_VENC_CreateChn, HI_MPI_VENC_SetChnAttr, VENC_PARAM_H264_VUI_S, and HI_MPI_VENC_SetH264Vui are modified. HI_MPI_VENC_SetMpeg4Param and HI_MPI_VENC_GetMpeg4Param are deleted.

In section 6.4, VENC_PARAM_VUI_ASPECT_RATIO_S, VENC_PARAM_VUI_H264_TIME_INFO_S, and VENC_PARAM_VUI_VIDEO_SIGNAL_S are added. The descriptions of VENC_PARAM_H264_VBR_S and VENC_PARAM_MPEG4_CBR_S are modified.

Chapter 9 Audio



In section 9.3, the descriptions in the **Note** fields of HI_MPI_AI_SaveFile and HI_MPI_AENC_SaveFile are modified.

In section 9.3.5, ACODEC_SET_INPUT_VOL, ACODEC_GET_INPUT_VOL, ACODEC_SET_OUTPUT_VOL, and ACODEC_GET_OUTPUT_VOL are added.

Chapter 11 Proc Debugging Information

Section 11.13 and section 11.16 are modified.

Section 11.22 is added.

Issue 00B05 (2014-11-09)

This issue is the fifth draft release, which incorporates the following changes:

Chapter 3 VI

In section 3.4, the description in the **Note** field of HI_MPI_VI_SetWDRAttr is modified.

Chapter 4 VO

In section 4.3, the descriptions of HI_MPI_VO_GetScreenFrame and HI_MPI_VO_GetChnFrame are updated.

Chapter 5 VPSS

In section 5.4, the **Note** field of VPSS_LDC_ATTR_S is added.

Chapter 6 VENC

In section 6.3, the **Note** field of HI_MPI_VENC_StartRecvPicEx is added. The descriptions of HI_MPI_VENC_SetRoiCfg, HI_MPI_VENC_GetRoiCfg, HI_MPI_VENC_SetH265eSliceSplit, and HI_MPI_VENC_SetRoiBgFrameRate are modified. Figure 6-6 is added.

In section 6.4, VENC_RC_PRIORITY_E and VENC_PACK_INFO_S are added. The descriptions of H265E_NALU_TYPE_E and VENC_ATTR_H264_CBR_S are modified.

Chapter 8 Region Management

Table 8-2 is updated.

Chapter 9 Audio

In section 9.3.5, the **Note** fields of ACODEC_SET_PD_DACL, ACODEC_SET_PD_DACR, ACODEC_SET_PD_ADCL, and ACODEC_SET_PD_ADCR are added.

Chapter 10 VGS

Table 10-1 is updated.

Chapter 11 Proc Debugging Information

Sections 11.3, 11.8, 11.9, and 11.14 are modified.

Issue 00B04 (2014-10-09)

This issue is the fourth draft release, which incorporates the following changes:

Chapter 1 Introduction to the HiMPP

Section 1.1 and section 1.3 are updated.



Chapter 2 System Control

Section 2.2.3 is modified.

In section 2.3, the descriptions of HI_MPI_SYS_InitPtsBase, HI_MPI_SYS_MmzFlushCache, and HI_MPI_VB_ExitModCommPool are modified.

Chapter 3 VI

In section 3.4, the description in the **Difference** field of HI_MPI_VI_SetChnAttr is modified. The descriptions in the **Note** fields of HI_MPI_VI_EnableChn and HI_MPI_VI_EnableBayerDump are modified.

In section 3.5, the description of VI_DCI_PARAM_S is modified.

Chapter 4 VO

Section 4.2 is modified.

Chapter 6 VENC

In section 6.3, the description in the **Note** field of HI_MPI_VENC_RequestIDR is modified.

Issue 00B03 (2014-09-25)

This issue is the third draft release, which incorporates the following changes:

Chapter 3 VI

In section 3.3, figure 3-2 is modified.

Section 3.4 is modified.

Chapter 4 VO

In section 4.3, contents related to the interface operation and cascade operation are deleted.

In section 4.4, some data structures are deleted.

Chapter 5 VPSS

In section 5.3, HI_MPI_VPSS_SetPreScale, HI_MPI_VPSS_GetPreScale, HI_MPI_VPSS_SetGrpField, HI_MPI_VPSS_GetGrpField, HI_MPI_VPSS_SetGrpSizer, and HI_MPI_VPSS_GetGrpSizer are deleted.

In section 5.4, VPSS_CAPSEL_E, VPSS_PRESCALE_INFO_S, and VPSS_SIZER_INFO_S are deleted.

Chapter 6 VENC

In section 6.2.3, contents related to frame skipping reference are deleted.

HI_MPI_VENC_SetRcPriority and HI_MPI_VENC_GetRcPriority are added.

Chapter 8 Region Management

The description in section 8.2.1 is modified.

In section 8.4, the VPSS overlay description is deleted.

Chapter 9 Audio

Section 9.2.1 is modified.



In section 9.4, AUDIO_RESAMPLE_TYPE_E and AUDIO_RESAMPLE_ATTR_S are deleted.

Chapter 10 VGS

Section 10.2.1 and section 10.2.2 are modified.

Chapter 11 Proc Debugging Information

In section 11.1, the description of H265E is added.

Sections 11.7 to 11.11, 11.14, and 11.21 are modified.

Issue 00B02 (2014-09-14)

This issue is the second draft release, which incorporates the following changes:

The contents related to the Hi3535 are deleted.

Chapter 2 System Control

In section 2.4, WDR_MODE_E is added.

Chapter 3 VI

In section 3.4, the description in the **Note** field of HI_MPI_VI_SetWDRAttr is modified.

Chapter 5 VPSS

In section 5.2.3, table 5-1 is modified.

In section 5.4, the description in the **Note** field of VPSS_CROP_INFO_S is modified.

Chapter 6 VENC

In section 6.3, HI_MPI_VENC_SetSuperFrameCfg and HI_MPI_VENC_GetSuperFrameCfg are added. The description in the **Note** field of HI_MPI_VENC_StartRecvPicEx is modified. HI_MPI_VENC_SetH264eRefMode and HI_MPI_VENC_GetH264eRefMode are deleted.

In section 6.4, VENC_H264_REF_MODE_E is deleted. VENC_PARAM_H265_VBR_S, VENC_PARAM_H265_CBR_S, and VENC_SUPERFRAME_CFG_S are added.

Chapter 8 Region Management

In section 8.4, OVERLAY_MAX_NUM_VO and COVER_MAX_NUM_VO are deleted.

Chapter 9 Audio

In section 9.3.1, HI_MPI_AI_ClrPubAttr and HI_MPI_AI_SaveFile are added.

In section 9.3.2, HI_MPI_AO_ClrPubAttr is added.

In section 9.3.3, HI_MPI_AENC_SaveFile is added.

In section 9.4.1, AUDIO_SAVE_FILE_INFO_S is added.

Chapter 10 VGS

In section 10.2.2, the function description is modified.

Chapter 11 Proc Debugging Information

Contents related to the VDEC and IVE are deleted.



Issue 00B01 (2014-08-12)

This issue is the first draft release.



Contents

1 Introduction to the HiMPP	1-1
1.1 Overview	1-1
1.2 System Architecture	1-1
1.3 Architecture of the HiMPP	1-2



Figures

Figure 1-1 System architecture of the HiMPP.....	1-2
Figure 1-2 Internal workflow of the HiMPP	1-3



1 Introduction to the HiMPP

1.1 Overview

HiSilicon provides a media processing platform (HiMPP for short) for rapid application development. For applications, the HiMPP shields the complex processing procedures at the bottom layer related to the chip, and provides HiMPP programming interfaces (MPIs) for implementing various functions. The functions include:

- Capturing of input videos
- H.265, H.264, MJPEG, JPEG, or MPEG4 encoding
- H264, MPEG4, or MPEG2 decoding
- Displaying of output videos
- Video and picture pre-processing, including noise reduction (NR), image enhancement (IE), sharpening (SP), and de-interlacing (DIE)
- On-screen display (OSD) overlaying for encoded streams
- Video detection and analysis
- Intelligent analysis
- Audio capturing and output
- Audio encoding and decoding

This document describes the architecture and modules of the HiMPP, and the MPIs of each module. It is intended for but not limited to application development engineers and technical support personnel.

1.2 System Architecture

As shown in [Figure 1-1](#), the HiMPP provides the following layers:

- Hardware layer
 - It is comprised of the Hi35xx and peripherals, including the flash memory, double-data rate (DDR), video sensor or video analog-to-digital converter (VADC), and audio analog-to-digital converter (AADC).
- Operating system (OS) layer
 - It is based on the Linux OS.
- MPP



Based on the OS layer, the MPP controls the media processing functions. That is, the MPP shields the details about hardware processing, and provides application programming interfaces (APIs) for the application layer.

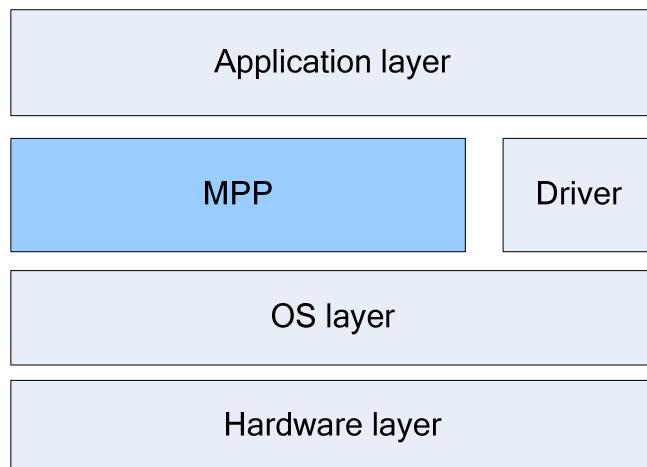
- Drivers

HiSilicon provides drivers for the hardware processing units of the Hi35xx. The drivers include the gigabit media access controller (GMAC) driver, secure digital input/output (SDIO) driver, inter-integrated circuit (I^2C) driver, Universal Serial Bus (USB) driver, and synchronous serial port (SSP) driver.

- Application layer

It is used for developing applications based on the MPP and drivers.

Figure 1-1 System architecture of the HiMPP



1.3 Architecture of the HiMPP

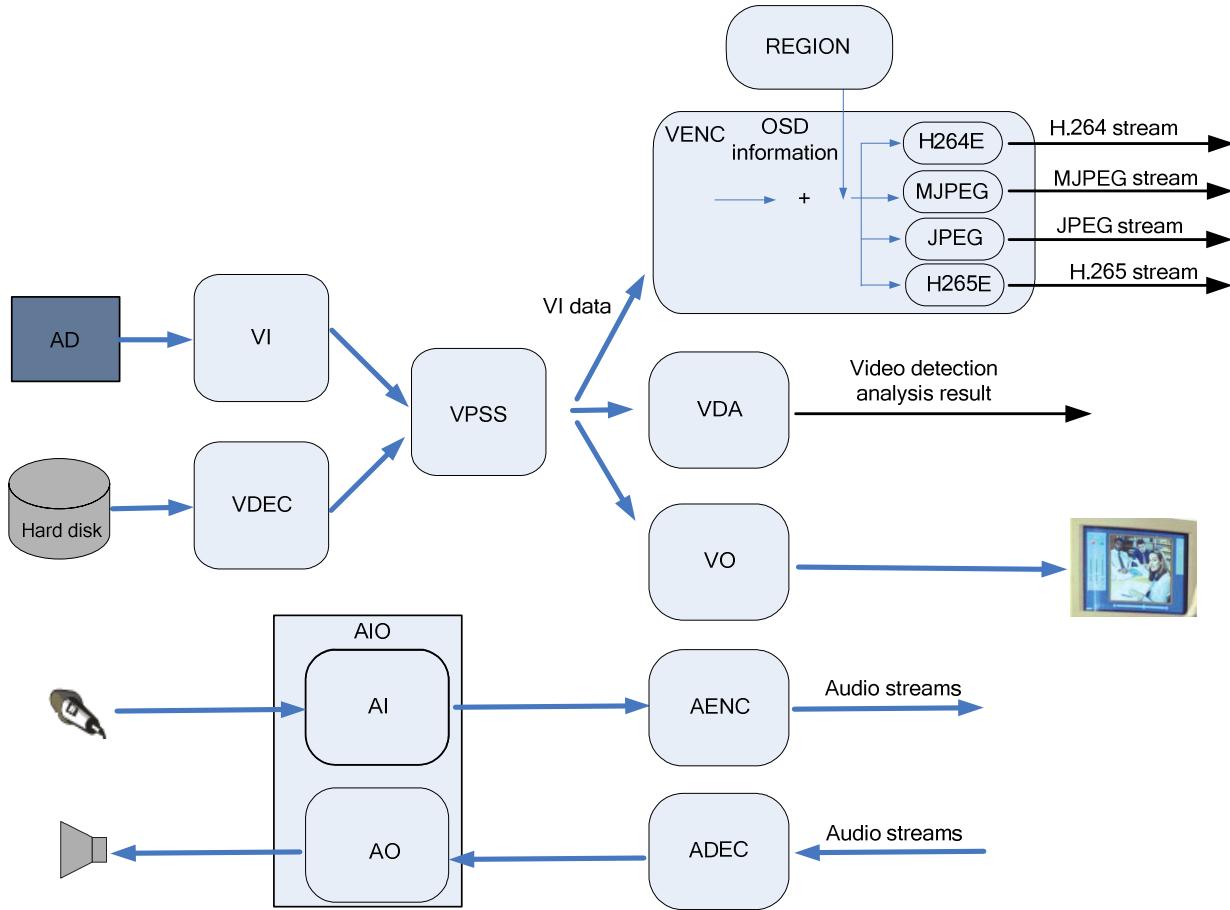
The HiMPP has the following modules:

- Video input unit (VIU)
- Video processing subsystem (VPSS)
- Video encoding (VENC) module
- Video decoding (VDEC) module
- Video output unit (VOU)
- Image signal processor (ISP)
- Video detection analysis (VDA) module
- Audio input unit (AIU)
- Audio output unit (AOU)
- Audio encoding (AENC) module
- Audio decoding (ADEC) module
- REGION module

[Figure 1-2](#) shows the internal workflow of the HiMPP.



Figure 1-2 Internal workflow of the HiMPP



The following describes the module functions:

- The VIU captures video pictures, crops and scales the pictures, and outputs pictures with different resolutions.
- The VDEC module decodes the encoded video streams and transfers the decoded streams to the VPSS or VOU. The VDEC module can decode the video streams in the format of H.264, MPEG4, or MPEG2.
- The VPSS receives the pictures from the VIU or VDEC module and performs denoising, image enhancement, and sharpening on the pictures. In addition, the VPSS can output the pictures from the same source as multi-channel pictures with different resolutions for encoding, previewing, or snapshot.
- The VENC module receives the pictures that are captured by the VIU and processed by the VPSS, and overlays the OSD pictures configured by using the REGION module, and encodes the pictures and output streams complying with different protocols.
- The VDA module receives pictures from the VIU, performs motion detection analysis and cover detection analysis, and outputs results.
- The VO module receives the pictures processed by the VPSS, performs play control, and outputs pictures to external video devices based on the configured output protocols.
- The AIU captures the audio data, and the AENC module encodes the captured audio data complying with multiple audio protocols and outputs encoded audio streams.



- You can transmit the audio streams from the Internet or external storage devices to the ADEC module. The ADEC module can decode the streams in different audio formats, and transmit the decoded data to the AOU for playing.



Contents

2 System Control	2-1
2.1 Overview	2-1
2.2 Functions	2-1
2.2.1 VB Pool.....	2-1
2.2.2 Binding the System	2-2
2.2.3 VI/VPSS Offline and Online Modes	2-3
2.3 API Reference	2-4
2.4 Data Structures	2-54
2.4.1 Basic Data Structures	2-54
2.4.2 System Control Data Structures	2-68
2.4.3 Public Video Data Structures	2-74
2.5 Error Codes	2-83
2.5.1 Error Codes of System Control MPIs	2-83
2.5.2 Error Codes of VB Pool MPIs	2-84



Figures

Figure 2-1 Typical data flows of public VB pools.....2-2



Tables

Table 2-1 Binding relationships supported by the HiMPP.....	2-2
Table 2-2 Calculation table for the VB size	2-69
Table 2-3 Error codes of the system control MPIs.....	2-84
Table 2-4 Error codes of VB pool MPIs	2-84



2 System Control

2.1 Overview

Based on the features of the Hi35xx, the system control module supports the following operations:

- Resets and initializes hardware components
- Initializes and deinitializes HiMPP modules
- Manages the mass physical memory and the status of HiMPP modules
- Provides version information about the current HiMPP system.

The HiMPP must be initialized before the HiMPP services are enabled by the applications. Similarly, the HiMPP must be deinitialized to release resources after the HiMPP services are disabled by applications.

2.2 Functions

2.2.1 VB Pool

The functions of the video buffer (VB) pool are as follows:

- Provides the mass physical memory for media services
- Allocates and releases the memory to make full use of the buffer pool
- Ensures that the physical memory resources are fully used by each media processing module.

A VB pool consists of the buffers with the same size and continuous physical addresses.

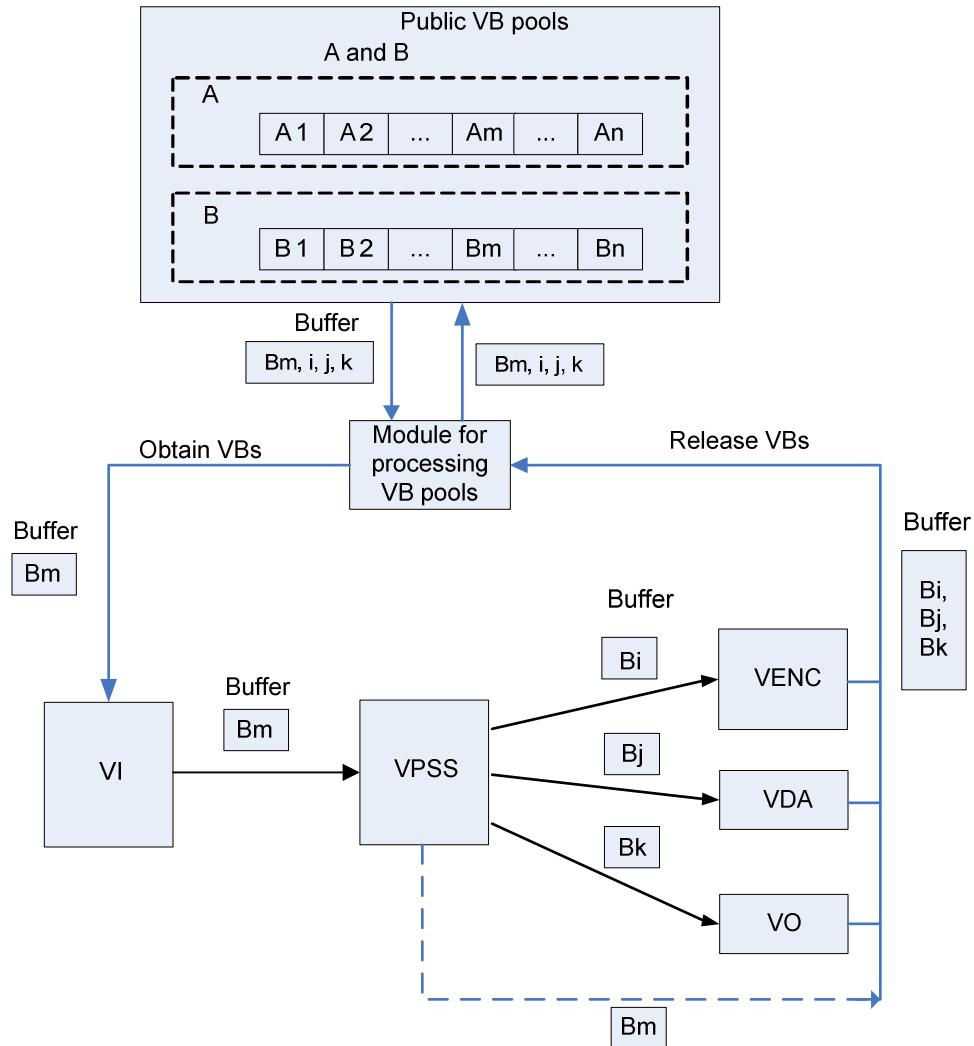
For the video input (VI) channel, public VB pools are required. All VI channels can obtain VB buffers from public VB pools for storing captured pictures. For example, the VB block Bm can be obtained from the public VB pool A. Public VB pools must be configured for VI channels before the system is initialized, because VI channels do not support the functions of creating and destroying public VB pools. The number of public VB pools and the size and number of buffers vary according to services.

The lifetime of a buffer is that the information of the buffer is transferred to other modules through a channel of the video processing subsystem (VPSS). See the solid line in [Figure 2-1](#). If the buffer information is not transparently transferred to other modules through the channel



of the VPSS, the buffer is returned to the public VB pool after the VPSS performs operations. See the dotted line in [Figure 2-1](#).

Figure 2-1 Typical data flows of public VB pools



2.2.2 Binding the System

The HiMPP provides a system binding MPI [HI_MPI_SYS_Bind](#). By calling this MPI, the data receiver binds a data source to establish an association. Note that only the data receiver can bind the data source. After binding, the data generated by the data source is automatically transmitted to the data receiver. [Table 2-1](#) shows the binding relationships supported by the HiMPP.

Table 2-1 Binding relationships supported by the HiMPP

Data Source	Data Receiver
VI	VO
	VENC



Data Source	Data Receiver
	VDA
	VPSS
	PCIV
VPSS	VO
	VENC
	VDA
	PCIV
	VPSS
VDEC	VPSS
	VO (SD device only)
	VENC
	VDA
	PCIV
VO (WBC)	VO
	VENC
	VPSS
	PCIV
AI	AENC
	AO
ADEC	AO

2.2.3 VI/VPSS Offline and Online Modes

The two collaboration modes of the VI and VPSS modules are as follows: (the mode switching is controlled by the system module parameter **vi_vpss_online** in the **load** script):

- Offline mode: The VI writes image data to the double data-rate (DDR) after parsing timings. Then the VPSS loads the data captured by the VI for image processing. This mode is the typical mode for the Hi3518 and Hi3520D.
- Online mode: The VI transfers data to the VPSS after parsing timings without writing data to the DDR. In this mode, the required bandwidth, memory, and end-to-end delay are reduced. Note that the operations such as CoverEx, OverlayEx, rotation, and LDC can be performed in online mode only after the VI writes data to the DDR by using VPSS channels. Some functions such as digital image stabilization (DIS) are supported only in offline mode.



2.3 API Reference

The system control module provides the functions of initializing the HiMPP system, binding and unbinding modules in the HiMPP system, obtaining the version number of the HiMPP, initializing a VB pool, and creating a VB pool.

The module provides the following MPIs:

- [`HI_MPI_SYS_SetConf`](#): Sets system control parameters.
- [`HI_MPI_SYS_GetConf`](#): Obtains system control parameters.
- [`HI_MPI_SYS_Init`](#): Initializes the HiMPP system.
- [`HI_MPI_SYS_Exit`](#): Deinitializes the HiMPP system.
- [`HI_MPI_SYS_Bind`](#): Binds a data source to a data receiver.
- [`HI_MPI_SYS_UnBind`](#): Unbinds a data source from a data receiver.
- [`HI_MPI_SYS_GetBindbyDest`](#): Obtains the information about a bound data source.
- [`HI_MPI_SYS_GetVersion`](#): Obtains the version number of the HiMPP.
- [`HI_MPI_SYS_GetCurPts`](#): Obtains the current presentation timestamp (PTS) of the HiMPP.
- [`HI_MPI_SYS_InitPtsBase`](#): initializes the HiMPP PTS.
- [`HI_MPI_SYS_SyncPts`](#): Synchronizes the HiMPP PTS.
- [`HI_MPI_SYS_Mmap`](#): Maps the storage address.
- [`HI_MPI_SYS_MmapCache`](#): Maps the storage address to the cache attribute.
- [`HI_MPI_SYS_Munmap`](#): Unmaps the storage address.
- [`HI_MPI_SYS_MflushCache`](#): Copies data from the cache to the media memory zone (MMZ) and invalidates data in the cache.
- [`HI_MPI_SYS_SetReg`](#): Sets the register values.
- [`HI_MPI_SYS_GetReg`](#): Obtains the register values.
- [`HI_MPI_SYS_MmzAlloc`](#): Allocates the MMZ in the user state.
- [`HI_MPI_SYS_MmzAlloc_Cached`](#): Allocates the MMZ supporting cache in the user state.
- [`HI_MPI_SYS_MmzFlushCache`](#): Copies data from the cache to the MMZ and invalidates data in the cache.
- [`HI_MPI_SYS_MmzFree`](#): Releases the MMZ in the user state.
- [`HI_MPI_SYS_SetMemConf`](#): Sets the DDR whose memory will be used.
- [`HI_MPI_SYS_GetMemConf`](#): Obtains the name of the DDR whose memory is used.
- [`HI_MPI_SYS_CloseFd`](#): Closes the file descriptor (FD) opened by the SYS module.
- [`HI_MPI_VB_SetConf`](#): Sets the attributes of a HiMPP VB pool.
- [`HI_MPI_VB_GetConf`](#): Obtains the attributes of a HiMPP VB pool.
- [`HI_MPI_VB_Init`](#): Initializes a HiMPP VB pool.
- [`HI_MPI_VB_Exit`](#): Deinitializes a HiMPP VB pool.
- [`HI_MPI_VB_CreatePool`](#): Creates a VB pool.
- [`HI_MPI_VB_DestroyPool`](#): Destroys a VB pool.
- [`HI_MPI_VB_GetBlock`](#): Obtains a buffer.
- [`HI_MPI_VB_ReleaseBlock`](#): Releases an obtained buffer.
- [`HI_MPI_VB_Handle2PhysAddr`](#): Obtains the physical address of a buffer.



- [HI_MPI_VB_Handle2PoolId](#): Obtains the ID of the VB pool where a buffer is located.
- [HI_MPI_VB_MmapPool](#): Maps the user-state virtual address to a VB pool.
- [HI_MPI_VB_MunmapPool](#): Unmaps the user-state address from a VB pool
- [HI_MPI_VB_GetBlkVirAddr](#): Obtains the user-state virtual address of a buffer in a VB pool.
- [HI_MPI_VB_InitModCommPool](#): Initializes a module public VB pool.
- [HI_MPI_VB_ExitModCommPool](#): Exits a module public VB pool.
- [HI_MPI_VB_SetModPoolConf](#): Sets the attributes of a module public VB pool.
- [HI_MPI_VB_GetModPoolConf](#): Obtains the attributes of a module public VB pool.
- [HI_MPI_SYS_SetProfile](#): Sets the power consumption scenario.
- [HI_MPI_SYS_GetViVpssMode](#): Obtains the mode (online/offline) information about the VIU/VPSS.
- [HI_MPI_SYS_GetVirMemInfo](#): Obtains information about the corresponding memory (including the physical address and cached attributes) based on the virtual address.
- [HI_MPI_VB_GetSupplementAddr](#): Obtains the auxiliary information of the VB block memory.
- [HI_MPI_VB_SetSupplementConf](#): Sets the auxiliary control information of the VB memory.
- [HI_MPI_VB_GetSupplementConf](#): Obtains the auxiliary control information of the VB memory.
- [HI_MPI_SYS_SetScaleCoefLevel](#): Sets the scaling coefficient level of the VPSS and video graphics subsystem (VGS).
- [HI_MPI_SYS_GetScaleCoefLevel](#): Obtains the scaling coefficient level of the VPSS and VGS.
- [HI_MPI_SYS_SetTimeZone](#): Sets the time zone information.
- [HI_MPI_SYS_GetTimeZone](#): Obtains the time zone information.

HI_MPI_SYS_SetConf

[Description]

Sets system control parameters.

[Syntax]

```
HI_S32 HI_MPI_SYS_SetConf(const MPP_SYS_CONF_S *pstSysConf);
```

[Parameter]

Parameter	Description	Input/Output
pstSysConf	Pointer to system control parameters Static attribute A static attribute can be set only before the system is initialized, the device is started, or the channel is enabled.	Input

[Return Value]



Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]

- Header files: hi_comm_sys.h, mpi_sys.h
- Library file: libmpi.a

[Note]

This MPI can be called to configure the HiMPP system only before the entire HiMPP system is initialized. Otherwise, the HiMPP system fails to be configured.

[Example]

```
HI_S32 s32ret;
VB_CONF_S struVbConf;
MPP_SYS_CONF_S struSysConf;

memset(&struVbConf, 0, sizeof(VB_CONF_S));

struVbConf.u32MaxPoolCnt          = 64;
struVbConf.astCommPool[0].u32BlkSize = 1920*1088*2;
struVbConf.astCommPool[0].u32BlkCnt  = 15;
memset(struVbConf.astCommPool[0].acMmzName, 0, sizeof(struVbConf.astCommPool[0].acMmzName));
s32ret = HI_MPI_VB_SetConf(&struVbConf);
if (HI_SUCCESS != s32ret)
{
    return s32ret;
}
s32ret = HI_MPI_VB_Init();
if (HI_SUCCESS != s32ret)
{
    return s32ret;
}
struSysConf.u32AlignWidth = 16;
/* set config of mpp system*/
s32ret = HI_MPI_SYS_SetConf (&struSysConf);
if (HI_SUCCESS != s32ret)
{
    printf("Set mpp sys config failed!\n");
    return s32ret;
}
```



```
/* init system*/
s32ret = HI_MPI_SYS_Init();
if (HI_SUCCESS != s32ret)
{
printf("Mpi init failed!\n");
return s32ret;
}
/* .... */

/* exit system*/
s32ret = HI_MPI_SYS_Exit();
if (HI_SUCCESS != s32ret)
{
printf("Mpi exit failed!\n");
return s32ret;
}

s32ret = HI_MPI_VB_Exit();
if (HI_SUCCESS != s32ret)
{
return s32ret;
}
```

[See Also]

[HI_MPI_VB_GetConf](#)

HI_MPI_SYS_GetConf

[Description]

Obtains system control parameters.

[Syntax]

```
HI_S32 HI_MPI_SYS_GetConf(MPP_SYS_CONF_S *pstSysConf);
```

[Parameter]

Parameter	Description	Input/Output
pstSysConf	Pointer to system control parameters Static attribute	Output

[Return Value]



Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]

- Header files: hi_comm_sys.h, mpi_sys.h
- Library file: libmpi.a

[Note]

You can obtain system control parameters only after calling [HI_MPI_SYS_SetConf](#) successfully.

[Example]

None

[See Also]

[HI_MPI_SYS_SetConf](#)

HI_MPI_SYS_Init

[Description]

Initializes the HiMPP system. The module including the AIU, AOU, VIU, VOU, VENC, VDEC, region management module, VDA module are initialized.

[Syntax]

```
HI_S32 HI_MPI_SYS_Init(HI_VOID);
```

[Parameter]

None

[Return Value]

Parameter	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]

- Header files: hi_comm_sys.h, mpi_sys.h
- Library file: libmpi.a

[Note]



- You must call [HI_MPI_SYS_SetConf](#) before initializing the HiMPP system. Otherwise, the HiMPP system fails to be initialized.
- The running of the HiMPP system depends on the buffer pool. Therefore, you must call [HI_MPI_VB_Init](#) to initialize the buffer pool before initializing the HiMPP system.
- If the system is initialized for several times, a code indicating success is returned each time. This has no effect on the running status of the HiMPP.
- You only need to use one process to initialize the system.

[Example]

See the example of [HI_MPI_SYS_SetConf](#).

[See Also]

[HI_MPI_SYS_Exit](#)

HI_MPI_SYS_Exit

[Description]

Deinitializes the HiMPP system. The modules including the AIU, AOU, VIU, VOU, VENC, VDEC, region management module, VDA module are destroyed or disabled.

[Syntax]

```
HI_S32 HI_MPI_SYS_Exit(HI_VOID);
```

[Parameter]

None

[Return Value]

Parameter	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]

- Header files: hi_comm_sys.h, mpi_sys.h
- Library file: libmpi.a

[Note]

- If a process is blocked in the MPI, the HiMPP system fails to be deinitialized. The HiMPP system can be deinitialized only when the MPI is not blocked.
- This MPI can be called repeatedly, and a code indicating success is returned.
- The AENC channel and ADEC channel are not destroyed after system deinitialization. Therefore, you need to destroy these channels. If the processes of creating these channels end, these channels are destroyed automatically.

[Example]



See the example of [HI_MPI_SYS_SetConf](#).

[See Also]

[HI_MPI_SYS_Init](#)

HI_MPI_SYS_Bind

[Description]

Binds a data source and a data receiver.

[Syntax]

```
HI_S32 HI_MPI_SYS_Bind(MPP_CHN_S *pstSrcChn, MPP_CHN_S *pstDestChn);
```

[Parameter]

Parameter	Description	Input/Output
pstSrcChn	Pointer to the source channel	Input
pstDestChn	Pointer to the destination channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]

- Header files: hi_comm_sys.h, mpi_sys.h
- Library file: libmpi.a

[Note]

- For details about the supported binding relationships, see [Table 2-1](#).
- A data receiver can be bound only to one data source.
- The binding establishes an association between a data source and a data receiver. After binding, the data generated by the data source is automatically transmitted to the data receiver.
- When the VIU and VDEC are data sources, related channels act as transmitters to transmit data to other modules. You can set the device ID to 0. The SDK does not check the entered device ID.
- When the VOI acts as the data source for transmitting write back (WBC) data, the device transmits data to other modules. You can set the channel ID to 0. The SDK does not check the entered channel ID.



- When the VPSS acts as the data receiver, it receives data from other modules by using the device (GROUP). You can set the channel ID to **0**. The SDK does not check the entered channel ID.
- When the VENC acts as the data receiver, it receives data from other modules by channel ID. You can set the device ID to 0. The SDK does not check the entered device ID.
- The device ID and channel ID do not need to be specified in other cases.

[Example]

```
HI_S32 s32Ret;  
MPP_CHN_S stBindSrc;  
MPP_CHN_S stBindDest;  
  
stBindDest.enModId = HI_ID_VPSS;  
stBindDest.s32ChnId = 0;  
stBindDest.s32DevId = 0;  
  
stBindSrc.enModId = HI_ID_VIU;  
stBindSrc.s32ChnId = 0;  
stBindSrc.s32DevId = 0;  
  
s32Ret = HI_MPI_SYS_Bind(&stBindSrc, &stBindDest);  
if (s32Ret)  
{  
    return HI_FAILURE;  
}
```

[See Also]

[HI_MPI_SYS_UnBind](#)

HI_MPI_SYS_UnBind

[Description]

Unbind a data source from a data receiver.

[Syntax]

```
HI_S32 HI_MPI_SYS_UnBind(MPP_CHN_S *pstSrcChn, MPP_CHN_S *pstDestChn);
```

[Parameter]

Parameter	Description	Input/Output
pstSrcChn	Pointer to the source channel	Input
pstDestChn	Pointer to the destination channel	Input

[Return Value]



Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]

- Header files: hi_comm_sys.h, mpi_sys.h
- Library file: libmpi.a

[Note]

None

[Example]

See the related sample.

[See Also]

[HI_MPI_SYS_Bind](#)

HI_MPI_SYS_GetBindbyDest

[Description]

Obtains the information about a bound data source.

[Syntax]

```
HI_S32 HI_MPI_SYS_GetBindbyDest(MPP_CHN_S *pstDestChn, MPP_CHN_S  
*pstSrcChn);
```

[Parameter]

Parameter	Description	Input/Output
pstSrcChn	Pointer to the source channel	Output
pstDestChn	Pointer to the destination channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]



- Header files: hi_comm_sys.h, mpi_sys.h
- Library file: libmpi.a

[Note]

None

[Example]

None.

[See Also]

- [HI_MPI_SYS_Bind](#)
- [HI_MPI_SYS_UnBind](#)

HI_MPI_SYS_GetVersion

[Description]

Obtains the version number of the HiMPP.

[Syntax]

```
HI_S32 HI_MPI_SYS_GetVersion(MPP_VERSION_S *pstVersion);
```

[Parameter]

Parameter	Description	Input/Output
pstVersion	Pointer to the version number Dynamic attribute. A dynamic attribute can be set at any time.	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]

- Header files: hi_comm_sys.h, mpi_sys.h
- Library file: libmpi.a

[Note]

None

[Example]

```
HI_S32 s32ret;
```



```
MPP_VERSION_S stVersion;  
  
s32ret = HI_MPI_SYS_GetVersion(&stVersion);  
if (HI_SUCCESS != s32ret)  
{  
    return s32ret;  
}  
printf("mpi version is %s\n", stVersion.aVersion);
```

[See Also]

None

HI_MPI_SYS_GetCurPts

[Description]

Obtains the current PTS of the HiMPP.

[Syntax]

```
HI_S32 HI_MPI_SYS_GetCurPts(HI_U64 *pu64CurPts);
```

[Parameter]

Parameter	Description	Input/Output
pu64CurPts	Pointer to the current PTS	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]

- Header files: hi_comm_sys.h, mpi_sys.h
- Library file: libmpi.a

[Note]

None

[Example]

None

[See Also]

None



HI_MPI_SYS_InitPtsBase

[Description]

Initializes the HiMPP PTS.

[Syntax]

```
HI_S32 HI_MPI_SYS_InitPtsBase(HI_U64 u64PtsBase);
```

[Parameter]

Parameter	Description	Input/Output
u64PtsBase	PTS base (in μ s)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]

- Header files: hi_comm_sys.h, mpi_sys.h
- Library file: libmpi.a

[Note]

Regardless of the original system PTS, initializing the PTS base forces the current system PTS to u64PtsBase. Therefore, you are recommended to call this MPI before a media service is enabled. For example, you can call this MPI immediately when the OS starts. If a media service is enabled, you can call [HI_MPI_SYS_SyncPts](#) to tune the PTS.

[Example]

None

[See Also]

None

HI_MPI_SYS_SyncPts

[Description]

Synchronizes the HiMPP PTS.

[Syntax]

```
HI_S32 HI_MPI_SYS_SyncPts(HI_U64 u64PtsBase);
```

[Parameter]



Parameter	Description	Input/Output
u64PtsBase	PTS base	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]

- Header files: hi_comm_sys.h, mpi_sys.h
- Library file: libmpi.a

[Note]

After the current system PTS (in μ s) is fine-tuned, the PTS does not roll back. When multiple chips are synchronized, the difference between the clock sources of the boards may be significant. Therefore, you are recommended to tune the PTS once a second.

[Example]

None

[See Also]

None

HI_MPI_SYS_Mmap

[Description]

Maps the storage address.

[Syntax]

```
HI_VOID * HI_MPI_SYS_Mmap(HI_U32 u32PhyAddr, HI_U32 u32Size);
```

[Parameter]

Parameter	Description	Input/Output
u32PhyAddr	Start address of the memory to be mapped	Input
u32Size	Number of mapped bytes	Input

[Return Value]



Return Value	Description
0	Invalid address
Other values	Valid address

[Requirement]

- Header files: hi_comm_sys.h, mpi_sys.h
- Library file: libmpi.a

[Note]

- This MPI is equivalent to the mmap function in Linux.
- The address entered must be a valid physical address.

[Example]

None

[See Also]

[HI_MPI_SYS_Munmap](#)

HI_MPI_SYS_MmapCache

[Description]

Maps the storage address to the cache attribute.

[Syntax]

```
HI_VOID * HI_MPI_SYS_MmapCache (HI_U32 u32PhyAddr, HI_U32 u32Size);
```

[Parameter]

Parameter	Description	Input/Output
u32PhyAddr	Start address of the memory to be mapped	Input
u32Size	Number of mapped bytes	Input

[Return Value]

Return Value	Description
0	Invalid address
Other values	Valid address

[Requirement]

- Header files: hi_comm_sys.h, mpi_sys.h
- Library file: libmpi.a



[Note]

- This MPI is equivalent to the mmap function in Linux.
- The address entered must be a valid physical address.

[Example]

None

[See Also]

[HI_MPI_SYS_Munmap](#)

HI_MPI_SYS_Munmap

[Description]

Unmaps the storage address.

[Syntax]

```
HI_S32 HI_MPI_SYS_Munmap(HI_VOID* pVirAddr, HI_U32 u32Size);
```

[Parameter]

Parameter	Description	Input/Output
pVirAddr	Address returned after mmap is called	Input
u32Size	Length of mapped area, in byte	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]

- Header files: hi_comm_sys.h, mpi_sys.h
- Library file: libmpi.a

[Note]

This MPI is equivalent to the munmap function in Linux.

[Example]

None

[See Also]

[HI_MPI_SYS_Mmap](#)



HI_MPI_SYS_MflushCache

[Description]

Copies data from the cache to the MMZ and invalidates data in the cache.

[Syntax]

```
HI_S32 HI_MPI_SYS_MflushCache(HI_U32 u32PhyAddr, HI_VOID *pVirAddr,  
HI_U32 u32Size);
```

[Parameter]

Parameter	Description	Input/Output
u32PhyAddr	Start physical address for storing the data to be used The address is 4-byte-aligned.	Input
pVirtAddr	Pointer to the start virtual address for storing the data to be used The address must be the user-mode address and cannot be empty. The virtual address must be the mapping address for the physical address.	Input
u32Size	Amount of the data to be used This parameter cannot be 0.	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]

- Header files: hi_comm_sys.h, mpi_sys.h
- Library file: libmpi.a

[Note]

- If the data in the cache is the latest data, you need to call this MPI to synchronize the data to the memory. This ensures that the hardware such as the IVE that cannot directly access the cache can obtain correct data.
- This MPI must be called if [HI_MPI_SYS_MmapCache](#) is called.
- You need to ensure that the transferred parameters are valid.

[See Also]

[HI_MPI_SYS_MmapCache](#)



HI_MPI_SYS_SetReg

[Description]

Sets register values.

[Syntax]

```
HI_S32 HI_MPI_SYS_SetReg(HI_U32 u32Addr, HI_U32 u32Value)
```

[Parameter]

Parameter	Description	Input/Output
u32Addr	Written address	Input
u32Value	Written value	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]

- Header files: hi_comm_sys.h, mpi_sys.h
- Library file: libmpi.a

[Note]

The address entered must be a valid physical address.

[Example]

None

[See Also]

[HI_MPI_SYS_GetReg](#)

HI_MPI_SYS_GetReg

[Description]

Obtains register values.

[Syntax]

```
HI_S32 HI_MPI_SYS_GetReg(HI_U32 u32Addr, HI_U32 *pu32Value)
```

[Parameter]



Parameter	Description	Input/Output
u32Addr	Physical address	Input
pu32Value	Value of a memory address	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]

- Header files: hi_comm_sys.h, mpi_sys.h
- Library file: libmpi.a

[Note]

The address entered must be a valid physical address.

[Example]

None

[See Also]

[HI_MPI_SYS_SetReg](#)

HI_MPI_SYS_MmzAlloc

[Description]

Allocates the MMZ in the user state.

[Syntax]

```
HI_S32 HI_MPI_SYS_MmzAlloc(HI_U32 *pu32PhyAddr, void **ppVitAddr, const
HI_CHAR *pstrMmb, const HI_CHAR *pstrZone, HI_U32 u32Len);
```

[Parameter]

Parameter	Description	Input/Output
pu32PhyAddr	Pointer to the physical address	Output
ppVirtAddr	Pointer to the virtual address	Output
pstrMmb	String pointer to the media memory block (MMB) name	Input
pstrZone	String pointer to the MMZ zone name	Input



Parameter	Description	Input/Output
u32Len	Buffer size	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]

- Header files: hi_comm_sys.h, mpi_sys.h
- Library file: libmpi.a

[Note]

The MMZ consists of multiple zones and each zone has multiple MMBs. You can call this MPI to allocate a memory block ***pstrMmb** with the size of **u32Len** in the ***pstrZone** zone of the MMZ. In this case, the pointers that point to the physical address and user-state virtual address are returned. If there is the **anonymous** zone in the MMZ, set ***pstrZone** to **null**. If ***pstrMmb** is set to **null**, the created MMB is named **<null>**.

[Example]

None

[See Also]

None

HI_MPI_SYS_MmzAlloc_Cached

[Description]

Allocates the MMZ supporting cache in the user state.

[Syntax]

```
HI_S32 HI_MPI_SYS_MmzAlloc_Cached(HI_U32 *pu32PhyAddr, HI_VOID
**ppVitAddr, const HI_CHAR *pstrMmb, const HI_CHAR *pstrZone, HI_U32
u32Len);
```

[Parameter]

Parameter	Description	Input/Output
pu32PhyAddr	Pointer to the physical address	Output
ppVirtAddr	Pointer to the virtual address	Output
pstrMmb	String pointer to the MMB name	Input



Parameter	Description	Input/Output
pstrZone	String pointer to the MMZ name	Input
u32Len	Buffer size	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]

- Header files: hi_comm_sys.h, mpi_sys.h
- Library file: libmpi.a

[Note]

- The differences between [HI_MPI_SYS_MmzAlloc_Cached](#) and [HI_MPI_SYS_MmzAlloc](#) are as follows: The memory that is allocated by calling [HI_MPI_SYS_MmzAlloc_Cached](#) supports cache. [HI_MPI_SYS_MmzAlloc_Cached](#) is recommended if the memory to be allocated will be frequently used. This improves the CPU read/write efficiency and system performance. For example, when intelligent video engine (IVE) operators are being used, a large amount of data is frequently read and written. If you allocate the memory by calling [HI_MPI_SYS_MmzAlloc_Cached](#), the working efficiency of the CPU is improved.
- When the CPU accesses the memory that is allocated by calling [HI_MPI_SYS_MmzAlloc_Cached](#), the data in the memory will be stored in the cache. The hardware devices such as the IVE can access only the physical memory rather than the cache. In this case, [HI_MPI_SYS_MmzFlushCache](#) needs to be called to synchronize data.

[Note]

Note

[See Also]

- [HI_MPI_SYS_MmzAlloc](#)
- [HI_MPI_SYS_MmzFlushCache](#)

HI_MPI_SYS_MmzFlushCache

[Description]

Copies data from the cache to the MMZ and invalidates data in the cache.

[Syntax]

```
HI_S32 HI_MPI_SYS_MmzFlushCache(HI_U32 u32PhyAddr, HI_VOID *pVitAddr,
```



```
HI_U32 u32Size);
```

[Parameter]

Parameter	Description	Input/Output
u32PhyAddr	Start physical address for storing the data to be used	Input
pVirtAddr	Pointer to the start virtual address for storing the data to be used	Input
u32Size	Amount of the data to be used	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]

- Header files: hi_comm_sys.h, mpi_sys.h
- Library file: libmpi.a

[Note]

- If the data in the cache is the latest data, you need to call this MPI to synchronize the data to the memory. This ensures that the hardware such as the IVE that cannot directly access the cache can obtain correct data.
- This MPI must be called if [HI_MPI_SYS_MmzAlloc_Cached](#) is called.
- If **u32PhyAddr** is set to **0**, the entire cache is operated.
- You need to ensure that the transferred parameters are valid.

[Example]

Note

[See Also]

[HI_MPI_SYS_MmzAlloc_Cached](#)

HI_MPI_SYS_MmzFree

[Description]

Releases the MMZ in the user state.

[Syntax]

```
HI_S32 HI_MPI_SYS_MmzFree(HI_U32 u32PhyAddr, HI_VOID *pVirtAddr);
```

[Parameter]



Parameter	Description	Input/Output
u32PhyAddr	Physical address	Input
pVirtAddr	Pointer to the virtual address	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]

- Header files: hi_comm_sys.h, mpi_sys.h
- Library file: libmpi.a

[Note]

The entered address must be a valid physical address. The pointer that points to the virtual address can be set to null.

[Example]

None

[See Also]

[HI_MPI_SYS_MmzAlloc](#)

HI_MPI_SYS_SetMemConf

[Description]

Sets the DDR whose memory is used by the device channel of a module.

[Syntax]

```
HI_S32 HI_MPI_SYS_SetMemConf (MPP_CHN_S *pstMppChn, const HI_CHAR  
*pcMmzName);
```

[Parameter]

Parameter	Description	Input/Output
pstMppChn	Information about a device channel of a module	Input
pcMmzName	Name of the DDR where a memory is located	Input

[Return Value]



Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]

- Header files: hi_comm_sys.h, mpi_sys.h
- Library file: libmpi.a

[Note]

- The DDR name must be the name of an existing DDR. Assume that there are two DDRs. One is unnamed and the other is ddr1. In this case, **pcMmzName** must be set to **null** or **ddr1**. This MPI is used to evenly allocate the memory.
- When the memory is evenly allocated for the VOU, ensure that the memories of the channels of the same device are from the same DDR. Otherwise, the SDK allocates memory from the DDR configured for channel 0 of this device by default.

[Example]

```
HI_S32 i, s32Ret;
HI_CHAR *pcMmzName;
MPP_CHN_S stMppChnVI;

/*Evenly allocate memory for 16D1 channels*/
for(i=0;i<16;i++)
{
    stMppChnVI.enModId = HI_ID_VIU;
    stMppChnVI.s32DevId = 0;
    stMppChnVI.s32ChnId = i;
    if(0 == (i%2))
    {
        pcMmzName = NULL;
    }
    else
    {
        pcMmzName = "ddr1";
    }
    s32Ret = HI_MPI_SYS_SetMemConf(&stMppChnVI,pcMmzName);
    if (HI_SUCCESS != s32ret)
    {
        printf("SetMemConf ERR !\n");
        return s32ret;
    }
}
```



[See Also]

[HI_MPI_SYS_SetMemConf](#)

HI_MPI_SYS_SetMemConf

[Description]

Obtains the name of the DDR whose memory is used by the device channel.

[Syntax]

```
HI_S32 HI_MPI_SYS_SetMemConf (MPP_CHN_S *pstMppChn, HI_CHAR *pcMmzName);
```

[Parameter]

Parameter	Description	Input/Output
pstMppChn	Information about a device channel of a module	Input
pcMmzName	Name of the DDR where a memory is located	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]

- Header files: hi_comm_sys.h, mpi_sys.h
- Library file: libmpi.a

[Note]

None

[Example]

None

[See Also]

[HI_MPI_SYS_SetMemConf](#)

HI_MPI_SYS_CloseFd

[Description]

Closes all the logs, system, and memory FDs that are opened by the SYS module.

[Syntax]

```
HI_S32 HI_MPI_SYS_CloseFd (HI_VOID);
```



[Parameter]

None

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]

- Header files: hi_comm_sys.h, mpi_sys.h
- Library file: libmpi.a

[Note]

Before calling this MPI, ensure that all modules are disabled.

[Example]

None

[See Also]

None

HI_MPI_VB_SetConf

[Description]

Sets the attributes of a HiMPP VB pool.

[Syntax]

```
HI_S32 HI_MPI_VB_SetConf (const VB_CONF_S *pstVbConf);
```

[Parameter]

Parameter	Description	Input/Output
pstVbConf	Pointer to VB pool attributes Static attribute	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."



[Requirement]

- Header files: hi_comm_vb.h, mpi_vb.h
- Library file: libmpi.a

[Note]

- The attributes of a VB pool can be set only before the system is initialized. Otherwise, an error code indicating failure is returned.
- The video buffer must be configured based on the application scenario. For details about configuration rules, see section [2.2.1 "VB Pool."](#)
- The size of each buffer block in the public VB pool varies according to the current picture pixel format and the picture compression mode (compressed or uncompressed). For details about the allocated size, see the description in [VB_CONF_S](#).

[Example]

```
HI_S32 s32ret;
VB_CONF_S stVbConf;

memset(&stVbConf, 0, sizeof(VB_CONF_S));
stVbConf.u32MaxPoolCnt = 128;
stVbConf.astCommPool[0].u32BlkSize = 768*576*2;
stVbConf.astCommPool[0].u32BlkCnt = 20;
stVbConf.astCommPool[1].u32BlkSize = 384*288*2;
stVbConf.astCommPool[1].u32BlkCnt = 40;

s32ret = HI_MPI_VB_SetConf(&stVbConf);
if (HI_SUCCESS != s32ret)
{
    printf("set vb err:0x%x\n", s32ret);
    return s32ret;
}

s32ret = HI_MPI_VB_Init();
if (HI_SUCCESS != s32ret)
{
    printf("init vb err:0x%x\n", s32ret);
    return s32ret;
}
/* ... */
(void)HI_MPI_VB_Exit();
```

[See Also]

[HI_MPI_VB_GetConf](#)



HI_MPI_VB_GetConf

[Description]

Obtains the attributes of a HiMPP VB pool.

[Syntax]

```
HI_S32 HI_MPI_VB_GetConf (VB_CONF_S *pstVbConf);
```

[Parameter]

Parameter	Description	Input/Output
pstVbConf	Pointer to VB pool attributes Static attribute	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vb.h, mpi_vb.h
- Library file: libmpia.a

[Note]

Before obtaining the attributes of a HiMPP VB pool, you must set the attributes of the VB pool by calling call [HI_MPI_VB_SetConf](#).

[Example]

None

[See Also]

[HI_MPI_VB_SetConf](#)

HI_MPI_VB_Init

[Description]

Initializes a HiMPP VB pool.

[Syntax]

```
HI_S32 HI_MPI_VB_Init (HI_VOID);
```

[Parameter]



None

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vb.h, mpi_vb.h
- Library file: libmpi.a

[Note]

- Before initializing a VB pool, you must set the attributes of the VB pool by calling [HI_MPI_VB_SetConf](#). Otherwise, the VB pool fails to be initialized.
- This MPI can be called repeatedly, and a code indicating success is returned.

[Example]

See the example of [HI_MPI_VB_SetConf](#).

[See Also]

[HI_MPI_VB_Exit](#)

HI_MPI_VB_Exit

[Description]

Deinitializes a HiMPP VB pool.

[Syntax]

```
HI_S32 HI_MPI_VB_Exit (HI_VOID);
```

[Parameter]

None

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vb.h, mpi_vb.h



- Library file: libmpi.a

[Note]

- Before deinitializing a HiMPP VB pool, you must call [HI_MPI_SYS_Exit](#) to deinitialize the HiMPP system. Otherwise, the VB pool fails to be deinitialized.
- This MPI can be called repeatedly, and a code indicating success is returned.
- After the VB pool is deinitialized, the configurations of the VB pool are retained.

[Example]

See the example of [HI_MPI_VB_SetConf](#).

[See Also]

[HI_MPI_VB_Init](#)

HI_MPI_VB_CreatePool

[Description]

Creates a VB pool.

[Syntax]

```
VB_POOL HI_MPI_VB_CreatePool(HI_U32 u32BlkSize, HI_U32 u32BlkCnt, const  
HI_CHAR *pcMmzName);
```

[Parameter]

Parameter	Description	Input/Output
u32BlkSize	Size of each buffer in a VB pool Value range: (0, 2^{32}), in byte	Input
u32BlkCnt	Number of buffers in a VB pool Value range: (0, 2^{32})	Input
pcMmzName	Name of the DDR where a VB pool is located	Input

[Return Value]

Return Value	Description
Other values	The VB pool ID is valid.
VB_INVALID_POOLID	A VB pool fails to be created, because the parameter is invalid or the reserved memory is insufficient.

[Requirement]

- Header files: hi_comm_vb.h, mpi_vb.h
- Library file: libmpi.a

[Note]



- A VB pool is an area of the available memory. It consists of several buffers with the same size. If the size of a VB pool to be created is larger than the reserved memory, the VB pool fails to be created.
- Ensure that the name of an existing DDR is entered. If the DDR does not exist, no memory is allocated.

[Example]

```
VB_POOL VbPool;
VB_BLK VbBlk;
HI_U32 u32BlkSize = 768*576*2;
HI_U32 u32BlkCnt = 15;
HI_U32 u32Addr;

/* create a video buffer pool*/
VbPool = HI_MPI_VB_CreatePool(u32BlkSize,u32BlkCnt,"ddr1");
if ( VB_INVALID_POOLID == VbPool )
{
printf("create vb err\n");
return HI_FAILURE;
}

/* get a buffer from pool*/
VbBlk = HI_MPI_VB_GetBlock(VbPool, u32BlkSize,"ddr1");
if (VB_INVALID_HANDLE == VbBlk )
{
printf("get vb block err\n");
(void)HI_MPI_VB_DestroyPool(VbPool);
return HI_FAILURE;
}

/* get the physical address of buffer*/
u32Addr = HI_MPI_VB_Handle2PhysAddr(VbBlk);
if (HI_NULL_PTR == u32Addr)
{
printf("blk to physaddr err\n");
(void)HI_MPI_VB_ReleaseBlock(VbBlk);
(void)HI_MPI_VB_DestroyPool(VbPool);
return HI_FAILURE;
}

/* use this address do something*/
/* then release the buffer*/
(void)HI_MPI_VB_ReleaseBlock(VbBlk);
/* destroy video buffer pool */
(void)HI_MPI_VB_DestroyPool(VbPool);
```

[See Also]



[HI_MPI_VB_DestroyPool](#)

[Description]

Destroys a VB pool.

[Syntax]

```
HI_S32 HI_MPI_VB_DestroyPool(VB_POOL Pool);
```

[Parameter]

Parameter	Description	Input/Output
Pool	VB pool ID Value range: [0, VB_MAX_POOLS)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vb.h, mpi_vb.h
- Library file: libmpi.a

[Note]

- If the VB pool to be destroyed does not exist, [HI_ERR_VB_UNEXIST](#) is returned.
- When the HiMPP system is deinitialized, all buffer pools including user-state buffer pools are destroyed.

[Example]

See the example of [HI_MPI_VB_CreatePool](#).

[See Also]

[HI_MPI_VB_CreatePool](#)

[HI_MPI_VB_GetBlock](#)

[Description]

Obtains a buffer.

[Syntax]

```
VB_BLK HI_MPI_VB_GetBlock(VB_POOL Pool, HI_U32 u32BlkSize, const HI_CHAR
```



*pcMmzName) ;

[Parameter]

Parameter	Description	Input/Output
Pool	VB pool ID Value range: [0, VB_MAX_POOLS)	Input
u32BlkSize	Buffer size Value range: (0, 2^{32}), in byte	Input
pcMmzName	Name of the DDR where a VB pool is located	Input

[Return Value]

Return Value	Description
Other values	The buffer handle is valid.
VB_INVALID_HANDLE	A buffer fails to be obtained.

[Error Code]

None

[Requirement]

- Header files: hi_comm_vb.h, mpi_vb.h
- Library file: libmpi.a

[Note]

- After creating a VB pool, you can call this MPI to obtain a buffer from the VB pool. That is, you can set the first parameter Pool to the ID of the created VB pool, and left the second parameter **u32BlkSize** blank. In this way, the size of the obtained buffer is equal to the size of the buffer that is specified when you create the VB pool. The **pcMmzName** parameter is invalid when a buffer is obtained from a specified VB pool.
- If you want to obtain a buffer with a specified size from a public VB pool, you can set the first parameter Pool to an invalid ID (VB_INVALID_POOLID), set the second parameter **u32BlkSize** to a required size of the buffer, and specify the DDR from whose public VB pools that buffers are obtained. If the specified DDR does not have public VB pool, no buffer is obtained.
- The public VB pool is mainly used for storing the captured pictures of the VIU. Therefore, misoperations performed on the public VB pool (for example, occupying too many buffers) may affect the running of the HiMPP system.

[Example]

See the example of [HI_MPI_VB_CreatePool](#).

[See Also]

[HI_MPI_VB_ReleaseBlock](#)



HI_MPI_VB_ReleaseBlock

[Description]

Releases an obtained buffer.

[Syntax]

```
HI_S32 HI_MPI_VB_ReleaseBlock(VB_BLK Block);
```

[Parameter]

Parameter	Description	Input/Output
Block	Buffer handle	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vb.h, mpi_vb.h
- Library file: libmpi.a

[Note]

If no buffer is available, call this MPI to release buffers.

[Example]

See the example of [HI_MPI_VB_CreatePool](#).

[See Also]

[HI_MPI_VB_GetBlock](#)

HI_MPI_VB_Handle2PhysAddr

[Description]

Obtains the physical address of a buffer.

[Syntax]

```
HI_U32 HI_MPI_VB_Handle2PhysAddr(VB_BLK Block);
```

[Parameter]



Parameter	Description	Input/Output
Block	Buffer handle	Input

[Return Value]

Return Value	Description
0	Invalid value, indicating that the buffer handle is invalid
Other values	The physical address is valid.

[Error Code]

None

[Requirement]

- Header files: hi_comm_vb.h, mpi_vb.h
- Library file: libmpi.a

[Note]

The specified buffer must be a valid buffer obtained from a HiMPP VB pool.

[Example]

See the example of [HI_MPI_VB_CreatePool](#).

[See Also]

None

HI_MPI_VB_Handle2PoolId

[Description]

Obtains the ID of the buffer pool where a frame buffer is located.

[Syntax]

```
VB_POOL HI_MPI_VB_Handle2PoolId (VB_BLK Block);
```

[Parameter]

Parameter	Description	Input/Output
Block	Buffer handle	Input

[Return Value]

Return Value	Description
Positive number or zero	The VB pool ID is valid.



Return Value	Description
Negative number	The VB pool ID is invalid.

[Error Code]

None

[Requirement]

- Header files: hi_comm_vb.h, mpi_vb.h
- Library file: libmpi.a

[Note]

The specified buffer must be a valid buffer obtained from a HiMPP VB pool.

[Example]

```
VB_POOL VbPool;
VB_BLK VbBlk; /* get vb blk id from somewhere */

/* get pool id */
VbPool = HI_MPI_VB_Handle2PoolId(VbBlk);
if (VB_INVALID_POOLID != VbPool)
{
    printf("pool id is %d\n", VbPool);
    /* use pool id do something*/
}
```

[See Also]

None

HI_MPI_VB_MmapPool

[Description]

Maps the user-state virtual address to a VB pool.

[Syntax]

```
HI_S32 HI_MPI_VB_MmapPool(VB_POOL Pool);
```

[Parameter]

Parameter	Description	Input/Output
Pool	VB pool ID Value range: [0, VB_MAX_POOLS)	Input

[Return Value]



Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vb.h, mpi_vb.h
- Library file: libmpi.a

[Note]

- The ID of the buffer pool must be valid.
- A code indicating success is returned if you call this MPI repeatedly.

[Example]

None

[See Also]

- [HI_MPI_VB_MunmapPool](#)
- [HI_MPI_VB_GetBlkVirAddr](#)

HI_MPI_VB_MunmapPool

[Description]

Unmaps the user-state address from a VB pool

[Syntax]

```
HI_S32 HI_MPI_VB_MunmapPool(VB_POOL Pool);
```

[Parameter]

Parameter	Description	Input/Output
Pool	VB pool ID Value range: [0, VB_MAX_POOLS)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]



- Header files: hi_comm_vb.h, mpi_vb.h
- Library file: libmpi.a

[Note]

- The ID of the buffer pool must be valid.
- The VB pool must be mapped.
- The buffers in the VB pool are not used by the MPI layer. If the buffers are used by the MPI layer, the system considers that the mapped user-state virtual address is used, and a code indicating failure is returned.

[Example]

None

[See Also]

- [HI_MPI_VB_MmapPool](#)
- [HI_MPI_VB_GetBlkVirAddr](#)

HI_MPI_VB_GetBlkVirAddr

[Description]

Obtains the user-state virtual address of a buffer in a VB pool.

[Syntax]

```
HI_S32 HI_MPI_VB_GetBlkVirAddr(VB_POOL Pool, HI_U32 u32PhyAddr, HI_VOID **ppVirAddr);
```

[Parameter]

Parameter	Description	Input/Output
Pool	VB pool ID Value range: [0, VB_MAX_POOLS)	Input
u32PhyAddr	Physical address for a buffer	Input
ppVirAddr	User-state virtual address	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vb.h, mpi_vb.h



- Library file: libmpi.a

[Note]

- The ID of the buffer pool and the physical address of the buffer must be valid.
- [HI_MPI_VB_MmapPool](#) must be called.

[Example]

None

[See Also]

- [HI_MPI_VB_MmapPool](#)
- [HI_MPI_VB_MunmapPool](#)

HI_MPI_VB_InitModCommPool

[Description]

Initializes a module public VB pool.

[Syntax]

```
HI_S32 HI_MPI_VB_InitModCommPool(VB\_UID\_E enVbUid);
```

[Parameter]

Parameter	Description	Input/Output
enVbUid	ID of the module the uses the module public VB pool	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vb.h, mpi_vb.h
- Library file: libmpi.a

[Note]

- Currently, the public VB pool applies only to the VDEC.
- Before calling this MPI, you must initialize the public VB pool by calling [HI_MPI_VB_Init](#).
- Before initializing a common public VB pool, you must set its attributes by calling [HI_MPI_VB_SetModPoolConf](#). Otherwise, the VB pool fails to be initialized.
- This MPI can be called repeatedly, and no error code indicating failure is returned.



- The VDEC public VB pool needs to be created only when the mode of allocating the memory for storing decoded frames is the module public VB pool mode (that is, when **VBSOURCE** is **0**). If **VBSOURCE** is **1**, the VDEC public VB pool does not need to be created.

[Example]

See the example of [HI_MPI_VB_SetConf](#).

[See Also]

[HI_MPI_VB_ExitModCommPool](#)

HI_MPI_VB_ExitModCommPool

[Description]

Exits a module public VB pool.

[Syntax]

```
HI_S32 HI_MPI_VB_ExitModCommPool(VB_UID_E enVbUid);
```

[Parameter]

Parameter	Description	Input/Output
enVbUid	ID of the module the uses the module public VB pool	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vb.h, mpi_vb.h
- Library file: libmpi.a

[Note]

- Call this MPI before calling [HI_MPI_VB_Exit](#). Otherwise, an error code indicating failure is returned.
- This MPI can be called repeatedly, and no error code indicating failure is returned.
- After a module public VB pool is exited, its settings are cleared.

[Example]

See the example of [HI_MPI_VB_SetConf](#).

[See Also]



HI_MPI_VB_InitModCommPool

HI_MPI_VB_SetModPoolConf

[Description]

Sets the attributes of a module public VB pool.

[Syntax]

```
HI_S32 HI_MPI_VB_SetModPoolConf(VB_UID_E enVbUid, const VB_CONF_S
*pstVbConf)
```

[Parameter]

Parameter	Description	Input/Output
enVbUid	ID of the module the uses the module public VB pool	Input
pstVbConf	Pointer to the attributes of a module public VB pool Static attribute	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vb.h, mpi_vb.h
- Library file: libmpi.a

[Note]

The module public VB pool must be configured as required. Otherwise, the memory is not sufficiently used.

[Example]

```
HI_S32 s32ret;
VB_CONF_S stVbConf;
VB_UID_E enVbUid = VB_UID_VDEC;

memset(&stVbConf, 0, sizeof(VB_CONF_S));
stVbConf.u32MaxPoolCnt = 128;
stVbConf.astCommPool[0].u32BlkSize = 768*576*2;
stVbConf.astCommPool[0].u32BlkCnt = 20;
stVbConf.astCommPool[1].u32BlkSize = 384*288*2;
```



```
stVbConf.astCommPool[1].u32BlkCnt = 40;

s32ret = HI_MPI_VB_SetModPoolConf(enVbUid, &stVbConf);
if (HI_SUCCESS != s32ret)
{
    printf("set vb err:0x%x\n", s32ret);
    return s32ret;
}
s32ret = HI_MPI_VB_InitModCommPool(enVbUid);
if (HI_SUCCESS != s32ret)
{
    printf("init vb err:0x%x\n", s32ret);
    return s32ret;
}
/* ... ...
(void)HI_MPI_VB_ExitModCommPool(enVbUid);
```

[See Also]

[HI_MPI_VB_GetModPoolConf](#)

HI_MPI_VB_GetModPoolConf

[Description]

Obtains the attributes of a module public VB pool.

[Syntax]

```
HI_S32 HI_MPI_VB_GetModPoolConf(VB_UID_E enVbUid, VB_CONF_S *pstVbConf);
```

[Parameter]

Parameter	Description	Input/Output
enVbUid	ID of the module the uses the module public VB pool	Input
pstVbConf	Pointer to the attributes of a module public VB pool Static attribute	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]



- Header files: hi_comm_vb.h, mpi_vb.h
- Library file: libmpi.a

[Note]

Before calling this MPI, you must set the attributes of a common public VB pool by calling [HI_MPI_VB_SetModPoolConf](#).

[Example]

None

[See Also]

[HI_MPI_VB_SetModPoolConf](#)

HI_MPI_SYS_SetProfile

[Description]

Sets the power consumption scenario. This interface is used to implement the low-power solutions of the media power domain. The operating frequency of each module of the media power domain is controlled by the input parameters of this interface. The voltage of the media power domain is controlled by using this interface and **hixx_pm.ko** of the power consumption module.

[Syntax]

```
HI_S32 HI_MPI_SYS_SetProfile(PROFILE\_TYPE\_E enProfile);
```

[Parameter]

Parameter	Description	Input/Output
enProfile	Type of the power consumption scenario	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vb.h, mpi_vb.h
- Library file: libmpi.a

[Note]

- Ensure that the HiMPP is in the sys_exit state before calling this interface.



- If the running scenario is beyond the range of **PROFILE_TYPE_E**, the frequency of each module can be changed in the **load** script. According to the current configuration in the **load** script, the default scenario is the 5-megapixel scenario for the Hi3516A.
- The function of this MPI is currently not implemented for Hi3518E V200 and Hi3519 V100.

[Example]

None

[See Also]

None

HI_MPI_SYS_GetViVpssMode

[Description]

Obtains the mode (online/offline) information about the VIU/VPSS.

[Syntax]

```
HI_S32 HI_MPI_SYS_GetViVpssMode(HI_U32* pu32Mode);
```

[Parameter]

Parameter	Description	Input/Output
pu32Mode	Offline/Online mode 0: offline 1: online	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]

- Header files: mpi_sys.h
- Library file: libmpi.a

[Note]

The offline/online mode is controlled in the **load** script. This MPI is used only to obtain the mode information.

[Example]

None



[See Also]

None

HI_MPI_SYS_GetVirMemInfo

[Description]

Obtains information about the corresponding memory (including the physical address and cached attributes) based on the virtual address.

[Syntax]

```
HI_S32 HI_MPI_SYS_GetVirMemInfo(const void* pVitAddr, SYS_VIRMEM_INFO_S*  
pstMemInfo);
```

[Parameter]

Parameter	Description	Input/Output
pVitAddr	User-mode virtual address	Input
pstMemInfo	Information about the memory corresponding to the virtual address, including the physical address and cached attributes	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]

- Header files: hi_comm_sys.h, mpi_sys.h
- Library file: libmpi.a

[Note]

- The input must be the user-mode virtual address.
- This MPI cannot be used across processes.

[Example]

None

[See Also]

None



HI_MPI_VB_GetSupplementAddr

[Description]

Obtains the auxiliary information of the VB block memory.

[Syntax]

```
HI_S32 HI_MPI_VB_GetSupplementAddr(VB_BLK_Block, VIDEO_SUPPLEMENT_S  
*pstSupplement);
```

[Parameter]

Parameter	Description	Input/Output
pstSupplement	Auxiliary information of the VB block memory, such as the type of the flash frame and DCF information	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vb.h, mpi_vb.h
- Library file: libmpi.a

[Note]

The DCF address information stored by **pstSupplement** is the kernel-mode virtual address.

[Example]

None

[See Also]

None

HI_MPI_VB_SetSupplementConf

[Description]

Sets the auxiliary control information of the VB memory.

[Syntax]

```
HI_S32 HI_MPI_VB_SetSupplementConf(const VB_SUPPLEMENT_CONF_S  
*pstSupplementConf);
```



[Parameter]

Parameter	Description	Input/Output
pstSupplementConf	Auxiliary information of the VB memory This parameter is used to control some functions related to the VB memory, such as JPEG Design rule for Camera File system (DCF).	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vb.h, mpi_vb.h
- Library file: libmpi.a

[Note]

- An extra piece of VB memory needs to be allocated for the DCF information, and the memory size is equal to the size of the ISP_DCF_INFO_S structure (16-byte aligned).
- Only the JPEG DCF function is supported currently, and the corresponding mask is **VB_SUPPLEMENT_JPEG_MASK**.
- This MPI needs to be called before HI_MPI_VB_Init is called so that the auxiliary information takes effect.
- If the DCF function is enabled, HI_MPI_ISP_Init needs to be called after HI_MPI_VB_Init is called because the ISP determines whether to allocate the memory for storing the DCF information based on the auxiliary VB information during initialization.

[Example]

None

[See Also]

[HI_MPI_VB_GetSupplementConf](#)

HI_MPI_VB_GetSupplementConf

[Description]

Obtains the auxiliary control information of the VB memory.

[Syntax]

```
HI_S32 HI_MPI_VB_GetSupplementConf(VB_SUPPLEMENT_CONF_S  
*pstSupplementConf);
```



[Parameter]

Parameter	Description	Input/Output
pstSupplementConf	Auxiliary information of the VB memory This parameter is used to control some functions related to the VB memory, such as JPEG DCF.	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vb.h, mpi_vb.h
- Library file: libmpi.a

[Note]

None

[Example]

None

[See Also]

[HI_MPI_VB_SetSupplementConf](#)

HI_MPI_SYS_SetScaleCoefLevel

[Description]

Sets the scaling coefficient level of the VPSS and VGS.

[Syntax]

```
HI_S32 HI_MPI_SYS_SetScaleCoefLevel (SCALE_RANGE_S *pstScaleRange,  
SCALE_COEFF_LEVEL_S *pstScaleCoeffLevel);
```

[Parameter]

Parameter	Description	Input/Output
pstScaleRange	Scaling range corresponding to the scaling coefficient	Input
pstScaleCoeffLevel	Scaling coefficient level	Input



[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]

- Header files: hi_comm_sys.h, mpi_sys.h
- Library file: libmpi.a

[Note]

- For details about the default scaling coefficient levels for different scaling ratios, see the **Note** field of [SCALE_COEFF_LEVEL_S](#).
- When the scaling ratio range is **SCALE_RANGE_5** (indicating no scaling), the effect is the same when different scaling coefficient levels are selected.
- When the scaling ratio range remains unchanged, pictures within the scaling ratio range look sharper as the scaling coefficient level is increased, and look more blurred as the scaling coefficient level is decreased. However, there are exceptions. The picture looks sharpest when the scaling coefficients of the horizontal luminance, horizontal chrominance, vertical luminance, and vertical chrominance are COEFF_LEVEL_4, COEFF_LEVEL_3, COEFF_LEVEL_5, and COEFF_LEVEL_3, respectively. That is, the picture looks sharpest when the scaling coefficients within the scaling ratio range of **SCALE_RANGE_5** are used. (This rule does not apply when scaling ratio range is **SCALE_RANGE_5**.)

[Example]

None

[See Also]

None

HI_MPI_SYS_GetScaleCoefLevel

[Description]

Obtains the scaling coefficient level of the VPSS and VGS.

[Syntax]

```
HI_S32 HI_MPI_SYS_GetScaleCoefLevel(SCALE_RANGE_S *pstScaleRange,  
SCALE_COEFF_LEVEL_S *pstScaleCoeffLevel);
```

[Parameter]

Parameter	Description	Input/Output
pstScaleRange	Scaling range corresponding to the scaling coefficient	Input



Parameter	Description	Input/Output
pstScaleCoeffLevel	Scaling coefficient level	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]

- Header files: hi_comm_sys.h, mpi_sys.h
- Library file: libmpi.a

[Note]

None

[Example]

None

[See Also]

None

HI_MPI_SYS_SetTimeZone

[Description]

Sets the time zone information.

[Syntax]

```
HI_S32 HI_MPI_SYS_SetTimeZone(HI_S32 s32TimeZone);
```

[Parameter]

Parameter	Description	Input/Output
s32TimeZone	Time zone, offset relative to the UTC time Unit: second Value range: [-86400, +86400], corresponding to [-24, +24] hours	Input

[Return Value]



Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]

- Header files: hi_comm_sys.h, mpi_sys.h
- Library file: libmpi.a

[Note]

The time zone setting is valid for the JPEG DCF information. After the setting, the photographing time (UTC time) will be converted into the local time.

[Example]

None

[See Also]

None

HI_MPI_SYS_GetTimeZone

[Description]

Obtains the time zone information.

[Syntax]

```
HI_S32 HI_MPI_SYS_GetTimeZone(HI_S32 *ps32TimeZone);
```

[Parameter]

Parameter	Description	Input/Output
ps32TimeZone	Time zone, offset relative to the UTC time Unit: second	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 2.5 "Error Codes."

[Requirement]

- Header files: hi_comm_sys.h, mpi_sys.h



- Library file: libmpi.a

[Note]

None

[Example]

None

[See Also]

None

2.4 Data Structures

2.4.1 Basic Data Structures

The basic data structures are defined as follows:

Common Data Structures

```
typedef unsigned char           HI_U8;
typedef unsigned short          HI_U16;
typedef unsigned int            HI_U32;

typedef signed char            HI_S8;
typedef short                  HI_S16;
typedef int                    HI_S32;

#ifndef _M_IX86
    typedef unsigned long long  HI_U64;
    typedef long long           HI_S64;
#else
    typedef __int64              HI_U64;
    typedef __int64              HI_S64;
#endif

typedef char                   HI_CHAR;
#define HI_VOID                 void

/*-----
 * const definition
 *-----*/
typedef enum {
    HI_FALSE = 0,
    HI_TRUE  = 1,
} HI_BOOL;
```



```
#ifndef NULL
#define NULL    0L
#endif

#define HI_NULL      0L
#define HI_SUCCESS   0
#define HI_FAILURE  (-1)

typedef HI_S32 AI_CHN;
typedef HI_S32 AO_CHN;
typedef HI_S32 AENC_CHN;
typedef HI_S32 ADEC_CHN;
typedef HI_S32 AUDIO_DEV;
typedef HI_S32 AVENC_CHN;
typedef HI_S32 VI_DEV;
typedef HI_S32 VI_WAY;
typedef HI_S32 VI_CHN;
typedef HI_S32 VO_DEV;
typedef HI_S32 VO_LAYER;
typedef HI_S32 VO_CHN;
typedef HI_S32 VO_WBC;
typedef HI_S32 GRAPHIC_LAYER;
typedef HI_S32 VENC_CHN;
typedef HI_S32 VDEC_CHN;
typedef HI_S32 VENC_GRP;
typedef HI_S32 VO_GRP;
typedef HI_S32 VDA_CHN;
typedef HI_S32 IVE_HANDLE;
typedef HI_S32 CLS_HANDLE;
typedef HI_S32 FD_CHN;
typedef HI_S32 MD_CHN;
typedef HI_S32 ISP_DEV;
typedef HI_S32 SENSOR_ID;
typedef HI_S32 MIPI_DEV;

/*Invalid channel ID, invalid way ID, invalid device ID, and invalid
handle ID*/
#define HI_INVALID_CHN (-1)
#define HI_INVALID_WAY (-1)
#define HI_INVALID_DEV (-1)
#define HI_INVALID_HANDLE (-1)
#define HI_INVALID_VALUE (-1)
#define HI_INVALID_TYPE (-1)
```



```
/*Maximum number of VB pools*/
#define VB_MAX_POOLS                256

/*Maximum number of public VB pools*/
#define VB_MAX_COMM_POOLS           16
/*Length of the MMZ name*/
#define MAX_MMZ_NAME_LEN            16
/*JPEG DCF mask for the auxiliary VB information */
#define VB_SUPPLEMENT_JPEG_MASK    0x1
```

Besides the preceding data structures, the following basic data structures are provided:

- [POINT_S](#): Defines the coordinate information.
- [SIZE_S](#): Defines the dimension information.
- [RECT_S](#): Defines the rectangle information.
- [MOD_ID_E](#): Defines the module ID enumeration.
- [VB_UID_E](#): Defines the enumeration type of the VB pool ID.
- [MPP_CHN_S](#): Defines the module, device, and channel.
- [BORDER_S](#): Defines the border attributes.
- [ROTATE_E](#): Defines the rotation enumeration.
- [WDR_MODE_E](#): Defines the wide dynamic range (WDR) mode enumeration.
- [CROP_INFO_S](#): Defines crop attributes.
- [PAYLOAD_TYPE_E](#): Defines the audio or video payload type.

POINT_S

[Description]

Defines the coordinate information.

[Syntax]

```
typedef struct hiPOINT_S
{
    HI_S32 s32X;
    HI_S32 s32Y;
} POINT_S;
```

[Member]

Member	Description
s32X	Horizontal coordinate
s32Y	Vertical coordinate



[Note]

None

[See Also]

None

SIZE_S

[Description]

Defines the dimension information.

[Syntax]

```
typedef struct hiSIZE_S
{
    HI_U32 u32Width;
    HI_U32 u32Height;
} SIZE_S;
```

[Member]

Member	Description
u32Width	Width
u32Height	Height

[Note]

None

[See Also]

None

RECT_S

[Description]

Defines the rectangle information.

[Syntax]

```
typedef struct hiRECT_S
{
    HI_S32    s32X;
    HI_S32    s32Y;
    HI_U32    u32Width;
    HI_U32    u32Height;
} RECT_S;
```

[Member]



Member	Description
s32X	Horizontal coordinate
s32Y	Vertical coordinate
u32Width	Width
u32Height	Height

[Note]

None

[See Also]

- RGN_ATTR_S
- OVERLAY_ATTR_S
- VI_CHN_ATTR_S
- VO_CHN_ATTR_S

MOD_ID_E

[Description]

Defines the module ID enumeration.

[Syntax]

```
typedef enum hiMOD_ID_E
{
    HI_ID_CMPI      = 0,
    HI_ID_VB        = 1,
    HI_ID_SYS       = 2,
    HI_ID_RGN       = 3,
    HI_ID_CHNL      = 4,
    HI_ID_VDEC      = 5,
    HI_ID_GROUP     = 6,
    HI_ID_VPSS      = 7,
    HI_ID_VENC      = 8,
    HI_ID_VDA       = 9,
    HI_ID_H264E     = 10,
    HI_ID_JPEGE     = 11,
    HI_ID_MPEG4E    = 12,
    HI_ID_H264D     = 13,
    HI_ID_JPEGD     = 14,
    HI_ID_VOU       = 15,
    HI_ID_VIU       = 16,
    HI_ID_DSU       = 17,
    HI_ID_VALG      = 18,
```



```
    HI_ID_RC      = 19,
    HI_ID_AIO     = 20,
    HI_ID_AI      = 21,
    HI_ID_AO      = 22,
    HI_ID_AENC    = 23,
    HI_ID_ADEC    = 24,
    HI_ID_AVENC   = 25,
    HI_ID_PCIV    = 26,
    HI_ID_PCIVFMW = 27,
    HI_ID_ISP     = 28,
    HI_ID_IVE     = 29,
    HI_ID_DCCM    = 31,
    HI_ID_DCCS    = 32,
    HI_ID_PROC    = 33,
    HI_ID_LOG     = 34,
    HI_ID_MST_LOG = 35,
    HI_ID_VD      = 36,
    HI_ID_VCMP    = 38,
    HI_ID_FB      = 39,
    HI_ID_HDMI    = 40,
    HI_ID_VOIE    = 41,
    HI_ID_TDE     = 42,
    HI_ID_USR     = 43,
    HI_ID_VEDU    = 44,
    HI_ID_VGS     = 45,
    HI_ID_H265E   = 46,
    HI_ID_FD      = 47,
    HI_ID_ODT     = 48, //Object detection trace
    HI_ID_VQA     = 49, //Video quality analysis
    HI_ID_LPR     = 50, // License Plate Recognition
    HI_ID_FISHEYE = 51,
    HI_ID_BUTT,
} MOD_ID_E;
```

[Member]

None

[Note]

None

[See Also]

None



VB_UID_E

[Description]

Defines the enumeration type of the VB pool ID.

[Syntax]

```
typedef enum hiVB_UID_E
{
    VB_UID_VIU          = 0,
    VB_UID_VOU          = 1,
    VB_UID_VGS          = 2,
    VB_UID_VENC         = 3,
    VB_UID_VDEC         = 4,
    VB_UID_VDA          = 5,
    VB_UID_H264E        = 6,
    VB_UID_JPEGE        = 7,
    VB_UID_MPEG4E       = 8,
    VB_UID_H264D        = 9,
    VB_UID_JPEGD        = 10,
    VB_UID_MPEG4D       = 11,
    VB_UID_VPSS         = 12,
    VB_UID_GRP          = 13,
    VB_UID_MPI          = 14,
    VB_UID_PCIV         = 15,
    VB_UID_AI           = 16,
    VB_UID_AENC         = 17,
    VB_UID_RC           = 18,
    VB_UID_VFMW         = 19,
    VB_UID_USER         = 20,
    VB_UID_H265E        = 21,
    VB_UID_BUTT
}
```

} VB_UID_E;

[Member]

None

[Note]

None

[See Also]

None

MPP_CHN_S

[Description]



Defines the module, device, and channel.

[Syntax]

```
typedef struct hiMPP_CHN_S
{
    MOD_ID_E    enModId;
    HI_S32      s32DevId;
    HI_S32      s32ChnId;
} MPP_CHN_S;
```

[Member]

Member	Description
enModId	Module ID
s32DevId	Device ID
s32ChnId	Channel ID

[Note]

None

[See Also]

- [HI_MPI_SYS_Bind](#)
- [HI_MPI_SYS_UnBind](#)
- [HI_MPI_SYS_GetBindbyDest](#)

BORDER_S

[Description]

Defines the border attributes.

[Syntax]

```
typedef struct hiBORDER_S
{
    HI_U32 u32TopWidth;           /* top border weight, in pixel*/
    HI_U32 u32BottomWidth;        /* bottom border weight, in pixel*/
    HI_U32 u32LeftWidth;          /* left border weight, in pixel*/
    HI_U32 u32RightWidth;         /* right border weight, in pixel*/
    HI_U32 u32Color;              /* border color, RGB888*/
} BORDER_S;
```

[Member]

Member	Description
u32TopWidth	Top border width, in pixel



Member	Description
u32BottomWidth	Bottom border width, in pixel
u32LeftWidth	Left border width, in pixel
u32RightWidth	Right border width, in pixel
u32Color	Border color. The color format is RGB888. Only the lower 24 bits of u32Color are valid.

[Note]

None

[See Also]

- VO_BORDER_S
- VPSS_CHN_ATTR_S

ROTATE_E

[Description]

Defines the rotation enumeration.

[Syntax]

```
typedef enum hiROTATE_E
{
    ROTATE_NONE = 0,
    ROTATE_90     = 1,
    ROTATE_180    = 2,
    ROTATE_270    = 3,
    ROTATE_BUTT
} ROTATE_E;
```

[Member]

Member	Description
ROTATE_NONE	No rotation
ROTATE_90	Clockwise rotation by 90°
ROTATE_180	Clockwise rotation by 180°
ROTATE_270	Clockwise rotation by 270°

[Note]

None

[See Also]



- HI_MPI_VI_SetRotate
- HI_MPI_VI_GetRotate
- HI_MPI_VPSS_SetRotate
- HI_MPI_VPSS_GetRotate

WDR_MODE_E

[Description]

Defines the WDR mode enumeration.

[Syntax]

```
typedef enum hiWDR_MODE_E
{
    WDR_MODE_NONE = 0,
    WDR_MODE_BUILT_IN,

    WDR_MODE_2To1_LINE,
    WDR_MODE_2To1_FRAME,
    WDR_MODE_2To1_FRAME_FULL_RATE,

    WDR_MODE_3To1_LINE,
    WDR_MODE_3To1_FRAME,
    WDR_MODE_3To1_FRAME_FULL_RATE,

    WDR_MODE_4To1_LINE,
    WDR_MODE_4To1_FRAME,
    WDR_MODE_4To1_FRAME_FULL_RATE,

    WDR_MODE_BUTT,
} WDR_MODE_E;
```

[Member]

Member	Description
WDR_MODE_NONE	Linear mode
WDR_MODE_BUILT_IN	Sensor combination WDR mode
WDR_MODE_2To1_LINE	WDR mode in which two frames are combined and output as crossed lines
WDR_MODE_2To1_FRAME	WDR mode in which two frames are combined and output as a frame
WDR_MODE_2To1_FRAME_FULL_RATE	WDR mode in which two frames are combined and output as a frame at full frame rate



Member	Description
WDR_MODE_3To1_LINE	WDR mode in which three frames are combined and output as crossed lines
WDR_MODE_3To1_FRAME	WDR mode in which three frames are combined and output as a frame
WDR_MODE_3To1_FRAME_FULL_RATE	WDR mode in which three frames are combined and output as a frame at full frame rate
WDR_MODE_4To1_LINE	WDR mode in which four frames are combined and output as crossed lines
WDR_MODE_4To1_FRAME	WDR mode in which four frames are combined and output as a frame
WDR_MODE_4To1_FRAME_FULL_RATE	WDR mode in which four frames are combined and output as a frame at full frame rate

[Note]

None

[See Also]

- HI_MPI_VI_SetWDRAttr
- HI_MPI_VI_GetWDRAttr
- HI_MPI_ISP_SetWDRMode
- HI_MPI_ISP_GetWDRMode

CROP_INFO_S

[Description]

Defines crop attributes.

[Syntax]

```
typedef struct hicrop_info_s
{
    HI_BOOL bEnable;
    RECT_S stRect;
} CROP_INFO_S;
```

[Member]

Member	Description
bEnable	Crop enable
stRect	Crop start position, width, and height



[Note]

None

[See Also]

- [HI_MPI_VI_SetDevCrop](#)
- [HI_MPI_VI_GetDevCrop](#)

PAYLOAD_TYPE_E

[Description]

Defines the audio or video payload type.

[Syntax]

```
typedef enum
{
    PT_PCMU = 0,
    PT_1016 = 1,
    PT_G721 = 2,
    PT_GSM = 3,
    PT_G723 = 4,
    PT_DVI4_8K = 5,
    PT_DVI4_16K = 6,
    PT_LPC = 7,
    PT_PCMA = 8,
    PT_G722 = 9,
    PT_S16BE_STEREO = 10,
    PT_S16BE_MONO = 11,
    PT_QCELP = 12,
    PT_CN = 13,
    PT_MPEGAUDIO = 14,
    PT_G728 = 15,
    PT_DVI4_3 = 16,
    PT_DVI4_4 = 17,
    PT_G729 = 18,
    PT_G711A = 19,
    PT_G711U = 20,
    PT_G726 = 21,
    PT_G729A = 22,
    PT_LPCM = 23,
    PT_CelB = 25,
    PT_JPEG = 26,
    PT_CUSM = 27,
    PT_NV = 28,
```



```
PT_PICW = 29,  
PT_CPV = 30,  
PT_H261 = 31,  
PT_MPEGVIDEO = 32,  
PT_MPEG2TS = 33,  
PT_H263 = 34,  
PT_SPEG = 35,  
PT_MPEG2VIDEO = 36,  
PT_AAC = 37,  
PT_WMA9STD = 38,  
PT_HEAAC = 39,  
PT_PCM_VOICE = 40,  
PT_PCM_AUDIO = 41,  
PT_AACLC = 42,  
PT_MP3 = 43,  
PT_ADPCM = 49,  
PT_AEC = 50,  
PT_X_LD = 95,  
PT_H264 = 96,  
PT_D_GSM_HR = 200,  
PT_D_GSM_EFR = 201,  
PT_D_L8 = 202,  
PT_D_RED = 203,  
PT_D_VDVI = 204,  
PT_D_BT656 = 220,  
PT_D_H263_1998 = 221,  
PT_D_MP1S = 222,  
PT_D_MP2P = 223,  
PT_D_BMPEG = 224,  
PT_MP4VIDEO = 230,  
PT_MP4AUDIO = 237,  
PT_VC1 = 238,  
PT_JVC ASF = 255,  
PT_D_AVI = 256,  
PT_MAX = 257,  
PT_DIVX3 = 257,  
PT_AVIS = 258,  
PT_REAL8 = 259,  
PT_REAL9 = 260,  
PT_VP6 = 261,  
PT_VP6F = 262,  
PT_VP6A = 263,  
PT_SORENSEN = 264,  
PT_H265 = 265,
```



```
PT_MAX      = 266,  
PT_AMR = 1001, /* add by mpp */  
PT_MJPEG = 1002,  
PT_AMRWB = 1003,  
PT_BUTT  
} PAYLOAD_TYPE_E;
```

[Member]

None

[Note]

None

[See Also]

- VENC_CHN_ATTR_S
- VDEC_CHN_ATTR_S
- AENC_CHN_ATTR_S
- ADEC_CHN_ATTR_S

PROFILE_TYPE_E

[Description]

Defines the enumeration of power consumption scenarios.

[Syntax]

```
typedef enum hiPROFILE_TYPE_E  
{  
    PROFILE_1080P_30 = 0,  
    PROFILE_3M_30,  
    PROFILE_1080P_60,  
    PROFILE_5M_30,  
    PROFILE_BUTT,  
} PROFILE_TYPE_E;
```

[Member]

Member	Description
PROFILE_1080P_30	Power consumption scenario: 1080p@ 30 fps capturing and encoding
PROFILE_3M_30	Power consumption scenario: 3-megapixel (2048 x 1536)@30 fps capturing and encoding
PROFILE_1080P_60	Power consumption scenario: 1080p@ 60 fps capturing and encoding
PROFILE_5M_30	Power consumption scenario: 5-megapixel (2592 x 1944)@30 fps capturing and encoding



[Note]

None

2.4.2 System Control Data Structures

The system control data structures are as follows:

- [MPP_SYS_CONF_S](#): Defines HiMPP system control attributes.
- [VB_CONF_S](#): Defines VB pool attributes.
- [MPP_VERSION_S](#): Defines the structure of the HiMPP version information.
- [SYS_VIRMEM_INFO_S](#): Defines information about a virtual memory.
- [SCALE_RANGE_E](#): Defines the enumerated types of the scaling ratio range for the VPSS and VGS.
- [COEFF_LEVEL_E](#): Defines the enumerated types of the scaling coefficient level for the VPSS and VGS.
- [SCALE_RANGE_S](#): Defines the structure of the horizontal and vertical scaling ratio ranges for the VPSS and VGS.
- [SCALE_COEFF_LEVEL_S](#): Defines the structure of the scaling coefficient level for the horizontal luminance, horizontal chrominance, vertical luminance, and vertical chrominance for the VPSS and VGS.

MPP_SYS_CONF_S

[Description]

Defines HiMPP system control attributes.

[Syntax]

```
typedef struct hiMPP_SYS_CONF_S
{
    /* stride of picture buffer must be aligned with this value.
     * you can choose a value from 1 to 1024,
     * and it except 1 must be multiple of 16.*/
    HI_U32 u32AlignWidth;
}MPP_SYS_CONF_S;
```

[Member]

Member	Description
u32AlignWidth	Stride for aligning the pictures used in the system, in byte Value range: [1, 1024]. 16-byte alignment is recommended. Static attribute

[Note]

None



[See Also]

[HI_MPI_SYS_SetConf](#)

VB_CONF_S

[Description]

Defines VB pool attributes.

[Syntax]

```
typedef struct hiVB_CONF_S
{
    HI_U32    u32MaxPoolCnt; /* max count of pools, (0,VB_MAX_POOLS] */
    Struct    hiVB_CPOOL_S
    {
        HI_U32    u32BlkSize;
        HI_U32    u32BlkCnt;
        HI_CHAR   acMmzName [MAX_MMZ_NAME_LEN];
    }astCommPool [VB_MAX_COMM_POOLS];
```

[Member]

Member	Description
u32MaxPoolCnt	Maximum number of VB pools in the entire system Value range: (0, VB_MAX_POOLS) Static attribute The value is fixed at VB_MAX_POOLS at present.
astCommPool	Attributes of the public VB pool. The members include the size (in byte) of each buffer, number of buffers in the public VB pool, and the name of the DDR where the VB pool is located. Static attribute

[Note]

The size of each buffer block in the public VB pool varies according to the current picture pixel format and the picture compression mode (compressed or uncompressed). See [Table 2-2](#).

Table 2-2 Calculation table for the VB size

Pixel Format	Whether the Picture Is Compressed	
	Compressed	Uncompressed
PIXEL_FORMAT_YUV_SEMIPLANAR_420	(Width + Stride of the compression header) x Height x 1.5	Width x Height x 1.5



Pixel Format	Whether the Picture Is Compressed	
	Compressed	Uncompressed
PIXEL_FORMAT_YUV_SEMIPLANAR_422	(Width + Stride of the compression header) x Height x 2	Width x Height x 2
PIXEL_FORMAT_YUV_400	(Width + Stride of the compression header) x Height	Width x Height

In [Table 2-2](#), compression indicates the 256-byte segment-based compression (only this compression mode is supported currently). In addition, the width is 16-pixel-aligned, the height is 2-pixel-aligned, and the stride of the compression header is 16. During compression, the size of the picture compression header can be obtained by directly calling `VB_PIC_HEADER_SIZE`, and the total picture size can be obtained by directly calling `VB_PIC_BLK_SIZE`. Both macro definitions are in `hi_define.h`.

[See Also]

[HI_MPI_SYS_SetConf](#)

MPP_VERSION_S

[Description]

Defines the structure of the HiMPP version information.

[Syntax]

```
typedef struct hiMPP_VERSION_S
{
    HI_CHAR aVersion[64];
}MPP_VERSION_S;
```

[Member]

Member	Description
aVersion	String for describing the version information, for example, HI_VERSION=Hi35xx_MPP_V1.0.4.0.

[Note]

None

[See Also]

[HI_MPI_SYS_GetVersion](#)

SYS_VIRMEM_INFO_S

[Description]



Defines information about a virtual memory.

[Syntax]

```
typedef struct hiSYS_VIRMEM_INFO_S
{
    HI_U32 u32PhyAddr;
    HI_BOOL bCached;
} SYS_VIRMEM_INFO_S;
```

[Member]

Member	Description
u32PhyAddr	Physical address corresponding to the virtual address
bCached	Cached mapping indicator

[Note]

None

[See Also]

[HI_MPI_SYS_GetVirMemInfo](#)

SCALE_RANGE_E

[Description]

Defines the enumerated types of the scaling ratio range for the VPSS and VGS.

[Syntax]

```
typedef enum hiSCALE_RANGE_E
{
    SCALE_RANGE_0 = 0,      /* scale range < 1/4 */
    SCALE_RANGE_1,          /* scale range >= 1/4 */
    SCALE_RANGE_2,          /* scale range >= 1/3 */
    SCALE_RANGE_3,          /* scale range >= 1/2 */
    SCALE_RANGE_4,          /* scale range >= 3/4 */
    SCALE_RANGE_5,          /* scale range == 1/1 */
    SCALE_RANGE_6,          /* scale range > 1 */
    SCALE_RANGE_BUTT,
} SCALE_RANGE_E;
```

[Member]

None

[Note]

None



[See Also]

- [HI_MPI_SYS_SetScaleCoefLevel](#)
- [HI_MPI_SYS_GetScaleCoefLevel](#)

COEFF_LEVEL_E

[Description]

Defines the enumerated types of the scaling coefficient level for the VPSS and VGS.

[Syntax]

```
typedef enum hiCOEFF_LEVEL_E
{
    COEFF_LEVEL_0 = 0,      /* coefficient level 0 */
    COEFF_LEVEL_1,         /* coefficient level 1 */
    COEFF_LEVEL_2,         /* coefficient level 2 */
    COEFF_LEVEL_3,         /* coefficient level 3 */
    COEFF_LEVEL_4,         /* coefficient level 4 */
    COEFF_LEVEL_5,         /* coefficient level 5 */
    COEFF_LEVEL_6,         /* coefficient level 6 */
    COEFF_LEVEL_BUTT,
} COEFF_LEVEL_E;
```

[Member]

None

[Note]

None

[See Also]

- [HI_MPI_SYS_SetScaleCoefLevel](#)
- [HI_MPI_SYS_GetScaleCoefLevel](#)

SCALE_RANGE_S

[Description]

Defines the structure of the horizontal and vertical scaling ratio ranges for the VPSS and VGS.

[Syntax]

```
typedef struct hiSCALE_RANGE_S
{
    SCALE_RANGE_E enHorizontal;
    SCALE_RANGE_E enVertical;
} SCALE_RANGE_S;
```

[Member]



Member	Description
enHorizontal	Range of the horizontal scaling ratio
enVertical	Range of the vertical scaling ratio

[Note]

None

[See Also]

- [HI_MPI_SYS_SetScaleCoefLevel](#)
- [HI_MPI_SYS_GetScaleCoefLevel](#)

SCALE_COEFF_LEVEL_S

[Description]

Defines the structure of the scaling coefficient level for the horizontal luminance, horizontal chrominance, vertical luminance, and vertical chrominance for the VPSS and VGS.

[Syntax]

```
typedef struct hiSCALE_COEFF_LEVEL_S
{
    COEFF_LEVEL_E enHorLum; /* horizontal luminance coefficient level */
    COEFF_LEVEL_E enHorChr; /* horizontal chrominance coefficient level */
    COEFF_LEVEL_E enVerLum; /* vertical luminance coefficient level */
    COEFF_LEVEL_E enVerChr; /* vertical chrominance coefficient level */
} SCALE_COEFF_LEVEL_S;
```

[Member]

Member	Description
enHorLum	Scaling coefficient level of horizontal luminance Value range: [COEFF_LEVEL_0, COEFF_LEVEL_5]
enHorChr	Scaling coefficient level of horizontal chrominance Value range: [COEFF_LEVEL_0, COEFF_LEVEL_3]
enVerLum	Scaling coefficient level of vertical luminance Value range: [COEFF_LEVEL_0, COEFF_LEVEL_6]
enVerChr	Scaling coefficient level of vertical chrominance Value range: [COEFF_LEVEL_0, COEFF_LEVEL_3]

[Note]

The following table lists the scaling coefficient levels when different scaling ratios are used.



Note: SR_x indicates SCALE_RANGE_x and CL_x indicates COEFF_LEVEL_x.

Scaling Ratio	Horizontal Luminance	Horizontal Chrominance	Vertical Luminance	Vertical Chrominance
SR_0	CL_0	CL_0	CL_0	CL_0
SR_1	CL_1	CL_0	CL_1	CL_0
SR_2	CL_2	CL_1	CL_2	CL_1
SR_3	CL_3	CL_2	CL_3	CL_2
SR_4	CL_3	CL_2	CL_4	CL_2
SR_5	CL_4	CL_3	CL_5	CL_3
SR_6	CL_5	CL_2	CL_6	CL_2

[See Also]

- [HI_MPI_SYS_SetScaleCoefLevel](#)
- [HI_MPI_SYS_GetScaleCoefLevel](#)

2.4.3 Public Video Data Structures

The public video data structures are as follows:

- [VIDEO_NORM_E](#): Defines the video input standard.
- [PIXEL_FORMAT_E](#): Defines the pixel format.
- [VIDEO_FIELD_E](#): Defines the frame or field type of a video picture.
- [FRAME_FLASH_TYPE_E](#): Defines the type of the flash frame.
- [VIDEO_SUPPLEMENT_S](#): Defines the supplementary information about a video picture frame.
- [VIDEO_FRAME_S](#): Defines the original video picture frame.
- [VIDEO_FRAME_INFO_S](#): Defines the video picture information.
- [BITMAP_S](#): Defines the bitmap information.
- [VIDEO_VBI_INFO_S](#): Defines the video vertical blanking interval (VBI) information.
- [VIDEO_FORMAT_E](#): Defines the video format.
- [COMPRESS_MODE_E](#): Defines the video compression format.
- [VB_SUPPLEMENT_CONF_S](#): Defines the structure of the auxiliary VB information.

VIDEO_NORM_E

[Description]

Defines the video input standard.

[Syntax]

```
typedef enum hiVIDEO_NORM_E
{
```



```
VIDEO_ENCODING_MODE_PAL=0,  
VIDEO_ENCODING_MODE_NTSC,  
VIDEO_ENCODING_MODE_AUTO,  
VIDEO_ENCODING_MODE_BUTT,  
} VIDEO_NORM_E;
```

[Member]

Member	Description
VIDEO_ENCODING_MODE_PAL	Phase alternating line (PAL)
VIDEO_ENCODING_MODE_NTSC	National Television Systems Committee (NTSC)
VIDEO_ENCODING_MODE_AUTO	Automatic

[Note]

The automatic mode is not supported.

[See Also]

None

PIXEL_FORMAT_E

[Description]

Defines the pixel format.

[Syntax]

```
typedef enum hiPIXEL_FORMAT_E  
{  
    PIXEL_FORMAT_RGB_1BPP = 0,  
    PIXEL_FORMAT_RGB_2BPP,  
    PIXEL_FORMAT_RGB_4BPP,  
    PIXEL_FORMAT_RGB_8BPP,  
    PIXEL_FORMAT_RGB_444,  
    PIXEL_FORMAT_RGB_4444,  
    PIXEL_FORMAT_RGB_555,  
    PIXEL_FORMAT_RGB_565,  
    PIXEL_FORMAT_RGB_1555,  
    /* 9 reserved */  
    PIXEL_FORMAT_RGB_888,  
    PIXEL_FORMAT_RGB_8888,  
    PIXEL_FORMAT_RGB_PLANAR_888,  
    PIXEL_FORMAT_RGB_BAYER_8BPP,  
    PIXEL_FORMAT_RGB_BAYER_10BPP,  
    PIXEL_FORMAT_RGB_BAYER_12BPP,
```



```
PIXEL_FORMAT_RGB_BAYER_14BPP,
PIXEL_FORMAT_RGB_BAYER,           /* 16 bpp */
PIXEL_FORMAT_YUV_A422,
PIXEL_FORMAT_YUV_A444,
PIXEL_FORMAT_YUV_PLANAR_422,
PIXEL_FORMAT_YUV_PLANAR_420,
PIXEL_FORMAT_YUV_PLANAR_444,
PIXEL_FORMAT_YUV_SEMIPLANAR_422,
PIXEL_FORMAT_YUV_SEMIPLANAR_420,
PIXEL_FORMAT_YUV_SEMIPLANAR_444,
PIXEL_FORMAT_UYVY_PACKAGE_422,
PIXEL_FORMAT_YUYV_PACKAGE_422,
PIXEL_FORMAT_VYUY_PACKAGE_422,
PIXEL_FORMAT_YCbCr_PLANAR,
PIXEL_FORMAT_YUV_400,           //Y
PIXEL_FORMAT_BUTT
} PIXEL_FORMAT_E;
```

[Member]

None

[Note]

None

[See Also]

- VI_CHN_ATTR_S
- OVERLAY_ATTR_S

VIDEO_FIELD_E

[Description]

Defines the frame/field type of a video picture.

[Syntax]

```
typedef enum hiVIDEO_FIELD_E
{
    VIDEO_FIELD_TOP      = 0x01,    /*Even field*/
    VIDEO_FIELD_BOTTOM   = 0x02,    /*Odd field*/
    VIDEO_FIELD_INTERLACED = 0x03,   /*Two interlaced fields*/
    VIDEO_FIELD_FRAME    = 0x04,    /*Frame*/
    VIDEO_FIELD_BUTT
} VIDEO_FIELD_E;
```

[Member]



Member	Description
VIDEO_FIELD_TOP	Top field
VIDEO_FIELD_BOTTOM	Bottom field
VIDEO_FIELD_INTERLACED	Interlaced
VIDEO_FIELD_FRAME	Frame

[Note]

None

[See Also]

[VIDEO_FRAME_S](#)

FRAME_FLASH_TYPE_E

[Description]

Defines the type of the flash frame.

[Syntax]

```
typedef enum hiFRAME_FLASH_TYPE_E
{
    FRAME_FLASH_OFF      = 0,
    FRAME_FLASH_ON       = 1,
    FRAME_FLASH_BUTT
} FRAME_FLASH_TYPE_E;
```

[Member]

Member	Description
FRAME_FLASH_OFF	The flash is turned off.
FRAME_FLASH_ON	The flash is turned on.

[Note]

None

[See Also]

[VIDEO_SUPPLEMENT_S](#)

VIDEO_SUPPLEMENT_S

[Description]

Defines the supplementary information about a video picture frame.



[Syntax]

```
typedef struct hiVIDEO_SUPPLEMENT_S
{
    HI_U32    u32JpegDcfPhyAddr;
    HI_VOID *pJpegDcfVirAddr;
    FRAME_FLASH_TYPE_E enFlashType;
}VIDEO_SUPPLEMENT_S;
```

[Member]

Member	Description
u32JpegDcfPhyAddr	Physical address of the JPEG DCF information
pJpegDcfVirAddr	Kernel-mode virtual address of the JPEG DCF information
enFlashType	Snapshot frame flag

[Note]

None

[See Also]

[VIDEO_FRAME_S](#)

VIDEO_FRAME_S

[Description]

Defines the original video picture frame.

[Syntax]

```
typedef struct hiVIDEO_FRAME_S
{
    HI_U32          u32Width;
    HI_U32          u32Height;
    VIDEO_FIELD_E   u32Field;
    PIXEL_FORMAT_E  enPixelFormat;
    VIDEO_FORMAT_E  enVideoFormat;
    COMPRESS_MODE_E enCompressMode;
    HI_U32          u32PhyAddr[3];
    HI_VOID         *pVirAddr[3];
    HI_U32          u32Stride[3];
    HI_U32          u32HeaderPhyAddr[3];
    HI_VOID         *pHeaderVirAddr[3];
    HI_U32          u32HeaderStride[3];
    HI_S16          s16OffsetTop; /* top offset of show area */
    HI_S16          s16OffsetBottom; /* bottom offset of show area */
}
```



```
    HI_S16          s16OffsetLeft;      /* left offset of show area */
    HI_S16          s16OffsetRight;     /* right offset of show area*/
    HI_U64          u64pts;
    HI_U32          u32TimeRef;
    HI_U32          u32PrivateData;
    VIDEO_SUPPLEMENT_S stSupplement;
}VIDEO_FRAME_S;
```

[Member]

Member	Description
u32Width	Picture width
u32Height	Picture height
u32Field	Frame/Field mode
enPixelFormat	Pixel format of a video picture
enVideoFormat	Video picture format
enCompressMode	Video compression mode
u32PhyAddr	Physical address
pVirAddr	Virtual address
u32Stride	Picture stride
u32HeaderPhyAddr	Physical address of the picture compression header
pHeaderVirAddr	Virtual address of the picture compression header
u32HeaderStride	Stride of the picture compression header
u16OffsetTop	Top offset of the displayed area
u16OffsetBottom	Bottom offset of the displayed area
u16OffsetLeft	Left offset of the displayed area
u16OffsetRight	Right offset of the displayed area
u64pts	Picture PTS
u32TimeRef	Frame sequence number of a picture
u32PrivateData	Private data
stSupplement	Supplementary information of the picture

[Note]

None

[See Also]



VI_PUB_ATTR_S

VIDEO_FRAME_INFO_S

[Description]

Defines the video picture information.

[Syntax]

```
typedef struct hiVIDEO_FRAME_INFO_S
{
    VIDEO_FRAME_S stVFrame;
    HI_U32          u32PoolId;
}VIDEO_FRAME_INFO_S;
```

[Member]

Member	Description
stVFrame	Video picture frame
u32PoolId	VB pool ID

[Note]

None

[See Also]

[VIDEO_FRAME_S](#)

BITMAP_S

[Description]

Defines the bitmap information.

[Syntax]

```
typedef struct hiBITMAP_S
{
    PIXEL_FORMAT_E enPixelFormat;
    HI_U32          u32Width;
    HI_U32          u32Height;
    HI_VOID         *pData;
} BITMAP_S;
```

[Member]

Member	Description
enPixelFormat	Pixel format of a bitmap



Member	Description
u32Width	Bitmap width
u32Height	Bitmap height
pData	Bitmap data

[Note]

None

[See Also]

None

VIDEO_VBI_INFO_S

[Description]

Defines the video VBI information.

[Syntax]

```
typedef struct hiVIDEO_VBI_INFO_S
{
    HI_U32 au32Data[VIU_MAX_VBI_LEN];
    HI_U32 u32Len;
}VIDEO_VBI_INFO_S;
```

[Member]

Member	Description
au32Data	VBI data
u32Len	VBI data length

[Note]

None

VIDEO_FORMAT_E

[Description]

Defines the video format.

[Syntax]

```
typedef enum hiVIDEO_FORMAT_E
{
    VIDEO_FORMAT_LINEAR      = 0x0,        /*Nature video line */
    VIDEO_FORMAT_TILE         = 0x1,        /*Tile cell: 256 pixel x 16 line,
}
```



```
default tile mode */
VIDEO_FORMAT_TILE64      = 0x2,      /*Tile cell: 64 pixel x 16 line */
VIDEO_FORMAT_BUTT
} VIDEO_FORMAT_E;
```

[Member]

Member	Description
VIDEO_FORMAT_LINEAR	Linear video format
VIDEO_FORMAT_TILE	Tile video format. The tile cell size is calculated as follows: Tile cell size = 256 pixels x 16 lines
VIDEO_FORMAT_TILE64	Small tile video format. The tile cell size is calculated as follows: Tile cell size = 64 pixels x 16 lines

[Note]

None

COMPRESS_MODE_E

[Description]

Defines the video compression format.

[Syntax]

```
typedef enum hiCOMPRESS_MODE_E
{
    COMPRESS_MODE_NONE      = 0x0,      /*No compress */
    COMPRESS_MODE_SEG        = 0x1,      /*256 bytes are compressed as a
segment. It is the default segment mode.*/
    COMPRESS_MODE_SEG128     = 0x2,      /*128 bytes are compressed as a
segment.*/
    COMPRESS_MODE_LINE        = 0x3,      /*The whole line is compressed as
a segment.*/
    COMPRESS_MODE_FRAME       = 0x4,      /* compress unit is the whole
frame */
    COMPRESS_MODE_BUTT
} COMPRESS_MODE_E;
```

[Member]

Member	Description
COMPRESS_MODE_NONE	Non-compression video format
COMPRESS_MODE_SEG	Segment-based compression video format. Every 256 bytes are compressed as a segment.



Member	Description
COMPRESS_MODE_SEG128	Segment-based compression video format. Every 128 bytes are compressed as a segment.
COMPRESS_MODE_LINE	Line-based compression video format. Each line is compressed as a segment.
COMPRESS_MODE_FRAME	Frame compression video format. The video is compressed by frame.

[Note]

None

VB_SUPPLEMENT_CONF_S

[Description]

Defines the structure of the auxiliary VB information.

[Syntax]

```
typedef struct hiVB_SUPPLEMENT_CONF_S
{
    HI_U32 u32SupplementConf;
}VB_SUPPLEMENT_CONF_S;
```

[Member]

Member	Description
u32SupplementConf	Control of the auxiliary information

[Note]

Only **VB_SUPPLEMENT_JPEG_MASK** is supported currently. Other parameters are invalid.

[See Also]

- [HI_MPI_VB_GetSupplementConf](#)
- [HI_MPI_VB_SetSupplementConf](#)

2.5 Error Codes

2.5.1 Error Codes of System Control MPIS

[Table 2-3](#) describes the error codes of system control MPIS.



Table 2-3 Error codes of the system control MPIS

Error Code	Macro Definition	Description
0xA0028003	HI_ERR_SYS_ILLEGAL_PARAM	The parameter configuration is invalid.
0xA0028006	HI_ERR_SYS_NULL_PTR	The pointer is null.
0xA0028009	HI_ERR_SYS_NOT_PERM	The operation is forbidden.
0xA0028010	HI_ERR_SYS_NOTREADY	The system control attributes are not configured.
0xA0028012	HI_ERR_SYS_BUSY	The system is busy.
0xA002800C	HI_ERR_SYS_NOMEM	The memory fails to be allocated due to some causes such as insufficient system memory.

2.5.2 Error Codes of VB Pool MPIS

Table 2-4 describes the error codes of VB pool MPIS.

Table 2-4 Error codes of VB pool MPIS

Error Code	Macro Definition	Description
0xA0018003	HI_ERR_VB_ILLEGAL_PARAM	The parameter configuration is invalid.
0xA0018005	HI_ERR_VB_UNEXIST	The VB pool does not exist.
0xA0018006	HI_ERR_VB_NULL_PTR	The pointer is null.
0xA0018009	HI_ERR_VB_NOT_PERM	The operation is forbidden.
0xA001800C	HI_ERR_VB_NOMEM	The memory fails to be allocated.
0xA001800D	HI_ERR_VB_NOBUF	The buffer fails to be allocated.
0xA0018010	HI_ERR_VB_NOTREADY	The system control attributes are not configured.
0xA0018012	HI_ERR_VB_BUSY	The system is busy.
0xA0018040	HI_ERR_VB_2MPOOLS	Too many VB pools are created.



Contents

3 VI.....	3-1
3.1 Overview	3-1
3.2 Concepts	3-1
3.3 Functions	3-2
3.4 API Reference	3-5
3.5 Data Structures	3-92
3.6 Error Codes	3-153



Figures

Figure 3-1 Functional block diagram of the Hi3516A/Hi3518E V200/Hi3519 V100 VIU.....	3-2
Figure 3-2 Workflow of the Hi3516A VI channels	3-2
Figure 3-3 Workflow of the Hi3518E V200 VI channels	3-2
Figure 3-4 Workflow of the Hi3519 V100 VI channels	3-3
Figure 3-5 DIS usage process of Hi3519 V100.....	3-4



Tables

Table 3-1 Mask configuration in the 1-channel 5-megapixel or 1080p scenario (12 bits).....	3-4
Table 3-2 Mask configuration in the 1-channel BT.1120 input scenario (16 bits).....	3-4
Table 3-3 Mask configuration in the 1xD1 input scenario (8 bits)	3-4
Table 3-4 Error codes of VI APIs	3-153



3 VI

3.1 Overview

The video input unit (VIU) receives video data from the ITU-R BT.656/BT.601/BT.1120 interface, digital camera interface, or mobile industry processor interface (MIPI) receive (RX) interfaces (including the MIPI interface, LVDS interface, and HiSPi interface). When the VIU works in offline mode, the received data is stored in the specified memory. When the VIU works in online mode, the VIU directly sends the received data to the video processing subsystem (VPSS). During this process, the VIU can crop the source picture. In this way, 1-channel picture is output when there is 1-channel picture input.

3.2 Concepts

Note the following:

- Video input (VI) device
 - A VI device supports multiple input timings and parses them.
- Online and offline modes
 - Offline mode: The VIU writes data to the DDR, and the module bound to the VIU reads data from the DDR.
 - Online mode: The VIU and VPSS transfer data online. In this mode, the VIU transmits data to the VPSS instead of writing data to the DDR.
- Physical VI channel
 - A physical VI channel transfers the data obtained after a VI device parses timings to the DDR. Before the data arrives in the DDR, pictures can be cropped. For details, see the related data sheet.
- Mask
 - Mask indicates the video data source of a VI device.
- Lens distortion correction (LDC)
 - The pictures captured by some low-end lenses are easily distorted. This function is used to correct distorted pictures.
- Dynamic contrast improvement (DCI)
 - This function is used to dynamically adjust the picture contrast. In this way, the bright region will not be too bright when the luminance of the dark region is increased, and the dark region will not be too dark when the luminance of the bright region is decreased.

- Digital image stabilization (DIS)

The DIS module compares the current picture with the previous two pictures, uses multiple degrees of freedom (DOF) DIS algorithms to calculate the jitter offset vectors of the current picture on each axis, and then corrects the current picture based on the jitter offset vectors. In this way, the anti-jitter effect is achieved.

- Image pixel format

In this document, sp422 indicates semi-planar4:2:2, sp420 indicates semi-planar4:2:0, YUV400 indicates that only the luminance component is used.

3.3 Functions

Functional Block Diagram

Figure 3-1 shows the functional block diagram of the Hi3516A/Hi3518E V200/Hi3519 V100 VIU.

Figure 3-1 Functional block diagram of the Hi3516A/Hi3518E V200/Hi3519 V100 VIU

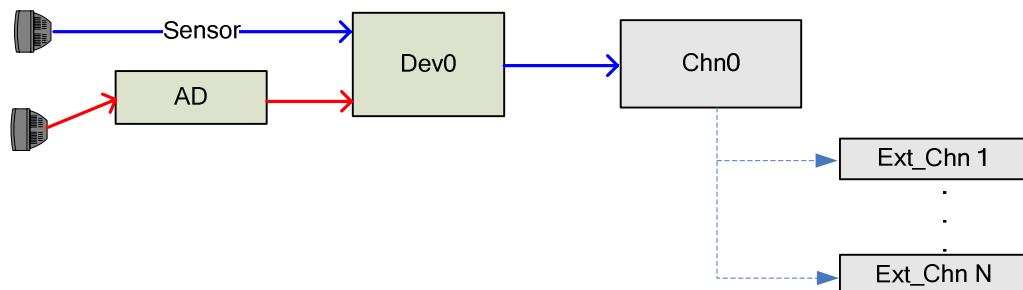


Figure 3-2 Workflow of the Hi3516A VI channels

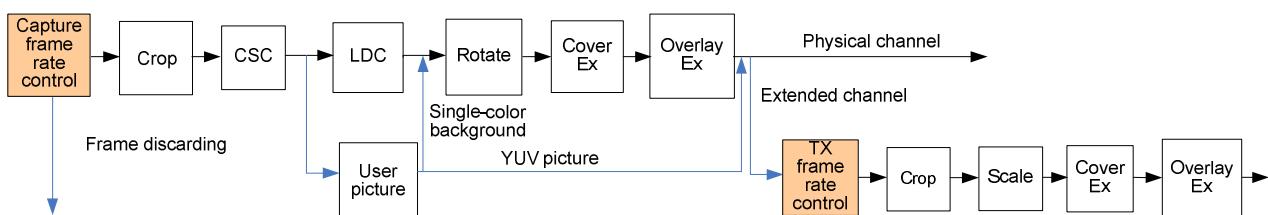


Figure 3-3 Workflow of the Hi3518E V200 VI channels

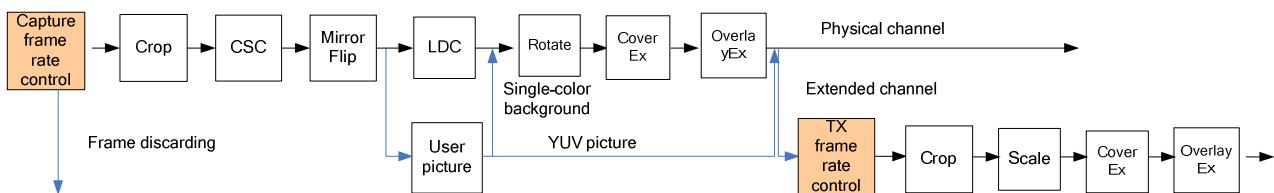
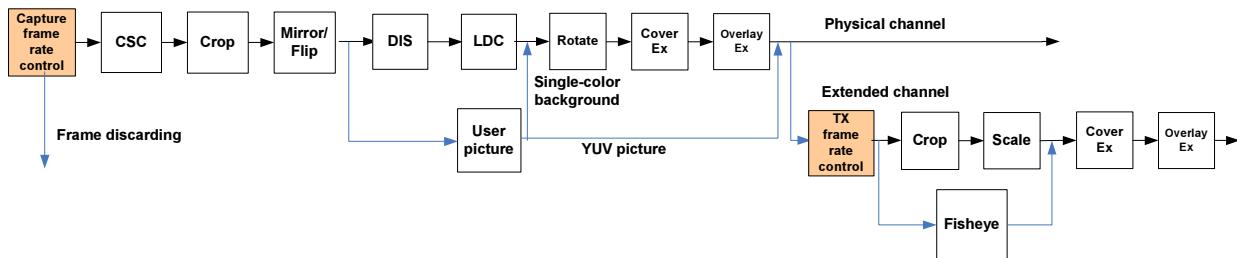


Figure 3-4 Workflow of the Hi3519 V100 VI channels

VI Device

The Hi3516A/Hi3518E V200/Hi3519 V100 supports only one VI device Dev0. Dev0 supports the inputs of the BT.656, BT.601, DC, and MIPI RX interfaces (MIPI, LVDS, and HiSPi). Dev0 does not support inconsecutive pixel clock timings.

Physical VI channel

The Hi3516A/Hi3518E V200/Hi3519 V100 VIU provides only one physical VI channel Chn0. This is no secondary channel for the Hi3516A/Hi3518E V200/Hi3519 V100, but the Hi3516A/Hi3518E V200/Hi3519 V100 supports extended channels.

The typical resolutions supported by Chn0 of the Hi3516A include 720p@30 fps, 1080p@30 fps, 1080p@60 fps, 2048 x 1536@30 fps, and 2592 x 1944@30 fps.

The typical resolutions supported by Chn0 of Hi3518E V200 include 720p@30 fps and 1080p@30 fps.

The typical resolutions supported by Chn0 of Hi3519 V100 include 1080p@120 fps, 2048 x 1536@60 fps, 2592 x 1944@60 fps, 3840 x 2160@30 fps, and 4608 x 3456@15 fps.

Extended VI Channels

Extended VI channels are used to scale pictures, and their data source is the physical VI channel. The Hi3516A/Hi3518E V200/Hi3519 V100 supports a maximum of 16 extended VI channels.

Binding Relationship

The binding relationship between the Hi3516A/Hi3518E V200/Hi3519 V100 physical VI channel and the device is fixed. The binding relationship cannot be changed.

Mask Configuration

The upper 12 bits of the mask correspond to the 12 pins (any consecutive 12 pins from D0 to D15, for example, D4–D15) on the hardware circuit. Configure appropriate mask based on the actual connection. The most significant bit (MSB) of the mask corresponds to D15 pin. For example, if the pins (D6–D15) connect to the 10-bit sensor, set the mask to **0xFFC00000**. If the pins (D6–D15) connect to the 14-bit sensor, set the mask to **0xFFFFC00000**.

- 1-channel 5-megapixel or 1080p input scenario (12-bit inputs)



In this scenario, configured masks by following [Table 3-1](#) when configuring VI device attributes.

Table 3-1 Mask configuration in the 1-channel 5-megapixel or 1080p scenario (12 bits)

Device ID	Mask 0	Mask 1
0	0xFFFF00000	0x0

- 1-channel BT.1120 HD input scenario (16-bit inputs)

In this scenario, configure masks by following [Table 3-2](#) when configuring VI device attributes.

Table 3-2 Mask configuration in the 1-channel BT.1120 input scenario (16 bits)

Device ID	Mask 0	Mask 1
0	0xFF000000	0x00FF0000

- 1xD1 input scenario (8-bit inputs)

In the 1xD1 input scenario, configure the masks by following [Table 3-3](#) when configuring VI device attributes.

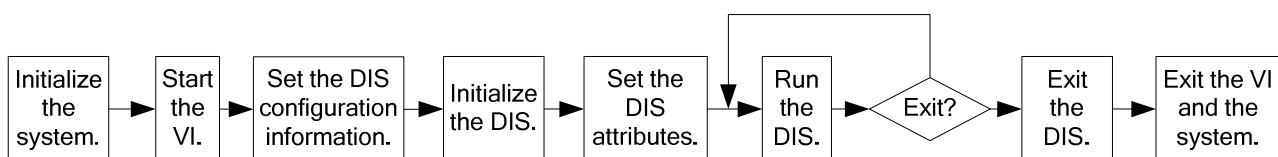
Table 3-3 Mask configuration in the 1xD1 input scenario (8 bits)

Device ID	Mask 0	Mask 1
0	0xFF000000	0x0

DIS Software Process

The VI channel must be enabled before the DIS MPIs are called. [Figure 3-5](#) shows the DIS usage process.

Figure 3-5 DIS usage process of Hi3519 V100





CAUTION

- The DIS supports only the offline mode and runs only on the physical channel.
- Currently the typical resolutions supported by the DIS include 3840 x 2160@30, 1080p@60, and 720p@120. Whether other resolutions (such as 3264 x 2448 and 4608 x 3456) are supported is not verified.
- The DIS input video pictures can only be in the semi-planar 420 or single-component format, and only uncompressed pictures are supported.
- The big-little environment of Hi3519 V100 needs to be used to guarantee the DIS performance. If the multi-service scenario is used, you are advised to increase the frequency of the A17 CPU to 1.15 GHz and bind the DIS thread to A17 for running.
- The VGS/fisheye module is required during DIS processing. If multiple modules call the VGS and fisheye functions, frame loss may occur during DIS processing due to insufficient VGS/fisheye performance.

3.4 API Reference

The VIU enables VI devices and VI channels and binds VI channels. The VIU provides the following MPP programming interfaces (MPIs):

- [HI_MPI_VI_SetDevAttr](#): Sets the attributes of a VI device.
- [HI_MPI_VI_GetDevAttr](#): Obtains the attributes of a VI device.
- [HI_MPI_VI_SetDevAttrEx](#): Sets the attributes of a VI device (this advanced MPI is called for special timings).
- [HI_MPI_VI_GetDevAttrEx](#): obtains the attributes of a VI device.
- [HI_MPI_VI_EnableDev](#): Enables a VI device.
- [HI_MPI_VI_DisableDev](#): Disables a VI device.
- [HI_MPI_VI_SetChnAttr](#): Sets the attribute of a VI channel.
- [HI_MPI_VI_GetChnAttr](#): Obtains the attribute of a VI channel.
- [HI_MPI_VI_EnableChn](#): Enables a VI channel.
- [HI_MPI_VI_DisableChn](#): Disables a VI channel.
- [HI_MPI_VI_EnableChnInterrupt](#): Enables a VI channel to respond to the hardware interrupt handler.
- [HI_MPI_VI_DisableChnInterrupt](#): Forbids a VI channel to respond to the hardware interrupt handler.
- [HI_MPI_VI_SetFrameDepth](#): Sets the maximum depth of the buffer for storing the obtained VI pictures.
- [HI_MPI_VI_GetFrameDepth](#): Obtains the maximum depth of the buffer for storing the obtained VI pictures.
- [HI_MPI_VI_GetFrame](#): Obtains VI pictures.
- [HI_MPI_VI_ReleaseFrame](#): Releases the buffer occupied by VI pictures.
- [HI_MPI_VI_SetUserPic](#): Sets a user picture that is used as the picture inserted when there is no video signal.
- [HI_MPI_VI_EnableUserPic](#): Enables the function of inserting a user picture.



- [HI_MPI_VI_DisableUserPic](#): Disables the function of inserting a user picture.
- [HI_MPI_VI_BindChn](#): Binds a VI channel.
- [HI_MPI_VI_UnBindChn](#): Unbinds a VI channel.
- [HI_MPI_VI_GetChnBind](#): Obtains the binding relationship of a VI channel.
- [HI_MPI_VI_GetFd](#): Obtains the device file descriptor (FD) corresponding to a VI channel.
- [HI_MPI_VI_CloseFd](#): Closes the FD of the device and channel.
- [HI_MPI_VI_Query](#): Queries the information about a VI channel such as the interrupt count and average frame rate.
- [HI_MPI_VI_SetFlashConfig](#): Sets the camera flash.
- [HI_MPI_VI_GetFlashConfig](#): Obtains camera flash settings.
- [HI_MPI_VI_TriggerFlash](#): Enables or disables the camera flash.
- [HI_MPI_VI_SetExtChnAttr](#): Sets the attributes of an extended VI channel.
- [HI_MPI_VI_GetExtChnAttr](#): Obtains the attributes of an extended VI channel.
- [HI_MPI_VI_SetExtChnCrop](#): Sets the cropping attributes of an extended VI channel.
- [HI_MPI_VI_GetExtChnCrop](#): Obtains the cropping attributes of an extended VI channel.
- [HI_MPI_VI_SetLDCAttr](#): Sets the LDC attribute.
- [HI_MPI_VI_GetLDCAttr](#): Obtains the LDC attribute.
- [HI_MPI_VI_SetCSCAttr](#): Sets the color space conversion (CSC) attribute of a VI device.
- [HI_MPI_VI_GetCSCAttr](#): Obtains the CSC attribute of a VI device.
- [HI_MPI_VI_SetRotate](#): Sets the rotation attribute of VI pictures.
- [HI_MPI_VI_GetRotate](#): Obtains the rotation attribute of VI pictures.
- [HI_MPI_VI_GetChnLuma](#): Obtains the channel luminance statistics.
- [HI_MPI_VI_SetFisheyeDevConfig](#): Sets the LMF parameters of the fisheye lens corresponding to the VI device.
- [HI_MPI_VI_GetFisheyeDevConfig](#): Obtains the LMF parameters of the fisheye lens corresponding to the VI device.
- [HI_MPI_VI_SetFisheyeAttr](#): Sets the fisheye attribute corresponding to an extended VI channel.
- [HI_MPI_VI_GetFisheyeAttr](#): Obtains the fisheye attribute corresponding to an extended VI channel.
- [HI_MPI_VI_SetDevDumpAttr](#): Sets the dump attribute of the VI device.
- [HI_MPI_VI_GetDevDumpAttr](#): Obtains the dump attribute of the VI device.
- [HI_MPI_VI_EnableBayerDump](#): Enables the function of obtaining raw data.
- [HI_MPI_VI_DisableBayerDump](#): Disables the function of obtaining raw data.
- [HI_MPI_VI_EnableBayerRead](#): Enables the function of reading raw data.
- [HI_MPI_VI_SendBayerData](#): Sends raw data to the ISP.
- [HI_MPI_VI_SendBayerDataEx](#): Send multi-channel raw data to the ISP. This MPI is an extended MPI of [HI_MPI_VI_SendBayerData](#).
- [HI_MPI_VI_DisableBayerRead](#): Disables the function of reading raw data.
- [HI_MPI_VI_SetDCIParam](#): Configures dynamic contrast improvement (DCI) parameters.
- [HI_MPI_VI_GetDCIParam](#): Obtains DCI parameters.
- [HI_MPI_VI_SetWDRAttr](#): Configures wide dynamic range (WDR) attributes.



- [HI_MPI_VI_GetWDRAttr](#): Obtains WDR attributes.
- [HI_MPI_VI_SetModParam](#): Configures the parameters of the VI module.
- [HI_MPI_VI_GetModParam](#): Obtains the parameters of the VI module.
- [HI_MPI_VI_SetDISConfig](#): Sets the DIS configuration information.
- [HI_MPI_VI_GetDISConfig](#): Obtains the DIS configuration information.
- [HI_MPI_VI_DISInit](#): Initializes the DIS.
- [HI_MPI_VI_SetDISAttr](#): Sets the DIS attributes.
- [HI_MPI_VI_GetDISAttr](#): Obtains the DIS attributes.
- [HI_MPI_VI_DISRun](#): Runs the DIS.
- [HI_MPI_VI_DISExit](#): Exits the DIS.
- [HI_MPI_VI_RegisterDISCallback](#): Registers the callback functions for obtaining and releasing the gyroscope data with the DIS.
- [HI_MPI_VI_UnRegisterDISCallback](#): Deregisters the callback functions for obtaining and releasing the gyroscope data from the DIS.

HI_MPI_VI_SetDevAttr

[Description]

Sets the attributes of a VI device. As basic device attributes support some chip configurations by default, the interconnection requirements of most sensors can be met. For some special timings, the advanced MPI [HI_MPI_VI_SetDevAttrEx](#) can be called to set all attributes of a VI device.

[Syntax]

```
HI_S32 HI_MPI_VI_SetDevAttr(VI_DEV ViDev, const VI_DEV_ATTR_S *pstDevAttr)
```

[Parameter]

Parameter	Description	Input/Output
ViDev	VI device ID Value range: [0, VIU_MAX_DEV_NUM)	Input
pstDevAttr	Pointer to the attributes of a VI device (static attribute)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

None



[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

- The binding relationship between the device and the channel is not allowed to be changed. Only 1-channel multiplexing working mode is supported. The BT.1120 interlaced input is not supported.
- Before calling this MPI, ensure that the VI device is disabled. If the VI device is enabled, you can disable it by calling [HI_MPI_VI_DisableDev](#).
- The **pstDevAttr** parameter is used to configure the video interface mode of a specified VI device to connect it to an external camera, sensor, or codec. The supported interface modes include BT.656 mode, BT.601 mode, DC mode, MIPI Rx (MIPI/LVDS/HiSPi), BT.1120 progressive mode, and BT.1120 interlaced mode. You need to configure the following information. For details, see section [3.5 "Data Structures"](#).
 - Interface mode information: BT.656 mode, BT.601 mode, DC mode, MIPI Rx(MIPI/LVDS/HiSPi), or BT.1120 mode
 - Working mode information: 1-, 2-, or 4-channel multiplexing mode
 - Data layout information: Arrangement of multi-channel data in multiplexing mode
 - Data information: interlaced or progressive input and YUV data input sequence
 - Sync timing information: attributes of the vertical or horizontal sync signal

[Example]

```
HI_S32 s32Ret;
VI_DEV ViDev = 0;
VI_CHN ViChn = 0;
VI_DEV_ATTR_S stDevAttr;
VI_CHN_ATTR_S stChnAttr;
stDevAttr.enIntfMode = VI_MODE_DIGITAL_CAMERA;
stDevAttr.enWorkMode = VI_WORK_MODE_1Multiplex;
stDevAttr.au32CompMask[0] = 0xFFFF0000;
stDevAttr.au32CompMask[1] = 0x0;
stDevAttr.enScanMode = VI_SCAN_PROGRESSIVE;
stDevAttr.s32AdChnId[0] = -1;
stDevAttr.s32AdChnId[1] = -1;
stDevAttr.s32AdChnId[2] = -1;
stDevAttr.s32AdChnId[3] = -1;
stDevAttr.stDevRect.s32X = 0;
stDevAttr.stDevRect.s32Y = 0;
stDevAttr.stDevRect.u32Width = 1920;
stDevAttr.stDevRect.u32Height = 1080;
s32Ret = HI_MPI_VI_SetDevAttr(ViDev, &stDevAttr);
if (s32Ret != HI_SUCCESS)
{
    printf("Set dev attributes failed with error code %#x!\n", s32Ret);
```



```
        return HI_FAILURE;
    }

s32Ret = HI_MPI_VI_EnableDev(ViDev);
if (s32Ret != HI_SUCCESS)
{
    printf("Enable dev failed with error code %#x!\n", s32Ret);
    return HI_FAILURE;
}

stChnAttr.stCapRest.s32X = 0;
stChnAttr.stCapRect.s32Y = 0;
stChnAttr.stCapRect.u32Width = 1920;
stChnAttr.stCapRect.u32Height = 1080;
stChnAttr.stDestSize.u32Width = 1920;
stChnAttr.stDestSize.u32Height = 1080;
stChnAttr.enCapSel = VI_CAPSEL_BOTH;
stChnAttr.enPixelFormat = PIXEL_FORMAT_YUV_SEMIPLANAR_420;
stChnAttr.bMirror = HI_FALSE;
stChnAttr.bFlip = HI_FALSE;
stChnAttr.s32SrcFrameRate = -1;
stChnAttr.s32DstFrameRate = -1;
stChnAttr.enCompressMode = COMPRESS_MODE_NONE;
s32Ret = HI_MPI_VI_SetChnAttr(ViChn, &stChnAttr);
if (s32Ret != HI_SUCCESS)
{
    printf("Set chn attributes failed with error code %#x!\n", s32Ret);
    return HI_FAILURE;
}

s32Ret = HI_MPI_VI_EnableChn(ViChn);
if (s32Ret != HI_SUCCESS)
{
    printf("Enable chn failed with error code %#x!\n", s32Ret);
    return HI_FAILURE;
}

/* now, vi is capturing images, you can do something else*/

s32Ret = HI_MPI_VI_DisableChn(ViChn);
if (s32Ret != HI_SUCCESS)
{
    printf("Disable chn failed with error code %#x!\n", s32Ret);
    return HI_FAILURE;
}

s32Ret = HI_MPI_VI_DisableDev(ViDev);
if (s32Ret != HI_SUCCESS)
{
```



```
    printf("Disable dev failed with error code %#x!\n", s32Ret);
    return HI_FAILURE;
}
```

[See Also]

- [HI_MPI_VI_GetDevAttr](#)
- [HI_MPI_VI_SetDevAttrEx](#)

HI_MPI_VI_GetDevAttr

[Description]

Obtains the attributes of a VI device.

[Syntax]

```
HI_S32 HI_MPI_VI_GetDevAttr(VI_DEV ViDev, VI_DEV_ATTR_S *pstDevAttr)
```

[Parameter]

Parameter	Description	Input/Output
ViDev	VI device ID Value range: [0, VIU_MAX_DEV_NUM)	Input
pstDevAttr	Pointer to the attributes of a VI device	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpia.a

[Note]

If the attributes of a VI device are not set, a code indicating failure is returned after this MPI is called.

[Example]

None



[See Also]

[HI_MPI_VI_SetDevAttr](#)

HI_MPI_VI_SetDevAttrEx

[Description]

Sets the attributes of a VI device. This MPI is called to set all attributes of a VI device only when [HI_MPI_VI_SetDevAttr](#) cannot meet the requirements of some special timings.

[Syntax]

```
HI_S32 HI_MPI_VI_SetDevAttrEx(VI_DEV ViDev, const VI_DEV_ATTR_EX_S
*pstDevAttrEx)
```

[Parameter]

Parameter	Description	Input/Output
ViDev	VI device ID Value range: [0, VIU_MAX_DEV_NUM)	Input
pstDevAttrEx	Pointer to the advanced attributes of a VI device (static attribute)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

- The binding relationship between the device and the channel is not allowed to be changed. Only 1-channel multiplexing working mode is supported.
- Before calling this MPI, ensure that the VI device is disabled. If the VI device is enabled, you can disable it by calling [HI_MPI_VI_DisableDev](#).
- The **pstDevAttrEx** parameter is used to configure the video interface mode of a specified VI device to connect it to an external camera, sensor, or codec. The supported interface modes include BT.656 mode, BT.601 mode, DC mode, MIPI RX (MIPI/LVDS/HiSPi), BT.1120 progressive mode, and BT.1120 interlaced mode. You



need to configure the following information. For details, see section [3.5 "Data Structures."](#)

- Interface mode information: BT.656 mode, BT.601 mode, DC mode, MIPI Rx(MIPI/LVDS/HiSPi), or interleave mode
- Working mode information: 1-, 2-, or 4-channel multiplexing mode
- Data layout information: Arrangement of multi-channel data in multiplexing mode
- Data information: interlaced or progressive input, YUV data input sequence, composite or separation mode, and component mode
- Clock information: sampling on the rising edge or falling edge
- Sync timing information: attributes of the vertical or horizontal sync signal

[Example]

None

[See Also]

[HI_MPI_VI_GetDevAttrEx](#)

[HI_MPI_VI_GetDevAttrEx](#)

[Description]

Obtains the attributes of a VI device.

[Syntax]

```
HI_S32 HI_MPI_VI_GetDevAttrEx(VI_DEV ViDev, VI_DEV_ATTR_EX_S  
*pstDevAttrEx)
```

[Parameter]

Parameter	Description	Input/Output
ViDev	VI device ID Value range: [0, VIU_MAX_DEV_NUM)	Input
pstDevAttrEx	Pointer to the attributes of a VI device	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

None

[Requirement]



- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

If the attributes of a VI device are not set, a code indicating failure is returned after this MPI is called.

[Example]

None

[See Also]

[HI_MPI_VI_SetDevAttrEx](#)

HI_MPI_VI_EnableDev

[Description]

Enables a VI device.

[Syntax]

```
HI_S32 HI_MPI_VI_EnableDev(VI_DEV ViDev);
```

[Parameter]

Parameter	Description	Input/Output
ViDev	VI device ID Value range: [0, VIU_MAX_DEV_NUM)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

- The binding relationship between the device and the channel is not allowed to be changed.



- Ensure that the attributes of a VI device are set before you enable it. Otherwise, a code indicating failure is returned.
- The VI device can be enabled repeatedly without any failure.

[Example]

See the example of [HI_MPI_VI_SetDevAttr](#).

[See Also]

[HI_MPI_VI_DisableDev](#)

HI_MPI_VI_DisableDev

[Description]

Disables a VI device.

[Syntax]

```
HI_S32 HI_MPI_VI_DisableDev(VI_DEV ViDev);
```

[Parameter]

Parameter	Description	Input/Output
ViDev	VI device ID Value range: [0, VIU_MAX_DEV_NUM)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

- A VI device can be disabled only after all VI channels bound to the device are disabled.
- The VI device can be disabled repeatedly without any failure.
- The SDK supports the low-power mode. A VI device is closed after being disabled. You must set the attributes of the VI device before enabling it.

[Example]



See the example of [HI_MPI_VI_SetDevAttr](#).

[See Also]

[HI_MPI_VI_EnableDev](#)

HI_MPI_VI_SetChnAttr

[Description]

Sets the attribute of a VI channel.

[Syntax]

```
HI_S32 HI_MPI_VI_SetChnAttr(VI_CHN ViChn, const VI_CHN_ATTR_S *pstAttr);
```

[Parameter]

Parameter	Description	Input/Output
ViChn	VI channel ID Value range: [0, VIU_MAX_PHYCHN_NUM)	Input
pstAttr	Pointer to the attribute of a VI channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

The limitations on each channel attribute item are as follows:

- Capture region (**stCapRect**)
 - **stCapRect** is used to crop the source picture. The width and height of the capture region (**stCapRect**) are static attributes, and other attributes are dynamic attributes.
 - **stCapStart** is used to configure the position of the rectangle to be captured relative to the start point of the source picture. For the start point, the horizontal coordinate is in the unit of pixel and the vertical coordinate is in the unit of line.
 - **s32X** and **u32Width** must be 2-pixel-aligned. **s32Y** and **u32Height** must be 2-pixel-aligned in progressive capture mode, and be 4-pixel-aligned in interlaced capture



mode. **s32X** and **s32Y** cannot be less than 0. The sum of **s32X** and **u32Width** cannot be greater than the width of **stDevRect** in the device attribute, and the sum of **s32Y** and **u32Height** cannot be greater than the height of **stDevRect** in the device attribute.

- When fixed pattern noise (FPN) calibration or correction is enabled for Hi3518E V200, **stCapRect** must be the same as **stDevRect** in the device attribute.
- Target picture size (**stDestSize**)
 - **stDestSize** must be set and its value must be within the region of the picture output by the external ADC. Otherwise, the VI device cannot work properly.
 - In progressive input mode, the width and height of **stDestSize** must be equal to those of **stCapRect**.
 - When the input picture is in interlaced mode, the capture height is set to the height of a field or two fields based on the field mode and the capture height must be a multiple of 4 in two-field capture mode.
- Field capture selection (**enCapSel**)
 - **enCapSel** is used to capture only one field when the source pictures are input in interlaced mode.
 - You are advised to capture the bottom field in single-field mode to prevent picture flicker.
- Pixel format (**enPixelFormat**): During rotation processing, only the sp420 and YUV400 output pixel formats are supported.
PIXEL_FORMAT_RGB_BAYER indicates that the channel outputs 16-bit raw data. **PIXEL_FORMAT_RGB_BAYER** is not supported when LDC or rotation is enabled.
- Source frame rate (**s32SrcFrameRate**): If you do not want to control the frame rate, set **s32SrcFrameRate** to **-1**.
- Target frame rate (**s32DstFrameRate**): If you do not want to control the frame rate, set **s32FrameRate** to **-1**. If you want to control the frame rate, you must set the source frame rate and target frame rate in sequence, and the target frame rate must be less than or equal to the source frame rate.
- You are advised not to control the frame rate in the snapshot camera solution. Otherwise, flash signal triggering is delayed, and snapshot cannot be responded in a timely manner.
- During LDC processing, only the uncompressed input and output are supported. During fisheye processing and DIS processing, only the uncompressed input is supported.
- Interlaced pictures do not support mirroring and flipping. If the function is enabled, the sequence of capturing top and bottom fields is reversed, which may affect the picture quality when pictures are processed by other modules. As the MPP does not check whether the function is enabled, you need to ensure that the function is disabled in interlaced mode.
- The channel attributes can be configured only after the device attributes are configured; otherwise, an error code indicating failure is returned.

[Example]

See the example of [HI_MPI_VI_SetDevAttr](#).

[See Also]

[HI_MPI_VI_GetChnAttr](#)

HI_MPI_VI_GetChnAttr

[Description]



Obtains the attribute of a VI channel.

[Syntax]

```
HI_S32 HI_MPI_VI_GetChnAttr(VI_CHN ViChn, VI_CHN_ATTR_S *pstAttr);
```

[Parameter]

Parameter	Description	Input/Output
ViChn	VI channel ID Value range: [0, VIU_MAX_PHYCHN_NUM)	Input
pstAttr	Pointer to the attribute of a VI channel (dynamic attribute)	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

You must set the attributes of a VI channel before calling this MPI. Otherwise, a code indicating failure is returned.

[Example]

```
HI_S32 s32ret;
VI_CHN ViChn = 0;
VI_CHN_ATTR_S stChnAttr;
/* first enable vi device and vi chn */

/* get channel attribute for vi chn */
s32ret = HI_MPI_VI_GetChnAttr(ViChn, &stChnAttr);
if (HI_SUCCESS != s32ret)
{
    printf("get vi chn attr err:0x%x\n", s32ret);
    return s32ret;
```



}

[See Also]

[HI_MPI_VI_SetChnAttr](#)

HI_MPI_VI_EnableChn

[Description]

Enables a VI channel.

[Syntax]

```
HI_S32 HI_MPI_VI_EnableChn(VI_CHN ViChn);
```

[Parameter]

Parameter	Description	Input/Output
ViChn	VI channel ID Value range: [0, VIU_MAX_NUM)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

- You must set the attributes of a VI channel and enable the VI device bound to the VI channel before enabling the VI channel.
- If the back-end module is bound to the VI, the back-end module obtains the video data after this MPI is successfully called.
- A VI channel can be enabled repeatedly without any failure.

[Example]

See the example of [HI_MPI_VI_SetDevAttr](#).

[See Also]



[HI_MPI_VI_DisableChn](#)

[Description]

Disables a VI channel.

[Syntax]

```
HI_S32 HI_MPI_VI_DisableChn(VI_CHN ViChn);
```

[Parameter]

Parameter	Description	Input/Output
ViChn	VI channel ID Value range: [0, VIU_MAX_CHN_NUM)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

- A VI channel can be disabled only after all the extended channels bound to it are disabled; otherwise, an error code indicating failure is returned.
- After a VI channel is disabled, it stops capturing VI data. If the VI channel is bound to a VO channel or a VENC channel, the VO channel or VENC channel does not receive pictures.
- A VI channel can be disabled repeatedly without any failure.

[Example]

See the example of [HI_MPI_VI_SetDevAttr](#).

[See Also]

[HI_MPI_VI_EnableChn](#)



HI_MPI_VI_EnableChnInterrupt

[Description]

Enables a VI channel to respond to the hardware interrupt handler.

[Syntax]

```
HI_S32 HI_MPI_VI_EnableChnInterrupt(VI_CHN ViChn);
```

[Parameter]

Parameter	Description	Input/Output
ViChn	VI channel ID Value range: [0, VIU_MAX_PHYCHN_NUM)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

- HI_MPI_VI_EnableChnInterrupt is an advanced interface, and it is not called typically.
- When an application detects exceptions in front-end ADC timings, [HI_MPI_VI_DisableChnInterrupt](#) is called to forbid the VI channel to respond to the hardware interrupt handler while the function of inserting user pictures is enabled. This reduces the system resources that are consumed when hardware reports a large amount of interrupts due to timing exceptions. When it is detected that the front-end ADC timings are normal, call HI_MPI_VI_EnableChnInterrupt to enable the VI channel to respond to the hardware interrupt handler again.
- Enable a VI channel before calling HI_MPI_VI_EnableChnInterrupt. Otherwise, an error is returned.
- HI_MPI_VI_EnableChnInterrupt can be called repeatedly, and no code indicating failure is returned.

[Example]

None



[See Also]

[HI_MPI_VI_DisableChnInterrupt](#)

HI_MPI_VI_DisableChnInterrupt

[Description]

Forbids a VI channel to respond to the hardware interrupt handler.

[Syntax]

```
HI_S32 HI_MPI_VI_DisableChnInterrupt(VI_CHN ViChn);
```

[Parameter]

Parameter	Description	Input/Output
ViChn	VI channel ID Value range: [0, VIU_MAX_PHYCHN_NUM)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

- `HI_MPI_VI_DisableChnInterrupt` is an advanced interface, and it is not called typically.
- When an application detects exceptions in front-end ADC timings, `HI_MPI_VI_DisableChnInterrupt` is called to forbid the VI channel to respond to the hardware interrupt handler while the function of inserting user pictures is enabled. This reduces the system resources that are consumed when hardware reports a large amount of interrupts due to timing exceptions. When it is detected that the front-end ADC timings are normal, call [HI_MPI_VI_EnableChnInterrupt](#) to enable the VI channel to respond to the hardware interrupt handler again.
- If no VI channel is enabled, you do not need to forbid the VI channel to respond to the hardware interrupt handler. In this case, a code indicating success is returned if you call `HI_MPI_VI_DisableChnInterrupt`.
- `HI_MPI_VI_DisableChnInterrupt` can be called repeatedly, and no code indicating failure is returned.



- The memory needs to be released when this MPI is called. Therefore, there is 120-ms delay in returning.

[Example]

None

[See Also]

[HI_MPI_VI_EnableChnInterrupt](#)

HI_MPI_VI_SetFrameDepth

[Description]

Sets the maximum depth of the buffer for storing the obtained VI pictures.

[Syntax]

```
HI_S32 HI_MPI_VI_SetFrameDepth(VI_CHN ViChn, HI_U32 u32Depth);
```

[Parameter]

Parameter	Description	Input/Output
ViChn	VI channel ID Value range: [0, VIU_MAX_PHYCHN_NUM)	Input
u32Depth	Maximum depth of the VI picture to be obtained The default value is 0. Value range: [0, 8]	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

- Only the offline mode is supported.



- This MPI is used to set the number of video frames buffered in a VI channel. If the number of buffered video frames is not set to 0, consecutive video frames can be obtained.
- If u32Depth is set to 0, the system does not buffer pictures for the VI channel. Therefore, you cannot obtain pictures from this VI channel. By default, the system does not buffer pictures for the VI channel. That is, the default value of u32Depth is 0.
- If u32Depth is set to an integer greater than 0, the system buffers u32Depth pictures for the VI channel. You can obtain the pictures by calling [HI_MPI_VI_GetFrame](#). Note the following cases:
 - Pictures are not fetched.
The system automatically replaces old pictures with latest pictures. This ensures that users can obtain the latest u32Depth pictures.
 - Pictures are fetched for u32Depth consecutive times and the video buffer (VB) is not released.
The system automatically stops buffering new VI picture because the system cannot obtain the VB. As a result, users cannot obtain new VI pictures. You are advised to call the release MPI after you call the obtain MPI.
 - The speed of obtaining and releasing pictures is lower than the speed of generating pictures in the VI channel.
The system automatically updates the old pictures that are not fetched. This ensures that the buffered pictures are the latest ones. As the speed of obtaining pictures cannot be ensured, the obtained pictures may be inconsecutive.
- When the system buffers u32Depth pictures for each VI channel, the system occupies the VBs in the MPP. You need to set sufficient VBs by calling [HI_MPI_VB_SetConf](#). Otherwise, the VIU cannot capture pictures properly because the system may occupy too many VBs for buffering pictures. As a result, users cannot obtain VI pictures. You are advised to set u32Depth dynamically. If you do not need to obtain pictures from VI channels, you can set u32Depth to 0 to reduce the number of VBs occupied by VI channels. When you want to obtain pictures, you can set u32Depth. Then you can obtain consecutive VI pictures generated after the time when you set u32Depth.

[Example]

See the example of [HI_MPI_VI_GetFrame](#).

[See Also]

[HI_MPI_VI_GetFrame](#)

HI_MPI_VI_GetFrameDepth

[Description]

Obtains the maximum depth of the buffer for storing the obtained VI pictures.

[Syntax]

```
HI_S32 HI_MPI_VI_GetFrameDepth(VI_CHN ViChn, HI_U32 *pu32Depth);
```

[Parameter]

Parameter	Description	Input/Output
ViChn	VI channel ID	Input



Parameter	Description	Input/Output
	Value range: [0, VIU_MAX_PHYCHN_NUM)	
pu32Depth	U32 pointer	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

Only the offline mode is supported.

[Example]

None

[See Also]

None

HI_MPI_VI_GetFrame

[Description]

Obtains pictures captured by a VI channel.

[Syntax]

```
HI_S32 HI_MPI_VI_GetFrame(VI_CHN ViChn, VIDEO_FRAME_INFO_S *pstFrameInfo,  
HI_S32 s32MilliSec);
```

[Parameter]

Parameter	Description	Input/Output
ViChn	VI channel ID Value range: [0, VIU_MAX_CHN_NUM)	Input
pstFrameInfo	Pointer to the structure of the VI frame information	Output



Parameter	Description	Input/Output
s32MilliSec	Timeout period	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

- Only the offline mode is supported.
- This MPI is used to obtain the information about pictures in a specified VI channel. The picture information includes the picture width, height, pixel format, PTS, and physical addresses of YUV components.
- HI_MPI_VI_GetFrame is available only after a channel is enabled.
- The occupied buffer can be released after HI_MPI_VI_GetFrame is called for multiple times. You are advised to call the release MPI after you call the obtain MPI.
- The physical address information is obtained from the VB used by the MPP. Therefore, the VB must be released after use by calling [HI_MPI_VI_ReleaseFrame](#).
- pstFrameInfo -> stVFrame .u32PhyAddr[0]** points to the physical address of the Y component and **pstFrameInfo -> stVFrame .u32PhyAddr[1]** points to the physical address of the C component.
- The default timeout period for this MPI is 2s. That is, this MPI is returned if no pictures are obtained in 2s.
- If **u32MilliSec** is set to **-1**, the block mode is used. In this case, a code is returned only after pictures are obtained. If **u32MilliSec** is greater than **0**, the non-block mode is used. **u32MilliSec** indicates the timeout period and its unit is ms. If no picture is obtained within **u32MilliSec**, a timeout error occurs.
- The 16-bit raw data after ISP processing can be obtained if the channel attribute is set to **PIXEL_FORMAT_RGB_BAYER**. However, **PIXEL_FORMAT_RGB_BAYER** is not supported when LDC or rotation is enabled.
- It is recommended that the ISP be frozen in the scenario where the raw data is dumped.

[Example]

```
HI_S32 s32ret;  
VI_CHN ViChn = 0;
```



```
VIDEO_FRAME_INFO_S stFrame;
HI_U32 u32Depth;
/* set max depth */
u32Depth = 10;
s32ret = HI_MPI_VI_SetFrameDepth(ViChn, u32Depth);
if (HI_SUCCESS != s32ret)
{
    printf("set max depth err:0x%x\n", s32ret);
    return s32ret;
}

/* get video frame from vi chn */
s32ret = HI_MPI_VI_GetFrame(ViChn, &stFrame)
if (HI_SUCCESS != s32ret)
{
    printf("get vi frame err:0x%x\n", s32ret);
    return s32ret;
}
/* deal with video frame ... */
/* release video frame */
(void)HI_MPI_VI_ReleaseFrame(ViChn, &stFrame);
```

[See Also]

- [HI_MPI_VI_ReleaseFrame](#)
- [HI_MPI_VI_SetChnAttr](#)

HI_MPI_VI_ReleaseFrame

[Description]

Releases the buffer occupied by VI pictures.

[Syntax]

```
HI_S32 HI_MPI_VI_ReleaseFrame(VI_CHN ViChn, VIDEO_FRAME_INFO_S
*pstFrameInfo);
```

[Parameter]

Parameter	Description	Input/Output
ViChn	VI channel ID Value range: [0, VIU_MAX_CHN_NUM)	Input
pstFrameInfo	Pointer to the storage structure of VI frames	Input

[Return Value]



Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

- Only the offline mode is supported.
- Ensure that the information about the pstFrameInfo structure is the same as the obtained information. Otherwise, the buffer fails to be released.
- After obtaining source pictures by calling HI_MPI_VI_GetFrame, you must call HI_MPI_VI_ReleaseFrame to release the buffer for storing the source pictures. That is, HI_MPI_VI_ReleaseFrame must be called if [HI_MPI_VI_GetFrame](#) is called.

[Example]

See the example of [HI_MPI_VI_GetFrame](#).

[See Also]

[HI_MPI_VI_GetFrame](#)

HI_MPI_VI_SetUserPic

[Description]

Sets a user picture that is used as the picture inserted when there is no video signal.

[Syntax]

```
HI_S32 HI_MPI_VI_SetUserPic(VI_CHN ViChn, VI_USERPIC_ATTR_S *pstUsrPic);
```

[Parameter]

Parameter	Description	Input/Output
ViChn	VI channel ID Value range: [0, VIU_MAX_CHN_NUM)	Input
pstUsrPic	Pointer to the structure of the user picture information	Input

[Return Value]



Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

Chip	Value Ranges for the Width and Height of the User Picture
Hi3516A	Width: [64, 2592] Height: [64, 2592]
Hi3518E V200	Width: [64, 2048] Height: [64, 2048]
Hi3519 V100	Width: [64, 4608] Height: [64, 4096]

Chip	Supported Pixel Format
Hi3516A	Semi-planar 4:2:2 and semi-planar 4:2:0
Hi3518E V200	Semi-planar 4:2:2, semi-planar 4:2:0, and single component
Hi3519 V100	Semi-planar 4:2:2, semi-planar 4:2:0, and single component

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

- Only the offline mode is supported.
- This MPI supports two user picture types: YUV picture and background picture only with a color.
- This MPI is used to replace the video picture output by the VI channel with a user-defined YUV picture or a background picture only with a color rather than the video data from the codec. This MPI must work with [HI_MPI_VI_EnableUserPic](#) and [HI_MPI_VI_DisableUserPic](#).

Currently, the ID of the VI channel is not used. That is, when you set a user picture, the setting is valid for all VI channel instead of a specific VI channel.

- After setting a user picture, you can insert a user picture into a specified VI channel by calling [HI_MPI_VI_EnableUserPic](#). Then the VI channel outputs the configured YUV picture or a background picture only with a color. Typically, the inserted picture is displayed and indicates no video when video signals are lost.



Generally, the size of the configured user picture should be the same as that of the VI channel. If the sizes are inconsistent, the user picture is automatically scaled to fit into the VI channel. If pictures with different resolutions are captured in the VI channel, you are advised to set the size of the user picture to the size of the picture with a higher resolution and set the capture mode to interlaced mode. Otherwise, jitter may occur when pictures are previewed on a VO device.

- For the video frame information structure of a user picture, you need to set the picture width, picture height, picture stride, YUV format, and physical addresses of the Y component and C component. You can obtain a video buffer from the public buffer pool of the MPP, obtain the physical address for storing the YUV data by reading the buffer information, and map the physical address into the user space. Then you can fill YUV data into the buffer. You must fill the Y component and interleaved data of U and V components in sequence. In little endian mode, the V and U components are filled in sequence.

When the function of inserting a user picture is enabled, the VIU uses the physical address in the user picture frame information. Therefore, after setting a user picture, do not release or destroy the corresponding buffer unless the buffer is no longer used. To modify the information about the picture, you can call HI_MPI_VI_SetUserPi again to configure another video buffer.

- You can dynamically call this MPI when you have enabled a VI channel or the function of inserting a user picture.
- It is recommended that the user picture after rotation be inserted in the rotation scenario. To support the user picture after the rotation function is enabled, the limit on the height of the user picture is different from that of the physical channel.

[Example]

```
HI_S32 s32ret;
HI_U32 u32Width;
HI_U32 u32Height;
HI_U32 u32LStride;
HI_U32 u32CStride;
HI_U32 u32LumaSize;
HI_U32 u32ChrmSize;
HI_U32 u32Size;
VB_BLK VbBlk;
HI_U32 u32PhyAddr;
HI_U8 *pVirAddr;

/*You need to obtain the picture width and height.*/
u32LumaSize = (u32LStride * u32Height);
u32ChrmSize = (u32CStride * u32Height) >> 2; /* 420*/
u32Size = u32LumaSize + (u32ChrmSize << 1);

/*Obtain video buffer block form common pool.*/
VbBlk = HI_MPI_VB_GetBlock(VB_INVALID_POOLID, u32Size);
if (VB_INVALID_HANDLE == VbBlk)
{
    return -1;
```



```
}

/*Obtain physical address*/
u32PhyAddr = HI_MPI_VB_Handle2PhysAddr(VbBlk);
if (0 == u32PhyAddr)
{
    return -1;
}

/* mmap physical address to virtual address*/
/* ... ... */

/*Obtain the pool ID.*/
pstVFrameInfo->u32PoolId = HI_MPI_VB_Handle2PoolId(VbBlk);
if (VB_INVALID_POOLID == pstVFrameInfo->u32PoolId)
{
    return -1;
}
pstVFrameInfo->stVFrame.u32PhyAddr[0] = u32PhyAddr;
pstVFrameInfo->stVFrame.u32PhyAddr[1] = pstVFrameInfo-
>stVFrame.u32PhyAddr[0] + u32LumaSize;
pstVFrameInfo->stVFrame.u32PhyAddr[2] = pstVFrameInfo-
>stVFrame.u32PhyAddr[1] + u32ChrmSize;

pstVFrameInfo->stVFrame.pVirAddr[0] = pVirAddr;
pstVFrameInfo->stVFrame.pVirAddr[1] = pstVFrameInfo-
>stVFrame.pVirAddr[0] + u32LumaSize;
pstVFrameInfo->stVFrame.pVirAddr[2] = pstVFrameInfo-
>stVFrame.pVirAddr[1] + u32ChrmSize;

pstVFrameInfo->stVFrame.u32Width = u32Width;
pstVFrameInfo->stVFrame.u32Height = u32Height;
pstVFrameInfo->stVFrame.u32Stride[0] = u32LStride;
pstVFrameInfo->stVFrame.u32Stride[1] = u32CStride;
pstVFrameInfo->stVFrame.u32Stride[2] = u32CStride;
pstVFrameInfo->stVFrame.enPixelFormat =
PIXEL_FORMAT_YUV_SEMIPLANAR_420;

/*Now you need obtain YUV Semi Planar Data, fill them into the virtual
address */
/* ... ... */
/* ... ... */

/*Enable VI channel ... ... */
```



```
/*First set user pic info*/
s32ret = HI_MPI_VI_SetUserPic(0, pstVFrameInfo);
if (s32ret)
{
    return -1;
}

/* ... ... */

/*Enable the function of inserting user pictures as required.*/
s32ret = HI_MPI_VI_EnableUserPic(0);
if (s32ret)
{
    return -1;
}
/* ... ... */

/*Disable insert user pic if you do not need.*/
s32ret = HI_MPI_VI_DisableUserPic(0);
if (s32ret)
{
    return -1;
}
```

[See Also]

None

HI_MPI_VI_EnableUserPic

[Description]

Enables the function of inserting a user picture.

[Syntax]

```
HI_S32 HI_MPI_VI_EnableUserPic(VI_CHN ViChn);
```

[Parameter]

Parameter	Description	Input/Output
ViChn	VI channel ID Value range: [0, VIU_MAX_PHYCHN_NUM)	Input

[Return Value]

Return Value	Description
0	Success



Return Value	Description
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

- Only the offline mode is supported.
- Before enabling the function of inserting a user picture, ensure that the frame information about the user picture is set.
- After the function of inserting user pictures is enabled, the current VI channel immediately outputs the configured user picture.
- This MPI can be repeatedly called.

[Example]

See the example of [HI_MPI_VI_SetUserPic](#).

[See Also]

None

HI_MPI_VI_DisableUserPic

[Description]

Disables the function of inserting a user picture.

[Syntax]

```
HI_S32 HI_MPI_VI_DisableUserPic(VI_CHN ViChn);
```

[Parameter]

Parameter	Description	Input/Output
ViChn	VI channel ID Value range: [0, VIU_MAX_PHYCHN_NUM)	Input

[Return Value]

Return Value	Description
0	Success



Return Value	Description
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

- Only the offline mode is supported.
- When the user picture is not required, you can call this MPI to restore the output of the source video data from the ADC.
- This MPI can be repeatedly called.

[Example]

See the example of [HI_MPI_VI_SetUserPic](#).

[See Also]

None

HI_MPI_VI_BindChn

[Description]

Binds a VI channel. This MPI is an advanced interface.

[Syntax]

```
HI_S32 HI_MPI_VI_BindChn (VI_CHN ViChn, const VI_CHN_BIND_ATTR_S  
*pstChnBindAttr);
```

[Parameter]

Parameter	Description	Input/Output
ViChn	VI channel ID	Input
pstChnBindAttr	Pointer to the binding attribute of a VI channel (static attribute)	Input

[Return Value]

Return Value	Description
0	Success



Return Value	Description
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

Chip	Supported or Not	Channel ID Range
Hi3516A/Hi3518E V200/Hi3519 V100	Not supported	None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

- This MPI is called only when you want to change the default binding relationship or bind multiple channels to the same {Dev, Way}. Before calling this MPI, you must cancel the default binding relationship by calling [HI_MPI_VI_UnBindChn](#).
- You must configure the attributes of a VI device before binding the device to a VI channel.
- A VI channel cannot be bound repeatedly. If a VI channel has been bound, you can bind it again only after it is unbound.
- The binding relationship is a static attribute. You must disable channels before configuring the binding relationship.

[Example]

None

[See Also]

None

HI_MPI_VI_UnBindChn

[Description]

Unbinds a VI channel. This MPI is an advanced interface.

[Syntax]

```
HI_S32 HI_MPI_VI_UnBindChn (VI_CHN ViChn);
```

[Parameter]

Parameter	Description	Input/Output
ViChn	VI channel ID	Input



[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

Chip	Supported or Not	Channel ID Range
Hi3516A/Hi3518E V200/Hi3519 V100	Not Supported	None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

- This MPI is called to cancel the binding relationship when you want to change the default binding relationship.
- The binding relationship can be canceled only when channels are disabled.
- No error code indicating failure is returned if a VI channel is unbound repeatedly.

[Example]

None

[See Also]

None

HI_MPI_VI_GetChnBind

[Description]

Obtains the binding relationship of a VI channel.

[Syntax]

```
HI_S32 HI_MPI_VI_GetChnBind(VI_CHN ViChn, VI_CHN_BIND_ATTR_S  
*pstChnBindAttr);
```

[Parameter]

Parameter	Description	Input/Output
ViChn	VI channel ID	Input
pstChnBindAttr	Pointer to the binding attribute of a VI channel	Output



[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

Chip	Setting Binding Relationship	Channel ID Range
Hi3516A/Hi3518E V200/Hi3519 V100	Not supported	None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

The default binding relationship of a VI channel can be obtained only after you set the attributes of the device port corresponding to the VI channel. Note that only the primary channel ID can be used to obtain the binding relationship.

[Example]

None

[See Also]

None

HI_MPI_VI_GetFd

[Description]

Obtains the device FD corresponding to a VI channel.

[Syntax]

```
HI_S32 HI_MPI_VI_GetFd(VI_CHN ViChn);
```

[Parameter]

Parameter	Description	Input/Output
ViChn	VI channel ID Value range: [0, VIU_MAX_CHN_NUM)	Input



[Return Value]

Return Value	Description
Positive number	Valid return value
Zero or negative number	Invalid return value

[Difference]

None

[Error Code]

None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

After obtaining the FD, you can obtain the video frames from multiple channels by calling the select function.

[Example]

```
VI_CHN ViChn = 0;  
HI_S32 s32ViFd = 0;  
  
/* enable vi dev and vi chn */  
/* get video frame from vi chn */  
s32ViFd = HI_MPI_VI_GetFd(ViChn);  
if (s32ViFd <= 0)  
{  
    return HI_FAILURE;  
}
```

[See Also]

None

HI_MPI_VI_CloseFd

[Description]

Closes the FD of the device and channel.

[Syntax]

```
HI_S32 HI_MPI_VI_CloseFd(HI_VOID);
```

[Parameter]

None



[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

None

[See Also]

[HI_MPI_VI_GetFd](#)

HI_MPI_VI_Query

[Description]

Queries the information about a VI channel such as the interrupt count and average frame rate.

[Syntax]

```
HI_S32 HI_MPI_VI_Query(VI_CHN ViChn, VI_CHN_STAT_S *pstStat);
```

[Parameter]

Parameter	Description	Input/Output
ViChn	VI channel ID Value range: [0, VIU_MAX_PHYCHN_NUM)	Input
pstStat	Pointer to the structure of the channel information	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."



[Difference]

None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

- The offline mode and online mode are supported.
- By calling this MPI, you can query the interrupt count, channel enable status, average frame rate, lost interrupt count, number of times that VBs fail to be obtained, picture width, and picture height.
- You can also query the current interrupt count by calling this MPI to check whether interrupts are generated.
- The frame rate queried by calling this MPI is the average frame rate that is calculated every 10 seconds. Therefore, the average frame rate is inaccurate.
- If the queried lost interrupt count continuously increases, an exception occurs in the VI channel.

[Example]

None

[See Also]

None

HI_MPI_VI_SetFlashConfig

[Description]

Sets the camera flash.

[Syntax]

```
HI_S32 HI_MPI_VI_SetFlashConfig(VI_DEV ViDev, const VI_FLASH_CONFIG_S  
*pstFlashConfig);
```

[Parameter]

Parameter	Description	Input/Output
ViDev	VI device ID Value range: [0, VIU_MAX_DEV_NUM)	Input
pstFlashConfig	Pointer to camera flash settings	Input

[Return Value]

Return Value	Description
0	Success



Return Value	Description
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

- The camera flash works in single-blink or multi-blink mode. In multi-blink mode, u32Interval must be greater than u32CapFrmIndex.
- For details about camera flash settings, see [VI_FLASH_CONFIG_S](#).

[Example]

```
HI_S32 s32Ret;
VI_DEV ViDev = 0;
VI_FLASH_CONFIG_S stFlashConfig;

/* first enable vi device and vi chn */

/* Init flash config */
stFlashConfig.enFlashMode = VI_FLASH_ONCE;
stFlashConfig.u32StartTime = 100;
stFlashConfig.u32Duration = 400;
stFlashConfig.u32CapFrmIndex = 2;

/* Set configuration for vi flash */
s32Ret = HI_MPI_VI_SetFlashConfig(ViDev, &stFlashConfig);
if (s32Ret != HI_SUCCESS)
{
    printf("Set flash config failed with error code %#x!\n", s32Ret);
    return HI_FAILURE;
}

/* Get configuration for vi flash */
s32Ret = HI_MPI_VI_GetFlashConfig(ViDev, &stFlashConfig);
if (s32Ret != HI_SUCCESS)
{
    printf("Get flash config failed with error code %#x!\n", s32Ret);
    return HI_FAILURE;
}
```



```
    }
    s32Ret = HI_MPI_VI_TriggerFlash(ViDev, HI_TRUE);
    if (s32Ret != HI_SUCCESS)
    {
        printf("Trigger flash failed with error code %#x!\n", s32Ret);
        return HI_FAILURE;
    }
```

[See Also]

- [HI_MPI_VI_GetFlashConfig](#)
- [HI_MPI_VI_TriggerFlash](#)

HI_MPI_VI_GetFlashConfig

[Description]

Obtains camera flash settings.

[Syntax]

```
HI_S32 HI_MPI_VI_GetFlashConfig(VI_DEV ViDev, VI_FLASH_CONFIG_S
*pstFlashConfig);
```

[Parameter]

Parameter	Description	Input/Output
ViDev	VI device ID Value range: [0, VIU_MAX_DEV_NUM)	Input
pstFlashConfig	Pointer to camera flash settings	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]



Before obtaining camera flash settings, you must set the camera flash. Otherwise, an error code indicating failure is returned.

[Example]

None

[See Also]

- [HI_MPI_VI_SetFlashConfig](#)
- [HI_MPI_VI_TriggerFlash](#)

HI_MPI_VI_TriggerFlash

[Description]

Enables or disables the camera flash.

[Syntax]

```
HI_S32 HI_MPI_VI_TriggerFlash (VI_DEV ViDev, HI_BOOL bEnable);
```

[Parameter]

Parameter	Description	Input/Output
ViDev	VI device ID Value range: [0, VIU_MAX_DEV_NUM)	Input
bEnable	Camera flash enable	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

Configure attributes of a camera flash before enabling or disabling it. Otherwise, an error code indicating failure is returned.

[Example]



None

[See Also]

- [HI_MPI_VI_SetFlashConfig](#)
- [HI_MPI_VI_GetFlashConfig](#)

HI_MPI_VI_SetExtChnAttr

[Description]

Sets the attributes of an extended VI channel.

[Syntax]

```
HI_S32 HI_MPI_VI_SetExtChnAttr(VI_CHN ViChn, const VI_EXT_CHN_ATTR_S *pstExtChnAttr);
```

[Parameter]

Parameter	Description	Input/Output
ViChn	ID of an extended VI channel Value range: [VIU_EXT_CHN_START, VIU_MAX_CHN_NUM)	Input
pstExtChnAttr	Pointer to extended channel attributes	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

Only the offline mode is supported.

The following describes extended channel attributes:

- Target picture size (**stDestSize**)
 - This attribute must be set and its value cannot be less than the minimum picture size or greater than the maximum picture size supported by the extended channel after



scaling. Otherwise, the extended channel has no output. The maximum minification multiple of the extended channel is [VIU_EXTCHN_MINIFICATION](#).

- **u32Width** must be 2-pixel-aligned. In progressive capture mode, **u32Height** must be 2-pixel-aligned; in interlaced dual-field capture mode, **u32Height** must be 4-pixel-aligned.
- Source frame rate (**s32SrcFrameRate**): If you do not want to control the frame rate, set **s32SrcFrameRate** to **-1**.
- Target frame rate (**s32DstFrameRate**): If you do not want to control the frame rate, set **s32FrameRate** to **-1**. If you want to control the frame rate, you must set the source frame rate and target frame rate in sequence, and the target frame rate must be less than or equal to the source frame rate.
- The output frame rate of an extended channel is obtained by referring to its control frame rates and the frame rates of its parent channel. Assume that the source frame rate of a sensor is 30 frame/s, the parent channel of the extended channel is physical channel 0, the source frame rate and target frame rate of channel 0 are 30 frame/s and 10 frame/s respectively, and the source frame rate and target frame rate of the extended channel is 10 frame/s and 2 frame/s. The actual frame rate of the extended channel is 2 frame/s.
- When the pixel format for the parent channel is the single component format, the pixel format for the extended channel must also be the single component format.
- The number of extended channels bound to a channel cannot exceed the maximum number of [VIU_MAX_EXTCHN_BIND_PER_CHN](#). Otherwise, binding fails.

[Example]

```
HI_S32 s32Ret;
VI_CHN ViChn = VIU_EXT_CHN_START;
VI_EXT_CHN_ATTR_S stExtChnAttr;

/*First enable VI devices and VI channels.*/

/*Initialize extended channel attributes.*/
stExtChnAttr.enPixelFormat = PIXEL_FORMAT_YUV_SEMIPLANAR_420;
stExtChnAttr.s32BindChn = 0;
stExtChnAttr.s32DstFrameRate = -1;
stExtChnAttr.s32SrcFrameRate = -1;
stExtChnAttr.stDestSize.u32Width = 720;
stExtChnAttr.stDestSize.u32Height = 576;

/*Set attributes of extended VI channels.*/
HI_MPI_VI_SetExtChnAttr(ViExtChn, &stExtChnAttr);
if (HI_SUCCESS != s32ret)
{
    printf("Set vi ext-chn attr err:0x%x\n", s32ret);
    return s32Ret;
}

/*Obtain attributes of extended VI channels.*/
s32Ret = HI_MPI_VI_GetExtChnAttr (ViChn, & stExtChnAttr);
```



```
if (HI_SUCCESS != s32ret)
{
    printf("Get vi ext-chn attr err:0x%x\n", s32ret);
    return s32Ret;
}
```

[See Also]

[HI_MPI_VI_GetExtChnAttr](#)

HI_MPI_VI_GetExtChnAttr

[Description]

Obtains the attributes of an extended VI channel.

[Syntax]

```
HI_S32 HI_MPI_VI_GetExtChnAttr(VI_CHN ViChn, VI_EXT_CHN_ATTR_S
*pstExtChnAttr);
```

[Parameter]

Parameter	Description	Input/Output
ViChn	ID of an extended VI channel Value range: [VIU_EXT_CHN_START, VIU_MAX_CHN_NUM)	Input
pstExtChnAttr	Pointer to extended channel attributes	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

- Only the offline mode is supported.



- Before obtaining extended channel attributes, you must set them. Otherwise, an error code indicating failure is returned.

[Example]

None

[See Also]

[HI_MPI_VI_SetExtChnAttr](#)

HI_MPI_VI_SetExtChnCrop

[Description]

Sets the cropping attributes of an extended VI channel.

[Syntax]

```
HI_S32 HI_MPI_VI_SetExtChnCrop(VI_CHN ViChn, const CROP_INFO_S
*pstExtChnCrop);
```

[Parameter]

Parameter	Description	Input/Output
ViChn	ID of an extended VI channel Value range: [VIU_EXT_CHN_START, VIU_MAX_CHN_NUM)	Input
pstExtChnCrop	Pointer to the cropping attributes of an extended VI channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

- Only the offline mode is supported.



- Note the following restrictions when configuring the cropping attributes of an extended VI channel:
 - You must configure the attributes of an extended VI channel before configuring the cropping attributes of the extended VI channel.
 - The cropping attributes are dynamic attributes.
 - The start coordinates of the rectangular region for the cropping attributes are used to configure the position of the rectangular picture to be cropped in the extended channel relative to the start point of the picture in the parent channel. For the start point, the horizontal coordinate is in the unit of pixel and the vertical coordinate is in the unit of line.
 - For the start point of the rectangular region for the cropping attributes, **s32X**, **s32Y**, and **u32Width** must be 2-pixel-aligned, and **u32Height** must be 2-pixel-aligned in progressive capture mode or 4-pixel-aligned in interlaced two-field capture mode.
 - For the start point, **s32X** and **s32Y** must be set to appropriate values to ensure that the width and height of the rectangle to be cropped are less than or equal to the width and height of the actual output picture of the parent channel.
 - The width (**u32Width**) and height (**u32Height**) of the rectangle to be cropped cannot be greater than the width and height of the actual output picture of the parent channel.
 - For the start point, the sum of **s32X** and **u32Width** cannot be greater than the maximum width of the actual output picture of the parent channel, and the sum of **s32Y** and **u32Height** cannot be greater than the maximum height of the actual output picture of the parent channel.

[Example]

```
HI_S32 s32Ret;
VI_CHN ViChn = VIU_EXT_CHN_START;
VI_EXT_CHN_ATTR_S stExtChnAttr;
CROP_INFO_S stExtChnCrop;

/* First enable vi device and vi chn */

/* Init ext-chn attr */
stExtChnAttr.enPixelFormat = PIXEL_FORMAT_YUV_SEMIPLANAR_420;
stExtChnAttr.s32BindChn = 0;
stExtChnAttr.s32DstFrameRate = -1;
stExtChnAttr.s32SrcFrameRate = -1;
stExtChnAttr.stDestSize.u32Width = 1920;
stExtChnAttr.stDestSize.u32Height = 1080;

/* Set attribute for vi ext-chn */
HI_MPI_VI_SetExtChnAttr(ViExtChn, &stExtChnAttr);
if (HI_SUCCESS != s32ret)
{
    printf("Set vi ext-chn attr err:0x%x\n", s32ret);
    return s32Ret;
}
```



```
/* Get attribute for vi ext-chn */
s32Ret = HI_MPI_VI_GetExtChnAttr (ViChn, & stExtChnAttr);
if (HI_SUCCESS != s32ret)
{
    printf("Get vi ext-chn attr err:0x%x\n", s32ret);
    return s32Ret;
}

/* Init ext-chn crop attr */
stExtChnCrop.bEnable = HI_TRUE;
stExtChnCrop.stRect.s32X = 800;
stExtChnCrop.stRect.s32Y = 200;
stExtChnCrop.stRect.u32Height = 360;
stExtChnCrop.stRect.u32Width = 480;

/* Set crop attribute for vi ext-chn */
HI_MPI_VI_SetExtChnCrop(ViExtChn, &stExtChnCrop);
if (HI_SUCCESS != s32ret)
{
    printf("Set vi ext-chn crop attr err:0x%x\n", s32ret);
    return s32Ret;
}

/* Get attribute for vi ext-chn */
s32Ret = HI_MPI_VI_GetExtChnAttr (ViChn, & stExtChnAttr);
if (HI_SUCCESS != s32ret)
{
    printf("Get vi ext-chn attr err:0x%x\n", s32ret);
    return s32Ret;
}

/* Enable vi ext-chn */
s32Ret = HI_MPI_VI_EnableChn (ViChn);
if (HI_SUCCESS != s32ret)
{
    printf("Enable chn failed with error code %#x!\n", s32Ret);
    return s32Ret;
}
```

[See Also]

[HI_MPI_VI_GetExtChnCrop](#)



HI_MPI_VI_GetExtChnCrop

[Description]

Obtains the cropping attributes of an extended VI channel.

[Syntax]

```
HI_S32 HI_MPI_VI_GetExtChnCrop(VI_CHN ViChn, CROP_INFO_S *pstExtChnCrop);
```

[Parameter]

Parameter	Description	Input/Output
ViChn	ID of an extended VI channel Value range: [VIU_EXT_CHN_START, VIU_MAX_CHN_NUM)	Input
pstExtChnCrop	Pointer to the cropping attributes of an extended VI channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

- Only the offline mode is supported.
- You must set the attributes and cropping attributes of an extended VI channel before obtaining them. Otherwise, an error code indicating failure is returned.

[Example]

None

[See Also]

[HI_MPI_VI_SetExtChnCrop](#)

HI_MPI_VI_SetLDCAttr

[Description]



Sets the LDC attribute.

[Syntax]

```
HI_S32 HI_MPI_VI_SetLDCAttr(VI_CHN ViChn, const VI_LDC_ATTR_S  
*pstLDCAttr);
```

[Parameter]

Parameter	Description	Input/Output
ViChn	ID of a physical VI channel Value range: [0, VIU_MAX_PHYCHN_NUM)	Input
pstLDCAttr	Pointer to the LDC attribute	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

Chip	Supported Pixel Format
Hi3516A	PIXEL_FORMAT_YUV_SEMIPLANAR_422 PIXEL_FORMAT_YUV_SEMIPLANAR_420
Hi3518E V200	PIXEL_FORMAT_YUV_SEMIPLANAR_422 PIXEL_FORMAT_YUV_SEMIPLANAR_420 PIXEL_FORMAT_YUV_400
Hi3519 V100	Not supported

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

- Only the offline mode is supported.
- Call this MPI only after setting channel attributes.
- The LDC attributes can be dynamically changed.
- Only the pictures output in non-field mode are supported.
- Only the LDC attributes of physical channels can be set.



- Hi3519 V100 does not support LDC.
- When LDC is enabled, you are advised to use the CoverEx region to replace the Cover region, because the Cover region may be distorted.

[Example]

```
HI_S32 s32Ret;
VI_LDC_ATTR_S stLDCAttr;

/*First enable VI devices and VI channel.*/

/*Initialize LDC attributes.*/
stLDCAttr.bEnable = HI_TRUE;
stLDCAttr.stAttr.enViewType = LDC_VIEW_TYPE_ALL; //LDC_VIEW_TYPE_CROP;
stLDCAttr.stAttr.s32CenterXOffset = 0;
stLDCAttr.stAttr.s32CenterYOffset = 0;
stLDCAttr.stAttr.s32Ratio = 255;

/*Set LDC attributes.*/
s32Ret = HI_MPI_VI_SetLDCAttr(ViChn, &stLDCAttr);
if (HI_SUCCESS != s32ret)
{
    printf("Set vi LDC attr err:0x%x\n", s32ret);
    return s32Ret;
}

/*Obtain LDC attributes.*/
s32Ret = HI_MPI_VI_GetLDCAttr (ViChn, &stLDCAttr);
if (HI_SUCCESS != s32ret)
{
    printf("Get vi LDC attr err:0x%x\n", s32ret);
    return s32Ret;
}
```

[See Also]

[HI_MPI_VI_GetLDCAttr](#)

HI_MPI_VI_GetLDCAttr

[Description]

Obtains the LDC attribute.

[Syntax]

```
HI_S32 HI_MPI_VI_GetLDCAttr(VI_CHN ViChn, VI_LDC_ATTR_S *pstLDCAttr);
```

[Parameter]



Parameter	Description	Input/Output
ViChn	ID of a physical VI channel Value range: [0, VIU_MAX_PHYCHN_NUM)	Input
pstLDCAttr	Pointer to the LDC attribute	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

Chip	Supported or Not
Hi3516A	Supported
Hi3518EV200	Supported
Hi3519V100	Not supported

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

Only the offline mode is supported.

[Example]

None

[See Also]

[HI_MPI_VI_SetLDCAttr](#)

HI_MPI_VI_SetCSCAttr

[Description]

Sets the CSC attribute of a VI device.

[Syntax]

```
HI_S32 HI_MPI_VI_SetCSCAttr(VI_DEV ViDev, const VI_CSC_ATTR_S  
*pstCSCAttr);
```



[Parameter]

Parameter	Description	Input/Output
ViDev	VI device ID Value range: [0, VIU_MAX_DEV_NUM)	Input
pstCSCAttr	Pointer to CSC attributes	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

- Call this MPI after setting the device attributes.
- This MPI is used to adjust the luminance, hue, contrast, and saturation.
- For the HD sensor, set enViCscType to VI_CSC_TYPE_709; for the SD sensor, set enViCscType to VI_CSC_TYPE_601.
- The value range of luminance, hue, contrast, or saturation is [0, 100], and the default value is 50.
- Parameters of the CSC attribute are as follows:

Parameter	Description
ext_csc_en	Luminance adjustment range 0 (default): [-32, +32] 1: [-128, +128] Note: The adjustment range is not the value range of the parameter. For details about the value range, see VI_CSC_ATTR_S .
csc_ct_mode	0: The luminance changes strongly with contrast adjustment. 1 (default): The luminance changes weakly with contrast adjustment.



Parameter	Description
csc_tv_en	Value range of the CSC constant coefficient matrix 0 (default): [0, 255] 1: [16, 235]

[Example]

```
HI_S32 s32Ret;
VI_CSC_ATTR_S stCscAttr;
/*First enable VI devices and VI channels.*/

/*Initialize CSC attributes.*/
stCscAttr.enViCscType = VI_CSC_TYPE_709;
stCscAttr.u32ContrVal = 50;
stCscAttr.u32HueVal = 50;
stCscAttr.u32LumaVal = 50;
stCscAttr.u32SatuVal = 50;

/*Set CSC attributes.*/
s32Ret = HI_MPI_VI_SetCSCAttr(ViDev, &stCscAttr);
if (HI_SUCCESS != s32ret)
{
    printf("Set vi CSC attr err:0x%x\n", s32ret);
    return s32Ret;
}

/*Obtain CSC attributes.*/
s32Ret = HI_MPI_VI_GetCSCAttr(ViDev, &stCscAttr);
if (HI_SUCCESS != s32ret)
{
    printf("Get vi CSC attr err:0x%x\n", s32ret);
    return s32Ret;
}
```

[See Also]

None

HI_MPI_VI_GetCSCAttr

[Description]

Obtains the CSC attribute of a VI device.

[Syntax]

```
HI_S32 HI_MPI_VI_GetCSCAttr(VI_DEV ViDev, VI_CSC_ATTR_S *pstCSCAttr);
```



[Parameter]

Parameter	Description	Input/Output
ViDev	VI device ID Value range: [0, VIU_MAX_DEV_NUM)	Input
pstCSCAttr	Pointer to CSC attributes	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

None

[Example]

None

[See Also]

None

HI_MPI_VI_SetRotate

[Description]

Sets the rotation attribute of VI pictures.

[Syntax]

```
HI_S32 HI_MPI_VI_SetRotate(VI_CHN ViChn, const ROTATE_E enRotate);
```

[Parameter]

Parameter	Description	Input/Output
ViChn	ID of a physical VI channel Value range: [0, VIU_MAX_PHYCHN_NUM)	Input



Parameter	Description	Input/Output
enRotate	Rotation attribute	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

- Only the offline mode is supported.
- The rotation attribute is a static attribute and cannot be dynamically modified. Set the rotation attribute after you set the channel attributes but before you enable a channel.
- Only the sp420 and YUV400 pictures that are output in non-field mode can be rotated.
- Pictures can be rotated only by 90°, 180°, or 270°.
- Only the pictures in a physical channel can be rotated.
- To rotate a picture by 180°, you are advised to use the mirroring and flipping functions. This reduces the bandwidth usage and improves performance.
- After the rotation attribute is set, the width and height of the output pictures from a physical channel may change. However, the width and height obtained from the channel attributes are still configured values. For example, if a 720p input picture is rotated by 90° or 270°, the actual width and height of the output picture is (720, 1280), but the width and height (DestSize) obtained by calling [HI_MPI_VI_GetChnAttr](#) is (1280, 720).
- After the rotation attribute is set, the user picture size must be set based on the picture size after rotation.
- After the rotation attribute is set, compression is not supported.
- The rotation function cannot be configured if the width of the physical channel for Hi3519 V100 exceeds 4096.

[Example]

```
HI_S32 s32Ret;  
VI_CHN ViChn = 0;  
ROTATE_E enRotate;  
  
/*First enable VI devices.*/
```



```
/*Set VI channel attributes.*/
s32Ret = HI_MPI_VI_SetChnAttr(ViChn, &stChnAttr);
if (s32Ret != HI_SUCCESS)
{
    printf("Set chn attributes failed with error code %#x!\n", s32Ret);
    return HI_FAILURE;
}

/*Set the rotation angle to 90°.*/
s32Ret = HI_MPI_VI_SetRotate(ViChn, ROTATE_90);
if (HI_SUCCESS != s32ret)
{
    printf("Set vi rotate err:0x%x\n", s32ret);
    return s32Ret;
}

s32Ret = HI_MPI_VI_EnableChn(ViChn);
if (s32Ret != HI_SUCCESS)
{
    printf("Enable chn failed with error code %#x!\n", s32Ret);
    return HI_FAILURE;
}

/*Obtain rotation attributes.*/
s32Ret = HI_MPI_VI_GetRotate(ViChn, &enRotate);
if (HI_SUCCESS != s32ret)
{
    printf("Get vi rotate err:0x%x\n", s32ret);
    return s32Ret;
}
```

[See Also]

None

HI_MPI_VI_GetRotate

[Description]

Obtains the rotation attribute of VI pictures.

[Syntax]

```
HI_S32 HI_MPI_VI_GetRotate(VI_CHN ViChn, ROTATE_E *penRotate);
```

[Parameter]



Parameter	Description	Input/Output
ViChn	ID of a physical VI channel Value range: [0, VIU_MAX_PHYCHN_NUM)	Input
penRotate	Pointer to the rotation attribute	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

Only the offline mode is supported.

[Example]

None

[See Also]

[HI_MPI_VI_SetRotate](#)

HI_MPI_VI_GetChnLuma

[Description]

Obtains the channel luminance statistics.

[Syntax]

```
HI_S32 HI_MPI_VI_GetChnLuma(VI_CHN ViChn, VI_CHN_LUM_S *pstLuma);
```

[Parameter]

Parameter	Description	Input/Output
ViChn	ID of a physical VI channel Value range: [0, VIU_MAX_PHYCHN_NUM)	Input
pstLuma	Pointer to the luminance statistics	Output



[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

None

[Example]

None

[See Also]

None

HI_MPI_VI_SetFisheyeDevConfig

[Description]

Sets the LMF parameters of the fisheye lens corresponding to the VI device.

[Syntax]

```
HI_S32 HI_MPI_VI_SetFisheyeDevConfig(VI_DEV ViDev, const FISHEYE_CONFIG_S *pstFisheyeConfig);
```

[Parameter]

Parameter	Description	Input/Output
ViDev	VI device ID Value range: [0, VIU_MAX_DEV_NUM)	Input
pstFisheyeConfig	Pointer to the LMF parameters of the fisheye lens. For details, see chapter 11 "Fisheye Subsystem".	Input

[Return Value]



Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

Chip	Whether the Configuration of the Fisheye LMF Parameters Is Supported
Hi3516A	Not supported
Hi3518E V200	Not supported
Hi3519 V100	Supported

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

- The attributes of the VI device must be configured before this MPI is called.
- For details about the parameters, see chapter 11 "Fisheye Subsystem" in the *HiMPP IPC V2.0 Media Processing Software Development Reference*.

[Example]

None

[See Also]

- [HI_MPI_VI_GetFlashConfig](#)
- [HI_MPI_VI_SetFisheyeAttr](#)
- [HI_MPI_VI_GetFisheyeAttr](#)

HI_MPI_VI_GetFisheyeDevConfig

[Description]

Obtains the LMF parameters of the fisheye lens corresponding to the VI device.

[Syntax]

```
HI_S32 HI_MPI_VI_GetFisheyeDevConfig(VI_DEV ViDev, const FISHEYE_CONFIG_S *pstFisheyeConfig);
```

[Parameter]



Parameter	Description	Input/Output
ViDev	VI device ID Value range: [0, VIU_MAX_DEV_NUM)	Input
pstFisheyeConfig	Pointer to the LMF parameters of the fisheye lens. For details, see chapter 11 "Fisheye Subsystem".	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

Chip	Whether the Acquisition of the Fisheye LMF Parameters Is Supported
Hi3516A	Not supported
Hi3518E V200	Not supported
Hi3519 V100	Supported

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

- For details about the parameters, see chapter 11 "Fisheye Subsystem" in the *HiMPP IPC V2.0 Media Processing Software Development Reference*.
- The attributes can be obtained only after they are configured. If this MPI is called to obtain the attributes before they are configured, an error code indicating failure is returned.

[Example]

None

[See Also]

- [HI_MPI_VI_SetFisheyeDevConfig](#)
- [HI_MPI_VI_SetFisheyeAttr](#)
- [HI_MPI_VI_GetFisheyeAttr](#)



HI_MPI_VI_SetFisheyeAttr

[Description]

Sets the fisheye attribute corresponding to an extended VI channel.

[Syntax]

```
HI_S32 HI_MPI_VI_SetFisheyeAttr(VI_CHN ViChn, const FISHEYE_ATTR_S *pstFisheyeAttr);
```

[Parameter]

Parameter	Description	Input/Output
ViChn	ID of an extended VI channel Value range: [VIU_EXT_CHN_START, VIU_MAX_CHN_NUM)	Input
pstFisheyeAttr	Pointer to the fisheye attribute of an extended VI channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

Chip	Whether the Configuration of the Fisheye Attribute of the Extended VI Channel Is Supported
Hi3516A	Not supported
Hi3518E V200	Not supported
Hi3519 V100	Supported

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

- The channel ID must be the ID of an extended VI channel, and the attributes of the extended channel must be configured before this MPI is called.



- For details about the parameters, see chapter 11 "Fisheye Subsystem" in the *HiMPP IPC V2.0 Media Processing Software Development Reference*.

[Example]

None

[See Also]

[HI_MPI_VI_GetFisheyeAttr](#)

HI_MPI_VI_GetFisheyeAttr

[Description]

Obtains the fisheye attribute corresponding to an extended VI channel.

[Syntax]

```
HI_S32 HI_MPI_VI_GetFisheyeAttr(VI_CHN ViChn, FISHEYE_ATTR_S  
*pstFisheyeAttr);
```

[Parameter]

Parameter	Description	Input/Output
ViChn	ID of an extended VI channel Value range: [VIU_EXT_CHN_START, VIU_MAX_CHN_NUM)	Input
pstFisheyeAttr	Fisheye attribute of an extended VI channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

Chip	Whether the Acquisition of the Fisheye Attribute of the Extended VI Channel Is Supported
Hi3516A	Not supported
Hi3518E V200	Not supported
Hi3519 V100	Supported

[Requirement]



- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

- The channel ID must be the ID of an extended VI channel.
- For details about the parameters, see chapter 11 "Fisheye Subsystem" in the *HiMPP IPC V2.0 Media Processing Software Development Reference*.
- The attributes can be obtained only after they are configured. If this MPI is called to obtain the attributes before they are configured, an error code indicating failure is returned.

[Example]

None

[See Also]

[HI_MPI_VI_SetFisheyeAttr](#)

HI_MPI_VI_SetDevDumpAttr

[Description]

Sets the dump attribute of the VI device.

[Syntax]

```
HI_S32 HI_MPI_VI_SetDevDumpAttr(VI_DEV ViDev, const VI_DUMP_ATTR_S
*pstDumpAttr);
```

[Parameter]

Parameter	Description	Input/Output
ViDev	VI device ID Value range: [0, VIU_MAX_DEV_NUM)	Input
pstDumpAttr	Pointer to the dump attribute of the VI device	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]



Chip	Whether the Configuration of the Dump Attribute Is Supported
Hi3516A	Supported
Hi3518E V200	Supported
Hi3519 V100	Supported

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

- For Hi3518E V200, only the raw and IR data is dumped.
- For Hi3519 V100 and Hi3516A, only the raw data is dumped.
- This MPI must work with [HI_MPI_VI_EnableBayerDump](#). If the dump attribute is not configured and the dump function is directly enabled, the raw data is dumped by default.
- The dump attribute supports the following pixel formats:
PIXEL_FORMAT_RGB_BAYER_8BPP, PIXEL_FORMAT_RGB_BAYER_10BPP,
PIXEL_FORMAT_RGB_BAYER_12BPP, PIXEL_FORMAT_RGB_BAYER_14BPP,
and PIXEL_FORMAT_RGB_BAYER.
- This MPI must be called before [HI_MPI_VI_EnableBayerDump](#) is called; otherwise, the error code HI_ERR_VI_FAILED_NOTDISABLE is returned.

[Example]

None

[See Also]

- [HI_MPI_VI_GetDevDumpAttr](#)
- [HI_MPI_VI_EnableBayerDump](#)

HI_MPI_VI_GetDevDumpAttr

[Description]

Obtains the dump attribute of the VI device.

[Syntax]

```
HI_S32 HI_MPI_VI_GetDevDumpAttr(VI_DEV ViDev, VI_DUMP_ATTR_S  
*pstDumpAttr);
```

[Parameter]

Parameter	Description	Input/Output
ViDev	VI device ID Value range: [0, VIU_MAX_DEV_NUM)	Input



Parameter	Description	Input/Output
pstDumpAttr	Pointer to the dump attribute of the VI device	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

Chip	Whether the Acquisition of the Dump Attribute Is Supported
Hi3516A	Supported
Hi3518E V200	Supported
Hi3519 V100	Supported

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

None

[Example]

None

[See Also]

[HI_MPI_VI_SetDevDumpAttr](#)

HI_MPI_VI_EnableBayerDump

[Description]

Enables the function of obtaining raw data.

[Syntax]

`HI_S32 HI_MPI_VI_EnableBayerDump(VI_DEV ViDev);`

[Parameter]



Parameter	Description	Input/Output
ViDev	VI device ID Value range: [0, VIU_MAX_DEV_NUM)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

- The width and height of the stored raw data are the same as those in the cropping attributes of an AI device.
- If the BayerDump is used in WDR mode, the WDR picture may be abnormal because the WDR write channels are multiplexed with the BayerDump, and when the data is dumped, one of the WDR write channels is disabled.
- The raw data is stored by using virtual channels. Channel 17 is used to store the raw data because the maximum ID of the extended channel is 16. The channel ID of the raw data can be obtained by using [VIU_GET_RAW_CHN](#). The maximum captured picture depth for channel 17 must be set before the raw data is obtained.

[Example]

None

[See Also]

[HI_MPI_VI_DisableBayerDump](#)

HI_MPI_VI_DisableBayerDump

[Description]

Disables the function of obtaining raw data.

[Syntax]

```
HI_S32 HI_MPI_VI_DisableBayerDump(VI_DEV ViDev);
```

[Parameter]



Parameter	Description	Input/Output
ViDev	VI device ID Value range: [0, VIU_MAX_DEV_NUM)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

None

[Example]

None

[See Also]

[HI_MPI_VI_EnableBayerRead](#)

HI_MPI_VI_EnableBayerRead

[Description]

Enables the function of reading raw data.

[Syntax]

```
HI_S32 HI_MPI_VI_EnableBayerRead(VI_DEV ViDev);
```

[Parameter]

Parameter	Description	Input/Output
ViDev	VI device ID Value range: [0, VIU_MAX_DEV_NUM)	Input

[Return Value]



Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

None

[Example]

For details, see [HI_MPI_VI_SendBayerData](#).

[See Also]

- [HI_MPI_VI_SendBayerData](#)
- [HI_MPI_VI_DisableBayerRead](#)

HI_MPI_VI_SendBayerData

[Description]

Sends raw data to the ISP.

[Syntax]

```
HI_S32 HI_MPI_VI_SendBayerData(VI_DEV ViDev, const VI_RAW_DATA_INFO_S
*pstRawData, HI_S32 s32MilliSec);
```

[Parameter]

Parameter	Description	Input/Output
ViDev	VI device ID Value range: [0, VIU_MAX_DEV_NUM)	Input
pstRawData	Pointer to the raw data to be read	Input
s32MilliSec	-1: Obtain pictures in block mode. 0: Obtain pictures in non-block mode. > 0: Waiting until timeout occurs.	Input

[Return Value]



Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

- The front-end timing is required for reading raw data. Therefore, ensure that the timing of raw data to be read is consistent with the front-end timing.
- This MPI can be called after the [HI_MPI_VI_EnableBayerRead](#) is called. Otherwise, [HI_ERR_VI_FAILED_NOTENABLE](#) is returned.

[Example]

```
/* Read raw file function */
typedef struct hiVI_MST_RAW_FRM_S
{
    VB_BLK VbBlk;
    VI_RAW_DATA_INFO_S stRawDataInfo;
    HI_U32 u32FrmSize;
}VI_MST_RAW_FRM_S;

#define VI_MST_MAX_RAW_NUM 20
typedef struct hiVI_MST_RAW_INFO_S
{
    VI_MST_RAW_FRM_S astMstRawFrm[VI_MST_MAX_RAW_NUM];
    HI_U32 u32FrameNum;
}VI_MST_RAW_INFO_S;

HI_S32 VI_COMM_ReadRawFile(char* file_name,
                           PIXEL_FORMAT_E enPixelFormat,
                           HI_U32 u32Width, HI_U32 u32Height,
                           HI_U32 u32FrameNum,
                           VI_MST_RAW_INFO_S *pstMstRawInfo)
{
    FILE *pfd;
    HI_S32 s32Ret = HI_SUCCESS;
    int i = 0;
    HI_U32 u32FrmSize;
```



```
VIDEO_FRAME_S *pstVideoFrame;

/* open RAW file */
pfd = fopen(file_name, "rb");
if (!pfd)
{
    printf("open file -> %s fail \n", file_name);
    return HI_FAILURE;
}
pstMstRawInfo->u32FrameNum = u32FrameNum;
/* allocate raw data memory */
s32Ret = VI_COMM_GetRawMemInfo(enPixelFormat,
                                u32Width, u32Height,
                                u32FrameNum,
                                pstRawDataInfo);

if (HI_SUCCESS != s32Ret)
{
    s32Ret = HI_FAILURE;
    goto EXIT;
}
for (i = 0; i < u32FrameNum; i++)
{
    pstVideoFrame = &pstMstRawInfo-
>astMstRawFrm[i].stRawDataInfo.stFrame.stVFrame;
    u32FrmSize = pstVideoFrame->u32Stride[0]* pstVideoFrame->u32Height;
    fread(pstVideoFrame->pVirAddr[0], u32FrmSize, 1, pfd);
}
EXIT:
fclose(pfd);
return s32Ret;
}

VI_MST_RAW_INFO_S stMstRawInfo;
HI_U32 i;
HI_S32 s32MilliSec = 2000;
/* first enable vi device */
/* set vi chn attr */
s32Ret = HI_MPI_VI_SetChnAttr(ViChn, &stChnAttr);
if (s32Ret != HI_SUCCESS)
{
    printf("Set chn attributes failed with error code %#x!\n", s32Ret);
    return HI_FAILURE;
}
```



```
s32Ret = HI_MPI_VI_EnableChn(ViChn);
if (s32Ret != HI_SUCCESS)
{
    printf("Enable chn failed with error code %#x!\n", s32Ret);
    return HI_FAILURE;
}

VI_COMM_ReadRawFile("./data/vi_dev_0_1920_1080_10_16bits.raw",
                     PIXEL_FORMAT_RGB_BAYER,
                     1920, 1080,
                     10,
                     &stMstRawInfo);

HI_MPI_VI_EnableBayerRead(ViDev);
for (i=0; i<stMstRawInfo.u32FrameNum; i++)
{
    HI_MPI_VI_SendBayerData(ViDev,
                           &stMstRawInfo.astMstRawFrm[i].stRawDataInfo, s32MilliSec);
}

HI_MPI_VI_DisableBayerRead(ViDev);
/* release raw data memory */
VI_COMM_FreeRawMemInfo(&stRawDataInfo);
...
```

[See Also]

- [HI_MPI_VI_EnableBayerRead](#)
- [HI_MPI_VI_DisableBayerRead](#)

HI_MPI_VI_SendBayerDataEx

[Description]

Send multi-channel raw data to the ISP. This MPI is an extended MPI of [HI_MPI_VI_SendBayerData](#).

[Syntax]

```
HI_S32 HI_MPI_VI_SendBayerDataEx(VI_DEV ViDev, const VI_RAW_DATA_INFO_S
*pstRawData[], HI_U32 u32DataNum, HI_S32 s32MilliSec);
```

[Parameter]

Parameter	Description	Input/Output
ViDev	VI device ID Value range: [0, VIU_MAX_DEV_NUM)	Input
pstRawData	Pointer to the raw data to be read	Input
u32DataNum	Number of virtual channels	Input
s32MilliSec	Timeout period	Input



[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

None

[Requirement]

- Header files: hi_comm_vi.h and mpi_vi.h
- Library file: libmpi.a

[Note]

- The front-end timing is required for reading raw data. Therefore, ensure that the timing of raw data to be read is consistent with the front-end timing.
- This MPI can be called only after [HI_MPI_VI_EnableBayerRead](#) is called. Otherwise, HI_ERR_VI_FAILED_NOTENABLE is returned.
- When multi-channel raw data is transmitted, the raw data is stored in sequence in **pstRawData**. For example, if 2-channel raw data is transmitted, **pstRawData[0]** stores the data of the first channel, and **pstRawData[1]** stores the data of the second channel.
- It is recommended that the ISP be frozen in the scenario where the raw data is sent. Otherwise, the effect may be poor.
- In the scenario where multi-channel raw data is sent, it is recommended that the number of bits of the raw data be greater than or equal to the number of bits of the sensor data. Otherwise, the picture effect may not be as expected.

[Example]

None

[See Also]

- [HI_MPI_VI_EnableBayerRead](#)
- [HI_MPI_VI_DisableBayerRead](#)
- [HI_MPI_VI_SendBayerData](#)

HI_MPI_VI_DisableBayerRead

[Description]

Disables the function of reading raw data.

[Syntax]

```
HI_S32 HI_MPI_VI_DisableBayerRead(VI_DEV ViDev);
```

[Parameter]



Parameter	Description	Input/Output
ViDev	VI device ID Value range: [0, VIU_MAX_DEV_NUM)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

None

[Example]

For details, see [HI_MPI_VI_SetBayerData](#).

[See Also]

- [HI_MPI_VI_EnableBayerRead](#)
- [HI_MPI_VI_SetBayerData](#)

HI_MPI_VI_SetDCIParam

[Description]

Configures DCI parameters.

[Syntax]

```
HI_S32 HI_MPI_VI_SetDCIParam(VI_DEV ViDev, const VI_DCI_PARAM_S  
*pstDciParam);
```

[Parameter]

Parameter	Description	Input/Output
ViDev	VI device ID Value range: [0, VIU_MAX_DEV_NUM)	Input



Parameter	Description	Input/Output
pstDciParam	Pointer to DCI parameters	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

None

[Example]

None

[See Also]

[HI_MPI_VI_GetDCIParam](#)

HI_MPI_VI_GetDCIParam

[Description]

Obtains DCI parameters.

[Syntax]

```
HI_S32 HI_MPI_VI_GetDCIParam(VI_DEV ViDev, VI_DCI_PARAM_S *pstDciParam);
```

[Parameter]

Parameter	Description	Input/Output
ViDev	VI device ID Value range: [0, VIU_MAX_DEV_NUM)	Input
pstDciParam	Pointer to DCI parameters	Output

[Return Value]



Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

None

[Example]

None

[See Also]

[HI_MPI_VI_SetDCIParam](#)

HI_MPI_VI_SetWDRAttr

[Description]

Configures WDR attributes.

[Syntax]

```
HI_S32 HI_MPI_VI_SetWDRAttr(VI_DEV ViDev, const VI_WDR_ATTR_S  
*pstWDRAttr);
```

[Parameter]

Parameter	Description	Input/Output
ViDev	VI device ID Value range: [0, VIU_MAX_DEV_NUM)	Input
pstWDRAttr	Pointer to WDR attributes	Input

[Return Value]

Return Value	Description
0	Success



Return Value	Description
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

Chip	Supported WDR Modes
Hi3516A	WDR_MODE_NONE WDR_MODE_BUILT_IN WDR_MODE_2To1_LINE WDR_MODE_2To1_FRAME WDR_MODE_2To1_FRAME_FULL_RATE
Hi3518E V200	WDR_MODE_NONE WDR_MODE_BUILT_IN
Hi3519 V100	WDR_MODE_NONE WDR_MODE_BUILT_IN WDR_MODE_2To1_LINE WDR_MODE_2To1_FRAME WDR_MODE_2To1_FRAME_FULL_RATE WDR_MODE_3To1_LINE WDR_MODE_3To1_FRAME WDR_MODE_3To1_FRAME_FULL_RATE WDR_MODE_4To1_LINE WDR_MODE_4To1_FRAME WDR_MODE_4To1_FRAME_FULL_RATE

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

- Call `HI_MPI_VI_SetWDRAttr` to configure WDR attributes after calling `HI_MPI_VI_SetDevAttr` and before calling `HI_MPI_VI_EnableDev`, because WDR attributes are obtained from device attributes. In addition, if device attributes are configured again, WDR attributes need to be configured again by calling `HI_MPI_VI_SetWDRAttr`.
- For the Hi3516A, if `s32Y` is not set to `0` in `DevAttr`, the WDR compression function cannot be enabled.
- If the original WDR mode to be switched and the target WDR mode are the same, it is recommended that the upper-layer applications do not reconfigure the MIPI. Otherwise, the image may fail to be captured. It is recommended that the upper-layer applications



determine whether the current WDR mode is the same as the WDR mode to be switched to. If yes, exit and do not perform the switching.

[Example]

None

[See Also]

[HI_MPI_VI_GetWDRAttr](#)

[Description]

Obtains WDR attributes.

[Syntax]

```
HI_S32 HI_MPI_VI_GetWDRAttr(VI_DEV ViDev, VI_WDR_ATTR_S *pstWDRAttr);
```

[Parameter]

Parameter	Description	Input/Output
ViDev	VI device ID Value range: [0, VIU_MAX_DEV_NUM)	Input
pstWDRAttr	Pointer to WDR attributes	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

None

[Example]

None

[See Also]



[HI_MPI_VI_SetWDRAttr](#)

HI_MPI_VI_SetModParam

[Description]

Configures the parameters of the VI module.

[Syntax]

```
HI_S32 HI_MPI_VI_SetModParam(const VI\_MOD\_PARAM\_S *pstModParam);
```

[Parameter]

Parameter	Description	Input/Output
pstModParam	Pointer to the parameters of the VI module	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

- This MPI is called to reconfigure some parameters of the module after the VI module is loaded. If this MPI is not called, use the transferred parameters of the module when the module is loaded or the default values.
- This MPI can be called only before the attributes of a VI device or the extended attributes are configured. Otherwise, an error code is returned.

[Example]

None

[See Also]

[HI_MPI_VI_GetModParam](#)

HI_MPI_VI_GetModParam

[Description]



Obtains the parameters of the VI module.

[Syntax]

```
HI_S32 HI_MPI_VI_GetModParam(VI_MOD_PARAM_S *pstModParam);
```

[Parameter]

Parameter	Description	Input/Output
pstModParam	Pointer to the parameters of the VI module	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

None

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

None

[Example]

None

[See Also]

[HI_MPI_VI_SetModParam](#)

HI_MPI_VI_SetDISConfig

[Description]

Sets the DIS configuration information.

[Syntax]

```
HI_S32 HI_MPI_VI_SetDISConfig(VI_CHN ViChn, const  
VI_DIS_CONFIG_S*pstDISConfig);
```

[Parameter]



Parameter	Description	Input/Output
ViChn	VI channel ID Value range: [0, VIU_MAX_PHYCHN_NUM)	Input
pstDISConfig	Pointer to the DIS configuration information	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

Chip	Supported or Not
Hi3516A	Not supported
Hi3518E V200	Not supported
Hi3519 V100	Supported

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

- The VI channel must be enabled before this MPI is called.
- To start the DIS, you must first call this MPI.
- This MPI cannot be called after the DIS is initialized. Otherwise, the corresponding error code is returned.
- The DIS can be configured only in the physical channel.
- The restrictions on the DIS configuration information are as follows:
 - **enCameraMode** indicates the DIS scenario mode. The **VI_DIS_CAMERA_MODE_IPC** mode is recommended for the IPC surveillance scenario, whereas the **VI_DIS_CAMERA_MODE_NORMAL** mode is recommended for other application scenarios such as the DV scenario.
 - **enMotionType** indicates the multi-axis DOF DIS algorithm. For details, see the *HiMPP DIS User Guide*.
 - The correction parameter **u32RollingShutterCoef** for the rolling shutter is an auxiliary parameter used for DIS processing. The recommended parameter value is 80.



- **u32BufNum** indicates the number of buffers used by DIS to store pictures. At least five buffers are required. If frame loss occurs for the output picture after DIS processing, the number of buffers can be increased. Note that at most eight buffers are allowed.
- **u32CropRatio** indicates the cropping ratio of the DIS output picture. The original picture is cropped based on a certain ratio after DIS processing, and then zoomed in for output. The recommended parameter value is 80, that is, the picture is cropped to 80% of the original picture. When the video input resolution is greater than or equal to 1920 x 1080, the minimum value of **u32CropRatio** is **50**. When the video input resolution is less than 1920 x 1080, the minimum value of **u32CropRatio** is **80**.
- The DIS output frame rate **s32FrameRate** must be the VI output frame rate. The value range is [1, 120].

[Example]

```
HI_S32 s32Ret = HI_SUCCESS;
VI_CHN ViChn = 0;
VI_DIS_CONFIG_S stDISConfig;
VI_DIS_ATTR_S stDISAttr;

/*first mpp sys init & start vi dev & chn*/

stDISConfig.enAccuracy = VI_DIS_ACCURACY_HIGH;
stDISConfig.enCameraMode = VI_DIS_CAMERA_MODE_IPC;
stDISConfig.enMotionType = VI_DIS_MOTION_4DOF_SOFT;
stDISConfig.u32FixLevel = 5;
stDISConfig.u32RollingShutterCoef = 80;
stDISConfig.u32BufNum = 6;
stDISConfig.u32CropRatio = 80;
stDISConfig.s32FrameRate = 30;
stDISConfig.bScale = HI_TRUE;
stDISConfig.u32DelayFrmNum = 0;
stDISConfig.u32RetCenterStrength = 7;
stDISConfig.u32GyroWeight = 0;

s32Ret = HI_MPI_VI_SetDISConfig(ViChn, &stDISConfig);
if (HI_SUCCESS != s32Ret)
{
    printf("SetDISConfig failed with %#x!\n", s32Ret);
    return s32Ret;
}
s32Ret = HI_MPI_VI_DISInit(ViChn);
if (HI_SUCCESS != s32Ret)
{
    printf("failed with %#x!\n", s32Ret);
    return s32Ret;
}
```



```
/* setDISAttr*/  
  
stDISAttr.bEnable = HI_TRUE;  
stDISAttr.u32MovingSubjectLevel = 1;  
stDISAttr.u32NoMovementLevel = 0;  
stDISAttr.u32TimeLag = 0;  
stDISAttr.enAngleType = VI_DIS_ANGLE_TYPE_DIAGONAL;  
stDISAttr.u32Vangle = 500;  
stDISAttr.bStillCrop = HI_FALSE;  
  
s32Ret = HI_MPI_VI_SetDISAttr(ViChn, &stDISAttr);  
if (HI_SUCCESS != s32Ret)  
{  
    printf("failed with %#x!\n", s32Ret);  
    return s32Ret;  
}
```

[See Also]

[HI_MPI_VI_GetDISConfig](#)

HI_MPI_VI_GetDISConfig

[Description]

Obtains the DIS configuration information.

[Syntax]

```
HI_S32 HI_MPI_VI_GetDISConfig(VI_CHN ViChn, VI_DIS_CONFIG_S*pstDISConfig);
```

[Parameter]

Parameter	Description	Input/Output
ViChn	VI channel ID Value range: [0, VIU_MAX_PHYCHN_NUM)	Input
pstDISConfig	Pointer to the DIS configuration information	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]



Chip	Supported or Not
Hi3516A	Not supported
Hi3518E V200	Not supported
Hi3519 V100	Supported

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library file: libmpi.a

[Note]

The DIS configuration information can be obtained only after it is set. If this MPI is called before the DIS configuration information is set, the corresponding error code is returned.

[Example]

None

[See Also]

[HI_MPI_VI_SetDISConfig](#)

HI_MPI_VI_DISInit

[Description]

Initializes the DIS.

[Syntax]

HI_S32 HI_MPI_VI_DISInit(VI_CHN ViChn);

[Parameter]

Parameter	Description	Input/Output
ViChn	VI channel ID Value range: [0, VIU_MAX_PHYCHN_NUM)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]



Chip	Supported or Not
Hi3516A	Not supported
Hi3518E V200	Not supported
Hi3519 V100	Supported

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library files: libmpi.a, libhidis.so

[Note]

The DIS can be initialized only after the DIS configuration information is set. If this MPI is called before the DIS configuration information is set, the corresponding error code is returned.

[Example]

See the example of [HI_MPI_VI_SetDISConfig](#).

[See Also]

[HI_MPI_VI_SetDISConfig](#)

HI_MPI_VI_SetDISAttr

[Description]

Sets the DIS attributes.

[Syntax]

```
HI_S32 HI_MPI_VI_SetDISAttr(VI_CHN ViChn, const VI_DIS_ATTR_S*pstDISAttr);
```

[Parameter]

Parameter	Description	Input/Output
ViChn	VI channel ID Value range: [0, VIU_MAX_PHYCHN_NUM)	Input
pstDISAttr	Pointer to the DIS attributes	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."



[Difference]

Chip	Supported or Not
Hi3516A	Not supported
Hi3518E V200	Not supported
Hi3519 V100	Supported

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library files: libmpi.a, libhidis.so

[Note]

- The DIS attributes can be set only after the DIS is initialized.
- This MPI allows the DIS to be dynamically enabled and disabled, values of **u32MovingSubjectLevel**, **u32NoMovementLevel**, **u32Timelag**, **enAngleType**, and **u32Vangle** to be dynamically changed, and the switch attribute of **bStillCrop** to be dynamically modified.

[Example]

See the example of [HI_MPI_VI_SetDISConfig](#).

[See Also]

- [HI_MPI_VI_DISInit](#)
- [HI_MPI_VI_GetDISAttr](#)

HI_MPI_VI_GetDISAttr

[Description]

Obtains the DIS attributes.

[Syntax]

```
HI_S32 HI_MPI_VI_GetDISAttr(VI_CHN ViChn, VI_DIS_ATTR_S*pstDISAttr);
```

[Parameter]

Parameter	Description	Input/Output
ViChn	VI channel ID Value range: [0, VIU_MAX_PHYCHN_NUM)	Input
pstDISAttr	Pointer to the DIS attributes	Output

[Return Value]



Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

Chip	Supported or Not
Hi3516A	Not supported
Hi3518E V200	Not supported
Hi3519 V100	Supported

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library files: libmpi.a, libhidis.so

[Note]

This MPI can be called only after the DIS attributes are set. Otherwise, an error code is returned.

[Example]

None

[See Also]

[HI_MPI_VI_SetDISAttr](#)

HI_MPI_VI_DISRun

[Description]

Runs the DIS.

[Syntax]

```
HI_S32 HI_MPI_VI_DISRun(VI_CHN ViChn);
```

[Parameter]

Parameter	Description	Input/Output
ViChn	VI channel ID Value range: [0, VIU_MAX_PHYCHN_NUM)	Input

[Return Value]



Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

Chip	Supported or Not
Hi3516A	Not supported
Hi3518E V200	Not supported
Hi3519 V100	Supported

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library files: libmpi.a, libhidis.so

[Note]

- This MPI can be called only after the DIS is initialized and the DIS attribute **bEnable** is set to **HI_TRUE**. Otherwise, an error code is returned.
- This MPI does not support multiple processes, and cannot be called repeatedly. In addition, this MPI must be in the same process as [HI_MPI_VI_SetDISConfig](#), [HI_MPI_VI_DISInit](#), [HI_MPI_VI_SetDISAttr](#), and [HI_MPI_VI_DISExit](#).
- This MPI is a block interface. You are advised to call it by using a real-time thread.

[Example]

None

[See Also]

- [HI_MPI_VI_SetDISAttr](#)
- [HI_MPI_VI_DISInit](#)

HI_MPI_VI_DISExit

[Description]

Exits the DIS.

[Syntax]

```
HI_S32 HI_MPI_VI_DISExit(VI_CHN ViChn);
```

[Parameter]



Parameter	Description	Input/Output
ViChn	VI channel ID Value range: [0, VIU_MAX_PHYCHN_NUM)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

Chip	Supported or Not
Hi3516A	Not supported
Hi3518E V200	Not supported
Hi3519 V100	Supported

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library files: libmpi.a, libhidis.so

[Note]

- You need to call [HI_MPI_VI_DISInit](#) and [HI_MPI_VI_DISRun](#) before calling this MPI to exit the DIS.
- This MPI does not support multiple processes, and must be called in the same process as [HI_MPI_VI_SetDISConfig](#), [HI_MPI_VI_DISInit](#), [HI_MPI_VI_SetDISAttr](#), and [HI_MPI_VI_DISRun](#).
- This MPI cannot be called repeatedly.
- When exceptions occur and the DIS is exited by pressing **Ctrl+C**, this MPI needs to be called during exception processing and the DIS running thread needs to be stopped.

[Example]

None

[See Also]

- [HI_MPI_VI_DISInit](#)
- [HI_MPI_VI_DISRun](#)



HI_MPI_VI_RegisterDISCallback

[Description]

Registers the callback functions for obtaining and releasing the gyroscope data with the DIS.

[Syntax]

```
HI_S32 HI_MPI_VI_RegisterDISCallback(VI_CHN ViChn,  
VI_DIS_CALLBACK_S*pstDISCallback);
```

[Parameter]

Parameter	Description	Input/Output
ViChn	VI channel ID Value range: [0, VIU_MAX_PHYCHN_NUM)	Input
pstDISCallback	Pointer to the structure of the DIS callback function	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

Chip	Supported or Not
Hi3516A	Not supported
Hi3518E V200	Not supported
Hi3519 V100	Supported

[Requirement]

- Header files: hi_comm_vi.h, mpi_vi.h
- Library files: libmpi.a, libhidis.so

[Note]

- This MPI can be called to register the callback functions only after [HI_MPI_VI_DISInit](#) is called and before [HI_MPI_VI_DISRun](#) is called.
- The callback functions for obtaining and releasing the gyroscope data must be registered at the same time. Otherwise, an error code is returned.



- This MPI does not support multiple processes, and must be called in the same process as [HI_MPI_VI_SetDISConfig](#), [HI_MPI_VI_DISInit](#), [HI_MPI_VI_SetDISAttr](#), and [HI_MPI_VI_DISRun](#).

[Example]

None

[See Also]

[HI_MPI_VI_UnRegisterDISCallback](#)

HI_MPI_VI_UnRegisterDISCallback

[Description]

Deregisters the callback functions for obtaining and releasing the gyroscope data from the DIS.

[Syntax]

`HI_S32 HI_MPI_VI_UnRegisterDISCallback(VI_CHN ViChn);`

[Parameter]

Parameter	Description	Input/Output
ViChn	VI channel ID Value range: [0, VIU_MAX_PHYCHN_NUM)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 3.6 "Error Codes."

[Difference]

Chip	Supported or Not
Hi3516A	Not supported
Hi3518E V200	Not supported
Hi3519 V100	Supported

[Requirement]

- Header files: `hi_comm_vi.h`, `mpi_vi.h`
- Library files: `libmpi.a`, `libhidis.so`



[Note]

- This MPI can be called only after [HI_MPI_VI_RegisterDISCallback](#) is called and before [HI_MPI_VI_DISExit](#) is called. Otherwise, an error code is returned.
- This MPI does not support multiple processes, and must be called in the same process as [HI_MPI_VI_SetDISConfig](#), [HI_MPI_VI_DisInit](#), [HI_MPI_VI_SetDISAttr](#), [HI_MPI_VI_RegisterDISCallback](#), and [HI_MPI_VI_DISRun](#).
- This MPI cannot be called repeatedly.

[Example]

None

[See Also]

[HI_MPI_VI_RegisterDISCallback](#)

3.5 Data Structures

The VI data types are as follows:

- [VIU_MAX_RAWCHN_NUM](#): Defines the maximum number of channels for obtaining raw data.
- [VIU_GET_RAW_CHN](#): Defines the ID of channel for obtaining raw data.
- [VIU_MAX_UFLIST_NUM](#): Defines the maximum number of user buffer queues.
- [VIU_MAX_DEV_NUM](#): Defines the maximum number of VI device ports.
- [VIU_MAX_WAY_NUM_PER_DEV](#): Defines the maximum number of data ways for a VI device.
- [VIU_MAX_PHYCHN_NUM](#): Defines the maximum number of physical VI channels.
- [VIU_MAX_CHN_NUM](#): Defines the maximum number of VI channels.
- [VIU_MAX_CHN_NUM](#): Defines the maximum number of VI channels.
- [VIU_DEV_MIN_WIDTH](#): Defines the minimum width of the pictures captured by a VI device.
- [VIU_DEV_MIN_HEIGHT](#): Defines the minimum height of the pictures captured by a VI device.
- [VIU_DEV_MAX_WIDTH](#): Defines the maximum width of the pictures captured by a VI device.
- [VIU_DEV_MAX_HEIGHT](#): Defines the maximum height of the pictures captured by a VI device.
- [VIU_CHN_MIN_WIDTH](#): Defines the minimum width supported by a physical VI channel.
- [VIU_CHN_MIN_HEIGHT](#): Defines the minimum height supported by a physical VI channel.
- [VIU_CHN_MAX_WIDTH](#): Defines the maximum width supported by a physical VI channel.
- [VIU_CHN_MAX_HEIGHT](#): Defines the maximum height supported by a physical VI channel.
- [VIU_EXTCHN_MIN_WIDTH](#): Defines the minimum width supported by an extended VI channel.



- **VIU_EXTCHN_MIN_HEIGHT**: Defines the minimum height supported by an extended VI channel.
- **VIU_EXTCHN_MAX_WIDTH**: Defines the maximum width supported by an extended VI channel.
- **VIU_EXTCHN_MAX_HEIGHT**: Defines the maximum height supported by an extended VI channel.
- **VIU_EXTCHN_MINIFICATION**: Defines the maximum scaling ratio of an extended VI channel.
- **VIU_DEV_COMP_MASK_NUM**: Defines the number of masks for the VI device component.
- **VIU_DEV_AD_CHN_NUM**: Defines the number of ADs for the VI device, which is meaningless for the IPC.
- **VIU_CSC_IDC_NUM**: Defines the number of input DC components for the CSC.
- **VIU_CSC_ODC_NUM**: Defines the number of output DC components for the CSC.
- **VIU_CSC_COEF_NUM**: Defines the conversion coefficient matrix of the CSC.
- **VI_INTF_MODE_E**: Defines the interface mode of a VI device.
- **VI_INPUT_MODE_E**: Defines the input mode of a VI device.
- **VI_COMBINE_MODE_E**: Defines the mode of the data received by a VI device.
- **VI_COMP_MODE_E**: Defines the component type of the data received by a VI device.
- **VI_CLK_EDGE_E**: Defines the type of the clock received by a VI device.
- **VI_SCAN_MODE_E**: Defines whether interlaced or progressive input pictures are received by a VI device.
- **VI_DATA_YUV_SEQ_E**: Defines the arrangement sequence of the YUV data received by a VI device.
- **VI_VSYNC_E**: Defines the vertical sync signal type of the input data of a VI device.
- **VI_VSYNC_NEG_E**: Defines the vertical sync signal polarity of the input data of a VI device.
- **VI_HSYNC_E**: Defines the horizontal sync signal type of the input data of a VI device.
- **VI_HSYNC_NEG_E**: Defines the horizontal sync signal polarity of the input data of a VI device.
- **VI_VSYNC_VALID_E**: Defines the vertical valid sync signal type of the input data of a VI device.
- **VI_VSYNC_VALID_NEG_E**: Defines the vertical valid sync signal polarity of the input data of a VI device.
- **VI_SYNC_CFG_S**: Defines the sync information about the BT.601 timing or DC timing received by a VI device.
- **VI_BT656_SYNC_CFG_S**: Defines the sync information about the BT.656 timing received by a VI device.
- **BT656_FIXCODE_E**: Defines the most significant bit (MSB) of the BT.656 timing reference code.
- **BT656_FIELD_POLAR_E**: Defines the polarity of the field indication bit (F) of the BT.656 timing reference code.
- **VI_DEV_ATTR_S**: Defines the attributes of a VI device.
- **VI_CAPSEL_E**: Defines the frame/field selection of the captured VI pictures.
- **VI_CHN_ATTR_S**: Defines the attributes of a VI channel.
- **VI_USERPIC_MODE_E**: Defines the user picture mode.



- [VI_USERPIC_BGC_S](#): Defines the information about a background picture only with a color.
- [VI_USERPIC_ATTR_S](#): Defines the information about a user picture.
- [VI_CHN_STAT_S](#): Defines the structure of VI channel information.
- [VI_DATA_PATH_E](#): Defines the VI data channel enumeration.
- [VI_EXT_CHN_ATTR_S](#): Defines the attributes of an extended VI channel.
- [LDC_VIEW_TYPE_E](#): Defines the LDC mode.
- [LDC_ATTR_S](#): Defines the LDC attribute.
- [VI_LDC_ATTR_S](#): Defines the VI LDC attribute.
- [VI_CSC_TYPE_E](#): Defines the YUV CSC standard.
- [VI_CSC_MATRIX_S](#): Defines the coefficients of the CSC matrix.
- [VI_CSC_ATTR_S](#): Defines the CSC attribute.
- [VI_FLASH_MODE_E](#): Defines the camera flash mode.
- [VI_FLASH_CONFIG_S](#): Defines camera flash settings.
- [VI_CHN_LUM_S](#): Defines the channel luminance statistics.
- [VI_RAW_DATA_INFO_S](#): Defines raw data attributes.
- [VI_DCI_PARAM_S](#): Defines DCI parameters.
- [VI_DUMP_ATTR_S](#): Defines the attribute of the data to be dumped.
- [VI_DUMP_TYPE_E](#): Defines the type of the data to be dumped for the VI device.
- [VI_WDR_ATTR_S](#): Defines the WDR attributes.
- [VI_MOD_PARAM_S](#): Defines the parameters of the VI module.
- [VI_DIS_CONFIG_S](#): Defines the DIS configuration information.
- [VI_DIS_ATTR_S](#): Defines the DIS attributes.
- [VI_DIS_ANGLE_TYPE_E](#): Defines the type of the lens view angle used in DIS.
- [VI_DIS_ACCURACY_E](#): Defines the DIS accuracy.
- [VI_DIS_CAMERA_MODE_E](#): Defines the DIS scenario mode.
- [VI_DIS_MOTION_TYPE_E](#): Defines the multi-DOF DIS algorithm.
- [VI_DIS_GYRO_DATA_S](#): Defines the structure of the gyroscope data used by the DIS.
- [VI_DIS_CALLBACK_S](#): Defines the structure that registers the callback functions with the DIS.

VIU_MAX_RAWCHN_NUM

[Description]

Defines the maximum number of channels for obtaining raw data.

[Syntax]

For Hi3516A/Hi3518E V200/Hi3519 V100:

```
#define VIU_MAX_RAWCHN_NUM           1
```

[Note]

A channel is used to obtain raw data. Its ID is obtained by using [VIU_GET_RAW_CHN](#).

[See Also]



- [HI_MPI_VI_EnableBayerDump](#)
- [HI_MPI_VI_GetFrame](#)

VIU_GET_RAW_CHN

[Description]

Defines the ID of channel for obtaining raw data.

[Syntax]

```
#define VIU_GET_RAW_CHN(ViDev, RawChn) \
do{ \
    RawChn = VIU_MAX_CHN_NUM + ViDev; \
}while(0)
```

[Note]

None

[See Also]

[HI_MPI_VI_GetFrame](#)

VIU_MAX_UFLIST_NUM

[Description]

Defines the maximum number of user buffer queues.

[Syntax]

```
#define VIU_MAX_UFLIST_NUM      (VIU_MAX_CHN_NUM + VIU_MAX_RAWCHN_NUM)
```

[Note]

None

[See Also]

None

VIU_MAX_DEV_NUM

[Description]

Defines the maximum number of VI device ports.

[Syntax]

For Hi3516A/Hi3518E V200/Hi3519 V100:

```
#define VIU_MAX_DEV_NUM      1
```

[Note]

None

[See Also]



None

VIU_MAX_WAY_NUM_PER_DEV

[Description]

Defines the maximum number of data ways for a VI device.

[Syntax]

For Hi3516A/Hi3518E V200/Hi3519 V100:

```
#define VIU_MAX_WAY_NUM_PER_DEV      1
```

[Note]

None

[See Also]

None

VIU_MAX_PHYCHN_NUM

[Description]

Defines the maximum number of physical VI channels.

[Syntax]

For Hi3516A/Hi3518E V200/Hi3519 V100:

```
#define VIU_MAX_PHYCHN_NUM      1
```

[Note]

None

[See Also]

None

VIU_EXT_CHN_START

[Description]

Defines the initial value of the extended VI channel ID.

[Syntax]

```
#define VIU_EXT_CHN_START      VIU_MAX_PHYCHN_NUM
```

[Note]

None

[See Also]

None



VIU_MAX_EXT_CHN_NUM

[Description]

Defines the maximum number of extended VI channels.

[Syntax]

For Hi3516A/Hi3518E V200/Hi3519 V100:

```
#define VIU_MAX_EXT_CHN_NUM           16
```

[Note]

None

[See Also]

None

VIU_MAX_EXTCHN_BIND_PER_CHN

[Description]

Defines the maximum number of extended channels bound to a VI channel.

[Syntax]

For Hi3516A/Hi3518E V200/Hi3519 V100:

```
#define VIU_MAX_EXTCHN_BIND_PER_CHN     8
```

[Note]

None

[See Also]

None

VIU_MAX_CHN_NUM

[Description]

Defines the maximum number of VI channels.

[Syntax]

For Hi3516A/Hi3518E V200/Hi3519 V100:

```
#define VIU_MAX_CHN_NUM (VIU_MAX_PHYCHN_NUM + VIU_MAX_EXT_CHN_NUM)
```

[Note]

For Hi3516A/Hi3518E V200/Hi3519 V100:

- Chn0 is a physical channel.
- Chn1 to chn16 are extended channels.

[See Also]

None



VIU_DEV_MIN_WIDTH

[Description]

Defines the minimum width of the pictures captured by a VI device.

[Syntax]

```
#define VIU_DEV_MIN_WIDTH      64
```

[Note]

None

[See Also]

[HI_MPI_VI_SetDevAttr](#)

VIU_DEV_MIN_HEIGHT

[Description]

Defines the minimum height of the pictures captured by a VI device.

[Syntax]

```
#define VIU_DEV_MIN_HEIGHT      64
```

[Note]

None

[See Also]

[HI_MPI_VI_SetDevAttr](#)

VIU_DEV_MAX_WIDTH

[Description]

Defines the maximum width of the pictures captured by a VI device.

[Difference]

Chip Type	Syntax
Hi3516A	<pre>#define VIU_DEV_MAX_WIDTH 2592</pre> <p>Note: The line buffer of the chip supports at most 2592 pixels.</p>
Hi3518E V200	<pre>#define VIU_DEV_MAX_WIDTH 2048</pre> <p>Note: The line buffer of the chip supports at most 2048 pixels.</p>
Hi3519 V100	<pre>#define VIU_DEV_MAX_WIDTH 4608</pre> <p>Note: The line buffer of the chip supports at most 4608 pixels.</p>

[Note]

None



[See Also]

[HI_MPI_VI_SetDevAttr](#)

VIU_DEV_MAX_HEIGHT

[Description]

Defines the maximum height of the pictures captured by a VI device.

[Difference]

Chip Type	Syntax
Hi3516A	#define VIU_DEV_MAX_HEIGHT 2200 Note: The resolution supported by the Hi3516A is 2592 x 1944. Considering that the DIS function requires shift of 128 pixels in the vertical direction, you need to increase the input height to 2200.
Hi3518E V200	#define VIU_DEV_MAX_HEIGHT 1208 Note: The resolution supported by the Hi3518E V200 is 1920 x 1080. Considering that the DIS function requires shift of 64 pixels in the vertical direction, you need to increase the input height to 1208.
Hi3519 V100	#define VIU_DEV_MAX_HEIGHT 4608

[Note]

None

[See Also]

[HI_MPI_VI_SetDevAttr](#)

VIU_CHN_MIN_WIDTH

[Description]

Defines the minimum width supported by a physical VI channel.

[Syntax]

```
#define VIU_CHN_MIN_WIDTH 64
```

[Note]

None

[See Also]

[HI_MPI_VI_SetChnAttr](#)

VIU_CHN_MIN_HEIGHT

[Description]



Defines the minimum height supported by a physical VI channel.

[Syntax]

```
#define VIU_CHN_MIN_HEIGHT      64
```

[Note]

None

[See Also]

[HI_MPI_VI_SetChnAttr](#)

VIU_CHN_MAX_WIDTH

[Description]

Defines the maximum width supported by a physical VI channel.

[Syntax]

```
#define VIU_CHN_MAX_WIDTH      VIU_DEV_MAX_WIDTH
```

[Note]

None

[See Also]

- [HI_MPI_VI_SetChnAttr](#)
- [VIU_DEV_MAX_WIDTH](#)

VIU_CHN_MAX_HEIGHT

[Description]

Defines the maximum height supported by a physical VI channel.

[Syntax]

```
#define VIU_CHN_MAX_HEIGHT      VIU_DEV_MAX_HEIGHT
```

[Note]

None

[See Also]

- [HI_MPI_VI_SetChnAttr](#)
- [VIU_DEV_MAX_HEIGHT](#)

VIU_EXTCHN_MIN_WIDTH

[Description]

Defines the minimum width supported by an extended VI channel.

[Syntax]

```
#define VIU_EXTCHN_MIN_WIDTH    64
```



[Note]

None

[See Also]

[HI_MPI_VI_SetExtChnAttr](#)

VIU_EXTCHN_MIN_HEIGHT

[Description]

Defines the minimum height supported by an extended VI channel.

[Syntax]

```
#define VIU_EXTCHN_MIN_HEIGHT 64
```

[Note]

None

[See Also]

[HI_MPI_VI_SetExtChnAttr](#)

VIU_EXTCHN_MAX_WIDTH

[Description]

Defines the maximum width supported by an extended VI channel.

[Difference]

Chip Type	Syntax
Hi3516A	#define VIU_EXTCHN_MAX_WIDTH 2592
Hi3518E V200	#define VIU_EXTCHN_MAX_WIDTH 2048
Hi3519 V100	#define VIU_EXTCHN_MAX_WIDTH 4608

[Note]

None

[See Also]

[HI_MPI_VI_SetExtChnAttr](#)

VIU_EXTCHN_MAX_HEIGHT

[Description]

Defines the maximum height supported by an extended VI channel.

[Difference]



Chip Type	Syntax
Hi3516A	#define VIU_EXTCHN_MAX_HEIGHT 2592
Hi3518E V200	#define VIU_EXTCHN_MAX_HEIGHT 2048
Hi3519 V100	#define VIU_EXTCHN_MAX_HEIGHT 4096

[Note]

None

[See Also]

[HI_MPI_VI_SetExtChnAttr](#)

VIU_EXTCHN_MINIFICATION

[Description]

Defines the maximum scaling ratio of an extended VI channel.

[Syntax]

#define VIU_EXTCHN_MINIFICATION 13

[Note]

None

[See Also]

[HI_MPI_VI_SetExtChnAttr](#)

VIU_DEV_COMP_MASK_NUM

[Description]

Defines the number of masks for the VI device component.

[Syntax]

#define VIU_DEV_COMP_MASK_NUM 2

[Note]

None

[See Also]

- [HI_MPI_VI_SetDevAttrEx](#)
- [HI_MPI_VI_SetDevAttr](#)

VIU_DEV_AD_CHN_NUM

[Description]

Defines the number of ADs for the VI device, which is meaningless for the IPC.



[Syntax]

```
#define VIU_DEV_AD_CHN_NUM      4
```

[Note]

None

[See Also]

- [HI_MPI_VI_SetDevAttrEx](#)
- [HI_MPI_VI_SetDevAttr](#)

VIU_CSC_IDC_NUM

[Description]

Defines the number of input DC components for the CSC.

[Syntax]

```
#define VIU_CSC_IDC_NUM      3
```

[Note]

None

[See Also]

- [VI_CSC_ATTR_S](#)
- [VI_CSC_TYPE_E](#)
- [HI_MPI_VI_SetCSCAttr](#)
- [HI_MPI_VI_GetCSCAttr](#)

VIU_CSC_ODC_NUM

[Description]

Defines the number of output DC components for the CSC.

[Syntax]

```
#define VIU_CSC_ODC_NUM      3
```

[Note]

None

[See Also]

- [VI_CSC_ATTR_S](#)
- [VI_CSC_TYPE_E](#)
- [HI_MPI_VI_SetCSCAttr](#)
- [HI_MPI_VI_GetCSCAttr](#)

VIU_CSC_COEF_NUM

[Description]



Defines the conversion coefficient matrix of the CSC.

[Syntax]

```
#define VIU_CSC_COEF_NUM 9
```

[Note]

None

[See Also]

- [VI_CSC_ATTR_S](#)
- [VI_CSC_TYPE_E](#)
- [HI_MPI_VI_SetCSCAttr](#)
- [HI_MPI_VI_GetCSCAttr](#)

VI_INTF_MODE_E

[Description]

Defines the interface mode of a VI device.

[Syntax]

```
typedef enum hiVI_INTF_MODE_E
{
    VI_MODE_BT656 = 0,
    VI_MODE_BT601,
    VI_MODE_DIGITAL_CAMERA,
    VI_MODE_BT1120_STANDARD,
    VI_MODE_BT1120_INTERLEAVED,
    VI_MODE_MIPI,
    VI_MODE_LVDS,
    VI_MODE_HISPI,
    VI_MODE_BUTT
} VI_INTF_MODE_E;
```

[Member]

Member	Description
VI_MODE_BT656	The data input protocol is the standard BT.656 protocol, the port data input mode is the Y/C composite mode, and the component mode is the single-component mode.
VI_MODE_BT601	The data input protocol is the standard BT.601 protocol, the port data input mode is the Y/C composite mode, and the component mode is the single-component mode.
VI_MODE_DIGITAL_CAMERA	The data input protocol is the DC protocol, the port data input mode is the Y/C composite mode, and the component mode is the single-



Member	Description
	component mode.
VI_MODE_BT1120_STANDARD	The data input protocol is the standard BT.1120 protocol (BT.656+dual components), the port data input mode is the Y/C separation mode, and the component mode is the dual-component mode.
VI_MODE_BT1120_INTERLEAVED	The data input protocol is the BT.1120 interleave protocol, the port data input mode is the Y/C separation mode, and the component mode is the dual-component mode.
VI_MODE_MIPI	The data input protocol is the MIPI protocol.
VI_MODE_LVDS	The data input protocol is the LVDS protocol.
VI_MODE_HISPI	The data input protocol is the HiSPi protocol.

[Note]

The interface mode is VI_MODE_DIGITAL_CAMERA for the sensor.

[See Also]

- [VI_DEV_ATTR_S](#)
- [HI_MPI_VI_SetDevAttr](#)

VI_INPUT_MODE_E

[Description]

Defines the input mode of a VI device.

[Syntax]

```
typedef enum hiVI_INPUT_MODE_E
{
    VI_INPUT_MODE_BT656 = 0,
    VI_INPUT_MODE_BT601,
    VI_INPUT_MODE_DIGITAL_CAMERA,
    VI_INPUT_MODE_INTERLEAVED,
    VI_INPUT_MODE_MIPI,
    VI_INPUT_MODE_LVDS,
    VI_INPUT_MODE_HISPI,
    VI_INPUT_MODE_BUTT
} VI_INPUT_MODE_E;
```

[Member]



Member	Description
VI_INPUT_MODE_BT656	The data input protocol is the standard BT.656 protocol.
VI_INPUT_MODE_BT601	The data input protocol is the standard BT.601 protocol.
VI_INPUT_MODE_DIGITAL_CAMERA	The data input protocol is the DC protocol.
VI_INPUT_MODE_INTERLEAVED	The data input protocol is the BT.1120 interleave protocol.
VI_INPUT_MODE_MIPI	The data input protocol is the MIPI protocol.
VI_INPUT_MODE_LVDS	The data input protocol is the LVDS protocol.
VI_INPUT_MODE_HISPI	The data input protocol is the HiSPi protocol.

[Note]

The input mode is VI_INPUT_MODE_DIGITAL_CAMERA for the sensor. If the device input complies with the standard BT.1120 protocol, the input mode is set to VI_INPUT_MODE_BT656 and [VI_COMP_MODE_E](#) is set to the dual-component mode. If the device input complies with the BT.1120 interleave protocol, the input mode is set to VI_INPUT_MODE_INTERLEAVED.

[See Also]

- [VI_DEV_ATTR_EX_S](#)
- [HI_MPI_VI_SetDevAttrEx](#)

VI_WORK_MODE_E

[Description]

Defines the multiplexed working mode of a VI device.

[Syntax]

```
typedef enum hIVI_WORK_MODE_E
{
    VI_WORK_MODE_1Multiplex = 0,
    VI_WORK_MODE_2Multiplex,
    VI_WORK_MODE_4Multiplex,
    VI_WORK_MODE_BUTT
} VI_WORK_MODE_E;
```

[Member]

Member	Description
VI_WORK_MODE_1Multiplex	1-channel multiplexing mode
VI_WORK_MODE_2Multiplex	2-channel multiplexing mode. The data input



Member	Description
	protocol must be the standard BT.656 protocol.
VI_WORK_MODE_4Multiplex	4-channel multiplexing mode. The data input protocol must be the standard BT.656 protocol.

[Note]

- When the multiplexing mode is set to 2-channel multiplexing mode or 4-channel multiplexing mode, the device input protocol must be the BT.656 protocol. There is no such limitation in 1-channel multiplexing mode.
- The Hi3516A/Hi3518E V200 can work only in 1-channel multiplexing mode.

[See Also]

- [VI_DEV_ATTR_S](#)
- [HI_MPI_VI_SetDevAttr](#)

VI_SCAN_MODE_E

[Description]

Defines whether interlaced or progressive input pictures are received by a VI device.

[Syntax]

```
typedef enum hiVI_SCAN_MODE_E
{
    VI_SCAN_INTERLACED = 0, /*Interlaced*/
    VI_SCAN_PROGRESSIVE, /*Progressive*/
    VI_SCAN_BUTT,
} VI_SCAN_MODE_E;
```

[Member]

Member	Description
VI_SCAN_INTERLACED	Interlaced picture
VI_SCAN_PROGRESSIVE	Progressive picture

[Note]

None

[See Also]

- [VI_DEV_ATTR_S](#)
- [HI_MPI_VI_SetDevAttrEx](#)



VI_DATA_YUV_SEQ_E

[Description]

Defines the arrangement sequence of the YUV data received by a VI device.

[Syntax]

```
typedef enum hiVI_DATA_YUV_SEQ_E
{
    VI_INPUT_DATA_VUVU = 0,
    VI_INPUT_DATA_UVUV,
    VI_INPUT_DATA_UYVY = 0,
    VI_INPUT_DATA_VYUY,
    VI_INPUT_DATA_YUYV,
    VI_INPUT_DATA_YVYU,
    VI_DATA_YUV_BUTT
} VI_DATA_YUV_SEQ_E;
```

[Member]

Member	Description
VI_INPUT_DATA_VUVU	When the YUV data is input in separation mode, input sequence of the C component is VUVU.
VI_INPUT_DATA_UVUV	When the YUV data is input in separation mode, input sequence of the C component is UVUV.
VI_INPUT_DATA_UYVY	When the YUV data is input in composite mode, the data input sequence is UYVY.
VI_INPUT_DATA_VYUY	When the YUV data is input in composite mode, the data input sequence is VYUY.
VI_INPUT_DATA_YUYV	When the YUV data is input in composite mode, the data input sequence is YUYV.
VI_INPUT_DATA_YVYU	When the YUV data is input in composite mode, the data input sequence is YVYU.

[Note]

None

[See Also]

- [VI_DEV_ATTR_S](#)
- [HI_MPI_VI_SetDevAttr](#)
- [VI_DEV_ATTR_EX_S](#)
- [HI_MPI_VI_SetDevAttrEx](#)



VI_CLK_EDGE_E

[Description]

Defines the type of the clock received by a VI device.

[Syntax]

```
typedef enum hiVI_CLK_EDGE_E
{
    VI_CLK_EDGE_SINGLE_UP = 0,
    VI_CLK_EDGE_SINGLE_DOWN,
    VI_CLK_EDGE_BUTT
} VI_CLK_EDGE_E;
```

[Member]

Member	Description
VI_CLK_EDGE_SINGLE_UP	Clock single-edge mode. The VI device samples clocks on the rising edge.
VI_CLK_EDGE_SINGLE_DOWN	Clock single-edge mode. The VI device samples clocks on the falling edge.

[Note]

None

[See Also]

- [VI_DEV_ATTR_S](#)
- [HI_MPI_VI_SetDevAttrEx](#)

VI_COMP_MODE_E

[Description]

Defines the component type of the data received by a VI device.

[Syntax]

```
typedef enum hiVI_COMP_MODE_E
{
    VI_COMP_MODE_SINGLE = 0,
    VI_COMP_MODE_DOUBLE = 1,
    VI_COMP_MODE_BUTT,
} VI_COMP_MODE_E;
```

[Member]

Member	Description
VI_COMP_MODE_SINGL	The component type of the input data is single-



Member	Description
	component.
VI_COMP_MODE_DOUBLE	The component type of the input data is dual-component.

[Note]

- This enumeration and the component mask au32CompMask[2] of [VI_DEV_ATTR_S](#) or [VI_DEV_ATTR_EX_S](#) must be configured at the same time.
- The VI device Dev0 supports a maximum of 16-bit inputs, and the 16-bit data lines are used to connect to external video sources. In the actual application scenario, only some data lines are used, which depends on hardware connection. For example, the camera connects to the VI channel by using the first 12-bit data lines (data lines corresponding to bit 0 to bit 11). In this case, the component type of the input data is set to single-component, and the component mask is set to **0xFFFF0000**.
- If the video data is input in dual-component mode, the device attribute must be set to dual-component and the component mask of each component must be specified based on the pin connection. For example, if Dev0 is set to BT.1120 dual-component input mode (eight bits for the Y component and C component respectively), the component mask of the Y component is set to 0xFF000000 and the component mask of the C component is set to 0xFF0000.

[See Also]

- [VI_DEV_ATTR_S](#)
- [VI_DEV_ATTR_EX_S](#)
- [HI_MPI_VI_SetDevAttrEx](#)
- [HI_MPI_VI_SetDevAttr](#)

VI_COMBINE_MODE_E

[Description]

Defines the mode of the data received by a VI device.

[Syntax]

```
typedef enum hivi_combine_mode_e
{
    VI_COMBINE_COMPOSITE = 0,
    VI_COMBINE_SEPARATE,
    VI_COMBINE_BUTT,
} VI_COMBINE_MODE_E;
```

[Member]

Member	Description
VI_COMBINE_COMPOSITE	Composite mode
VI_COMBINE_SEPARATE	Separation mode



[Note]

If the Y component and C component are input by using the same data line, the data mode is set to VI_COMBINE_COMPOSITE; if the Y component and C component are input by using different data lines, the data mode is set to VI_COMBINE_SEPARATE such as BT.1120.

[See Also]

- [VI_DEV_ATTR_EX_S](#)
- [HI_MPI_VI_BindChn](#)

VI_VSYNC_E

[Description]

Defines the vertical sync signal type of the input data of a VI device.

[Syntax]

```
typedef enum hivI_VSYNC_E
{
    VI_VSYNC_FIELD = 0,
    VI_VSYNC_PULSE,
} VI_VSYNC_E;
```

[Member]

Member	Description
VI_VSYNC_FIELD	Vertical sync invert mode. A field occurs when the level is inverted once. This member indicates the field number in BT.601 mode and line active signal in DC mode.
VI_VSYNC_PULSE	Vertical sync pulse mode. That is, when a pulse arrives, a new frame or field starts.

[Note]

None

[See Also]

- [VI_VSYNC_NEG_E](#)
- [VI_SYNC_CFG_S](#)

VI_VSYNC_NEG_E

[Description]

Defines the vertical sync signal polarity of the input data of a VI device.

[Syntax]

```
typedef enum hivI_VSYNC_NEG_E
```



```
{  
    VI_VSYNC_NEG_HIGH = 0,  
    VI_VSYNC_NEG_LOW  
} VI_VSYNC_NEG_E;
```

[Member]

Member	Description
VI_VSYNC_NEG_HIGH	If VI_VSYNC_E is set to VI_VSYNC_FIELD , the level of vsync signal of the even field is high. If VI_VSYNC_E is set to VI_VSYNC_PULSE , the positive pulse indicates vsync pulse.
VI_VSYNC_NEG_LOW	If VI_VSYNC_E is set to VI_VSYNC_FIELD , the level of vsync signal of the even field is low. If VI_VSYNC_E is set to VI_VSYNC_PULSE , the negative pulse indicates vsync pulse.

[Note]

None

[See Also]

- [VI_VSYNC_E](#)
- [VI_SYNC_CFG_S](#)

VI_HSYNC_E

[Description]

Defines the horizontal sync signal type of the input data of a VI device.

[Syntax]

```
typedef enum hiVI_HSYNC_E  
{  
    VI_HSYNC_VALID_SIGNAL = 0,  
    VI_HSYNC_PULSE,  
} VI_HSYNC_E;
```

[Member]

Member	Description
VI_HSYNC_VALID_SIGNAL	Horizontal sync data valid signal
VI_HSYNC_PULSE	Horizontal sync pulse signal

[Note]



None

[See Also]

- [VI_HSYNC_NEG_E](#)
- [VI_SYNC_CFG_S](#)

VI_HSYNC_NEG_E

[Description]

Defines the horizontal sync signal polarity of the input data of a VI device.

[Syntax]

```
typedef enum hiVI_HSYNC_NEG_E
{
    VI_HSYNC_NEG_HIGH = 0,
    VI_HSYNC_NEG_LOW
} VI_HSYNC_NEG_E;
```

[Member]

Member	Description
VI_HSYNC_NEG_HIGH	If VI_HSYNC_E is set to VI_HSYNC_VALID_SIGNAL , the high level indicates valid data. If VI_HSYNC_E is set to VI_HSYNC_PULSE , the positive pulse indicates sync pulse.
VI_HSYNC_NEG_LOW	If VI_HSYNC_E is set to VI_HSYNC_VALID_SIGNAL , the low level indicates valid data. If VI_HSYNC_E is set to VI_HSYNC_PULSE , the negative pulse indicates sync pulse.

[Note]

None

[See Also]

- [VI_HSYNC_E](#)
- [VI_SYNC_CFG_S](#)

VI_VSYNC_VALID_E

[Description]

Defines the vertical valid sync signal type of the input data of a VI device.

[Syntax]

```
typedef enum hiVI_VSYNC_VALID_E
{
```



```
VI_VSYNC_NORM_PULSE = 0,  
VI_VSYNC_VALID_SINGAL,  
} VI_VSYNC_VALID_E;
```

[Member]

Member	Description
VI_VSYNC_NORM_PULSE	Vertical valid sync flag. It is used in DC mode.
VI_VSYNC_VALID_SINGAL	Line active signal of the vertical sync timing. It is used in DC mode.

[Note]

This data type must be configured in DC mode.

[See Also]

- [VI_VSYNC_E](#)
- [VI_VSYNC_VALID_NEG_E](#)
- [VI_SYNC_CFG_S](#)

VI_VSYNC_VALID_NEG_E

[Description]

Defines the vertical valid sync signal polarity of the input data of a VI device.

[Syntax]

```
typedef enum hiVI_VSYNC_VALID_NEG_E  
{  
    VI_VSYNC_VALID_NEG_HIGH = 0,  
    VI_VSYNC_VALID_NEG_LOW  
} VI_VSYNC_VALID_NEG_E;
```

[Member]

Member	Description
VI_VSYNC_VALID_NEG_HIGH	If VI_VSYNC_VALID_E is set to VI_VSYNC_NORM_SINGAL , the high level indicates valid signal.
VI_VSYNC_VALID_NEG_LOW	If VI_VSYNC_VALID_E is set to VI_VSYNC_NORM_SINGAL , the low level indicates valid signal.

[Note]

This data type is valid only when **VI_VSYNC_VALID_E** is set to **VI_VSYNC_NORM_SINGAL**.



[See Also]

- [VI_VSYNC_VALID_E](#)
- [VI_SYNC_CFG_S](#)

VI_TIMING_BLANK_S

[Description]

Defines the blanking information about the input timing of a VI device.

[Syntax]

```
typedef struct hIVI_TIMING_BLANK_S
{
    HI_U32 u32HsyncHfb;
    HI_U32 u32HsyncAct;
    HI_U32 u32HsyncHbb;
    HI_U32 u32VsyncVfb;
    HI_U32 u32VsyncVact;
    HI_U32 u32VsyncVbb;
    HI_U32 u32VsyncVbfb;
    HI_U32 u32VsyncVbact;
    HI_U32 u32VsyncVbbb;
}VI_TIMING_BLANK_S;
```

[Member]

Member	Description
u32HsyncHfb	Horizontal front blanking width
u32HsyncAct	Horizontal active width
u32HsyncHbb	Horizontal back blanking width
u32VsyncVfb	Vertical front blanking height of the frame picture or the odd vertical front blanking height during frame input
u32VsyncVact	Vertical active height of the frame picture or the odd vertical active width during frame input
u32VsyncVbb	Vertical back blanking height of the frame picture or the odd vertical back blanking height during frame input
u32VsyncVbfb	Even vertical front blanking height during interlaced input (invalid during frame input)
u32VsyncVbact	Even vertical active height during interlaced input (invalid during frame input)
u32VsyncVbbb	Even vertical back blanking height during interlaced input (invalid during frame input)



[Note]

This data structure takes effect only in BT.601 timing.

[See Also]

[VI_SYNC_CFG_S](#)

VI_SYNC_CFG_S

[Description]

Defines the sync information about the BT.601 timing or DC timing received by a VI device.

[Syntax]

```
typedef struct hiVI_SYNC_CFG_S
{
    VI_VSYNC_E          enVsync;
    VI_VSYNC_NEG_E     enVsyncNeg;
    VI_HSYNC_E          enHsync;
    VI_HSYNC_NEG_E     enHsyncNeg;
    VI_VSYNC_VALID_E   enVsyncValid;
    VI_VSYNC_VALID_NEG_E enVsyncValidNeg;
    VI_TIMING_BLANK_S stTimingBlank;
} VI_SYNC_CFG_S;
```

[Member]

Member	Description
enVsync	Type of the vertical sync signal
enVsyncNeg	Polarity of the vertical sync signal
enHsync	Type of the horizontal sync signal
enHsyncNeg	Polarity of the horizontal sync signal
enVsyncValid	Type of the vertical valid sync signal
enVsyncValidNeg	Polarity of the vertical valid sync signal
stTimingBlank	Blanking information about the input timing

[Note]

- In BT.601 mode, only the vertical sync signals **enVsync** and **enVsyncNeg** need to be configured. **enVsync** indicates the type of the vertical sync signal of the **VI_P_VSYNC_FIELD** pin, and **enVsyncNeg** indicates the polarity of the vertical sync signal of the **VI_P_VSYNC_FIELD** pin. The two signals determine the vertical sync timing to be used.
 - **enVsync** is set to **VI_VSYNC_FIELD**, which indicates the field number in BT.601 mode. In this case, if **enVsyncNeg** is set to **VI_VSYNC_NEG_HIGH**, the high level indicates even field and the low level indicates odd field; if **enVsyncNeg** is set to



VI_VSYNC_NEG_LOW, the low level indicates even field and the high level indicates odd field

- **enVsync** is set to **VI_VSYNC_PULSE**, which indicates the vertical sync pulse. That is, when a new pulse arrives, a new field or frame starts. In this case, if **enVsyncNeg** is set to **VI_VSYNC_NEG_HIGH**, the positive pulse indicates vertical sync pulse; if **enVsyncNeg** is set to **VI_VSYNC_NEG_LOW**, the negative pulse indicates vertical sync pulse.
- In DC mode, the vertical sync signals **enVsync**, **enVsyncValid**, and **enVsyncValidNeg** need to be configured. **enVsync** indicates the type of the vertical sync signal of the **VI_P_VSYNC_FIELD** pin, **enVsyncValid** indicates the vertical sync signal flag of the **VI_P_VSYNC_FIELD** pin, and **enVsyncValidNeg** indicates the polarity of the vertical valid signal of the **VI_P_VSYNC_FIELD** pin. **enVsyncValidNeg** is valid only when **enVsyncValid** is set to **VI_VSYNC_VALID_SINGAL**.
 - **enVsync** is set to **VI_VSYNC_PULSE**, which indicates the vertical timing pulse mode. If the hsync signal is valid in the vertical blanking area, **enVsyncValid** must be set to **VI_VSYNC_VALID_SINGAL** to select the vertical valid flag and **enVsyncValidNeg** must be set to **VI_VSYNC_VALID_NEG_HIGH** (active high) or **VI_VSYNC_VALID_NEG_LOW** (active low). In other cases, **enVsyncValidNeg** can be ignored.
 - If the input timing is in vertical timing line active mode, **enVsync** must be set to **VI_VSYNC_FIELD**.
- **enHsync** indicates the type of the horizontal sync signal of the **VI_P_HSYNC_VD** pin, and **enHsyncNeg** indicates the polarity of the horizontal sync signal of the **VI_P_HSYNC_VD** pin. **enHsync** and **enHsyncNeg** determine the horizontal sync timing to be used.
 - **enHsync** is set to **VI_HSYNC_VALID_SIGNAL**, which indicates the data valid signal. In this case, if **enHsyncNeg** is set to **VI_HSYNC_NEG_HIGH**, the high level indicates data valid; if **enHsyncNeg** is set to **VI_HSYNC_NEG_LOW**, the low level indicates data valid.
 - **enHsync** is set to **VI_HSYNC_PULSE**, which indicates the horizontal sync pulse. That is, when a new pulse arrives, a new line starts. In this case, if **enHsyncNeg** is set to **VI_HSYNC_NEG_HIGH**, the positive pulse indicates horizontal sync pulse; if **enHsyncNeg** is set to **VI_HSYNC_NEG_LOW**, the negative pulse indicates horizontal sync pulse.

Note that the blanking area does not need to be configured for the horizontal or vertical valid data.

For details about the timings supported by the Hi3516A, see the *Hi3516A HD IP Camera SoC Data Sheet*.

[See Also]

- [VI_DEV_ATTR_S](#)
- [HI_MPI_VI_SetDevAttr](#)
- [VI_DEV_ATTR_S](#)
- [HI_MPI_VI_SetDevAttrEx](#)

BT656_FIXCODE_E

[Description]

Defines the most significant bit (MSB) of the BT.656 timing reference code.



[Syntax]

```
typedef enum hiBT656_FIXCODE_E
{
    BT656_FIXCODE_1 = 0,
    BT656_FIXCODE_0
}BT656_FIXCODE_E;
```

[Member]

Member	Description
BT656_FIXCODE_1	The MSB of the start of active video (SAV) or end of active video (EAV) of the BT.656 protocol is fixed at 1.
BT656_FIXCODE_0	The MSB of the SAV or EAV of the BT.656 protocol is fixed at 0.

[Note]

None

[See Also]

- [VI_DEV_ATTR_EX_S](#)
- [HI_MPI_VI_SetDevAttrEx](#)

BT656_FIELD_POLAR_E

[Description]

Defines the polarity of the field indication bit (F) of the BT.656 timing reference code.

[Syntax]

```
typedef enum hiBT656_FIELD_POLAR_E
{
    BT656_FIELD_POLAR_STD = 0,
    BT656_FIELD_POLAR_NSTD
}BT656_FIELD_POLAR_E;
```

[Member]

Member	Description
BT656_FIELD_POLAR_STD	In standard mode, F is 0 in the first field and 1 in the second field.
BT656_FIELD_POLAR_NSTD	In non-standard mode, F is 1 in the first field and 0 in the second field.

[Note]



None

[See Also]

- [VI_DEV_ATTR_EX_S](#)
- [HI_MPI_VI_SetDevAttrEx](#)

VI_BT656_SYNC_CFG_S

[Description]

Defines the sync information about the BT.656 timing received by a VI device.

[Syntax]

```
typedef struct hiVI_BT656_SYNC_CFG_S
{
    BT656_FIXCODE_E      enFixCode;
    BT656_FIELD_POLAR_E enFieldPolar;
}VI_BT656_SYNC_CFG_S;
```

[Member]

Member	Description
enFixCode	MSB of the BT.656 timing reference code
enFieldPolar	Polarity of the field indication bit (F) of the BT.656 timing reference code

[Note]

For details about the definitions related to BT.656, see related protocols.

[See Also]

- [VI_DEV_ATTR_EX_S](#)
- [HI_MPI_VI_SetDevAttrEx](#)

VI_DEV_ATTR_S

[Description]

Defines the attributes of a VI device.

[Syntax]

```
typedef struct hiVI_DEV_ATTR_S
{
    VI_INTF_MODE_E      enIntfMode;
    VI_WORK_MODE_E      enWorkMode;
    HI_U32   au32        CompMask[VIU_DEV_COMP_MASK_NUM];
    VI_SCAN_MODE_E      enScanMode;
    HI_S32              s32AdChnId[VIU_DEV_AD_CHN_NUM];
```



```
/*The preceding parameters must be set in BT.601 mode or DC mode and
are invalid in other modes.*/
VI_DATA_YUV_SEQ_E    enDataSeq;
VI_SYNC_CFG_S         stSynCfg;
VI_DATA_PATH_E        enDataPath;
VI_DATA_TYPE_E        enInputDataType;
RECT_S                stDevRect;
HI_BOOL               bDataRev;
} VI_DEV_ATTR_S;
```

[Member]

Member	Description
enIntfMode	Interface mode
enWorkMode	1-, 2-, or 4-channel multiplexing mode
au32CompMask[VIU_DEV_COMP_MASK_NUM]	Component mask When enIntfMode is set to VI_MODE_BT1120_STANDARD , the component masks of the Y component and C component must be configured. In other cases, only single-component mask needs to be configured.
enScanMode	Input scanning mode (progressive or interlaced)
s32AdChnId[VIU_DEV_AD_CHN_NUM]	Its value range is [-1, +3]. Typically, the default value -1 is recommended.
enDataSeq	Input data sequence (only the YUV format is supported). This member must be configured in BT.601 mode or DC mode and is invalid in other modes.
stSynCfg	Sync timing. This member must be configured in BT.601 mode or DC mode and is invalid in other modes.
enDataPath	Input data path. If data is from a sensor without ISP, select the ISP channel. If data is from a sensor with an ISP or from the ADC, select the bypass channel (in this case, the build-in ISP will be disabled). The value Raw Data is used only for debugging. If enDataPath is set to a raw data channel, the raw data from the sensor can be captured.
enInputDataType	Input data type. Typically, the input data from the sensor is RGB data, and the input data from the ADC is YUV data.
bDataRev	The data lines of the AD/sensor may reversely connect to those of the VI due to hardware



Member	Description
	routing restrictions. For example, the VIU_DATA0 pin on the VIU may connect to the DATA7 pin on the AD, and the VIU_DATA7 pin on the VIU may connect to the DATA0 pin on the AD. If the pins on the AD or sensor connect to the same pins on the VIU, bDataRev is HI_FALSE . Otherwise, bDataRev is HI_TRUE .
stDevRect	The VI device can configure the start position, width, and height of the picture to be captured. The source video picture is cropped by the VI device (front end), which reduces the pictures written by the WDR/FPN as well as the required bandwidth and memory. The value ranges of the width and height for the captured picture are shown as follows: Width: [VIU_DEV_MIN_WIDTH , VIU_DEV_MAX_WIDTH] Height: [VIU_DEV_MIN_HEIGHT , VIU_DEV_MAX_HEIGHT]

[Difference]

None

[Note]

- When **s32AdChnId[i]** is set to **-1**, the MPP detects the **AdId** numbered i by default. An AdId cannot be repeatedly detected. That is, **s32AdChnId** cannot be set to **-1, 0, 2, or 3** because the values of **s32AdChnId[0]** and **s32AdChnId[1]** conflict when the value **-1** of **s32AdChnId[0]** is converted into 0.
- When **enWorkMode** is set to 1-channel multiplexing mode, **s32AdChnId[0]** must be set to **-1** or **0**; when **WorkMode** is set to 2-channel multiplexing mode, **s32AdChnId[0]** or **s32AdChnId[1]** must be set to **-1, 0, or +1**.
- You are advised to set **s32AdChnId** to **-1**. That is, the AdIds 0, 1, 2, and 3 are detected in sequence in default binding configurations.
- Because the Hi3516A/Hi3518E V200/Hi3519 V100 supports only 1-channel multiplexing mode, **s32AdChnId** must be set to **-1** or **0**.
- In **stDevRect**, the value of **(s32X + u32Width)** cannot be greater than the width of the actual input picture, and the value of **(s32Y + u32Height)** cannot be greater than the height of the actual input picture. Otherwise, there is no output picture.

[See Also]

[HI_MPI_VI_SetDevAttr](#)

VI_DEV_ATTR_EX_S

[Description]



Defines the extended attributes of a VI device.

[Syntax]

```
typedef struct hiVI_DEV_ATTR_EX_S
{
    VI_INPUT_MODE_E enInputMode;
    VI_WORK_MODE_E   enWorkMode;
    VI_COMBINE_MODE_E enCombineMode;
    VI_COMP_MODE_E   enCompMode;
    VI_CLK_EDGE_E    enClkEdge;
    HI_U32           au32CompMask[VIU_DEV_COMP_MASK_NUM];
    VI_SCAN_MODE_E   enScanMode;
    HI_S32           s32AdChnId[VIU_DEV_AD_CHN_NUM];
    VI_DATA_YUV_SEQ_E enDataSeq;
    VI_SYNC_CFG_S    stSynCfg;
    VI_BT656_SYNC_CFG_S stBT656SynCfg;
    VI_DATA_PATH_E   enDataPath;
    VI_DATA_TYPE_E   enInputDataType;
    HI_BOOL          bDataRev;
    RECT_S           stDevRect;
} VI_DEV_ATTR_EX_S;
```

[Member]

Member	Description
enInputMode	Interface mode
enWorkMode	1-, 2-, or 4-channel multiplexing mode
enCombineMode	Y/C composite or separation mode
enCompMode	Component mode (single-component or dual-component), corresponding to enCombineMode
au32CompMask[VIU_DEV_COMP_MASK_NUM]	Component mask When enInputMode is set to VI_INPUT_MODE_BT656 , enCompMode is set to VI_COMP_MODE_DOUBLE , and enCombineMode is set to VI_COMBINE_SEPARATE , VI_MODE_BT1120_STANDARD is supported. In this case, the component masks of the Y component and C component must be configured. In other cases, only single-component mask needs to be configured.
enClkEdge	Clock edge mode (sampling on the rising or falling edge). This member is not supported currently. The clock edge mode can be switched by using the clock phase control function of the VI interface. For details about the configuration,



Member	Description
	see the description of the CRG register (VI interface clock phase control register) in the "System" chapter of data sheet for the corresponding chip.
enScanMode	Input scanning mode (progressive or interlaced)
enDataSeq	Input data sequence (only the YUV format is supported)
s32AdChnId[VIU_DEV_AD_CHN_NUM]	For details, see the description of VI_DEV_ATTR_S .
stSynCfg	Sync timing. This member must be configured in BT.601 mode or DC mode and is invalid in other modes.
stBT656SynCfg	Sync timing. This member must be configured in BT.656 mode.
enDataPath	Input data path. If data is from a sensor without ISP, select the ISP channel. If data is from a sensor with an ISP or from the ADC, select the bypass channel (in this case, the build-in ISP will be disabled). The value Raw Data is used only for debugging. If enDataPath is set to a raw data channel, the raw data from the sensor can be captured.
enInputDataType	Input data type. Typically, the input data from the sensor is RGB data, and the input data from the ADC is YUV data.
bDataRev	The data lines of the AD/sensor may reversely connect to those of the VI due to hardware routing restrictions. For example, AD_DATA0–AD_DATA7 connect to VIU_DATA7–VIU_DATA0 in sequence. To be specific, AD_DATA0 connects to VIU_DATA7, AD_DATA7 connects to VIU_DATA0, and so on. This is the inversion based on mask configuration, that is, when the connected pins are determined. If the pins on the AD or sensor connect to the same pins on the VIU, bDataRev is HI_FALSE . Otherwise, bDataRev is HI_TRUE .
stDevRect	The VI device can configure the start position, width, and height of the picture to be captured. The source video picture is cropped by the VI device (front end), which reduces the pictures written by the WDR/FPN as well as the required bandwidth and memory. The value ranges of the width and height for the captured picture are shown as follows:



Member	Description
	Width: [VIU_DEV_MIN_WIDTH, VIU_DEV_MAX_WIDTH] Height: [VIU_DEV_MIN_HEIGHT, VIU_DEV_MAX_HEIGHT]

[Difference]

None

[Note]

None

[See Also]

[HI_MPI_VI_SetDevAttrEx](#)

VI_CHN_BIND_ATTR_S

[Description]

Defines the binding attributes of a VI channel.

[Syntax]

```
typedef struct hiVI_CHN_BIND_ATTR_S
{
    VI_DEV ViDev;
    VI_WAY ViWay;
} VI_CHN_BIND_ATTR_S;
```

[Member]

Member	Description
ViDev	Device port bound to a VI channel
ViWay	Way to which the device port bound to a VI channel belongs

[Note]

None

[See Also]

[HI_MPI_VI_BindChn](#)

VI_CAPSEL_E

[Description]

Defines the frame/field selection of the captured VI pictures.



[Syntax]

```
typedef enum hiVI_CAPSEL_E
{
    VI_CAPSEL_TOP=0,
    VI_CAPSEL_BOTTOM,
    VI_CAPSEL_BOTH,
    VI_CAPSEL_BUTT
} VI_CAPSEL_E;
```

[Member]

Member	Description
VI_CAPSEL_TOP	The top field is selected.
VI_CAPSEL_BOTTOM	The bottom field is selected (recommended when a single field is captured).
VI_CAPSEL_BOTH	Both the top field and bottom field are selected.

[Note]

None

[See Also]

None

VI_CHN_ATTR_S

[Description]

Defines the attributes of a VI channel.

[Syntax]

```
typedef struct hiVI_CHN_ATTR_S
{
    RECT_S          stCapRect;
    SIZE_S          stDestSize;
    VI_CAPSEL_E    enCapSel;
    PIXEL_FORMAT_E enPixFormat;
    COMPRESS_MODE_E enCompressMode;
    HI_BOOL        bMirror;
    HI_BOOL        bFlip;
    HI_S32         s32SrcFrameRate;
    HI_S32         s32DstFrameRate;
} VI_CHN_ATTR_S;
```

[Member]



Member	Description
stCapRect	<p>Start coordinates of the capture region relative to the device picture size, and width and height of the capture region.</p> <p>The value ranges of the width and height for the capture region are shown as follows:</p> <p>Width: [VIU_CHN_MIN_WIDTH, VIU_CHN_MAX_WIDTH] Height: [VIU_CHN_MIN_HEIGHT, VIU_CHN_MAX_HEIGHT]</p>
stDestSize	<p>Target picture size</p> <p>The value ranges of the width and height for the target picture are shown as follows:</p> <p>Width: [VIU_CHN_MIN_WIDTH, VIU_CHN_MAX_WIDTH] Height: [VIU_CHN_MIN_HEIGHT, VIU_CHN_MAX_HEIGHT]</p>
enCapSel	<p>Frame/Field select</p> <p>It is used only in interlaced mode. You are advised to capture the bottom field in single-field mode.</p> <p>In progressive mode, this parameter must be set to VI_CAPSEL_BOTH.</p>
enPixelFormat	Pixel storage format
enCompressMode	Whether to compress. The non-compression mode and 256-byte-segment-based compression mode are supported.
bMirror	Whether to mirror.
bFlip	Whether to flip.
s32SrcFrameRate	<p>Source frame rate</p> <p>It cannot be less than -1. You are advised to set the source frame rate to the frame rate of the connected camera.</p>
s32DstFrameRate	<p>Target frame rate</p> <p>It must be less than or equal to the source frame rate, and cannot be less than -1. The source frame rate and target frame rate must be both -1 or neither is -1. If both the source frame rate and target frame rate are -1, the frame rate is not controlled.</p>

[Difference]

Chip Type	Supported Pixel Format
Hi3516A	PIXEL_FORMAT_YUV_SEMIPLANAR_420 PIXEL_FORMAT_YUV_SEMIPLANAR_422
Hi3518E V200	PIXEL_FORMAT_YUV_SEMIPLANAR_420 PIXEL_FORMAT_YUV_SEMIPLANAR_422 PIXEL_FORMAT_YUV_400 PIXEL_FORMAT_RGB_BAYER



Chip Type	Supported Pixel Format
Hi3519 V100	PIXEL_FORMAT_YUV_SEMIPLANAR_420 PIXEL_FORMAT_YUV_SEMIPLANAR_422 PIXEL_FORMAT_YUV_400 PIXEL_FORMAT_RGB_BAYER

Chip Type	Mirror/Flip
Hi3516A	Not supported
Hi3518E V200	Supported in offline mode and not supported in online mode
Hi3519 V100	Supported in offline mode and not supported in online mode

[Note]

- Static attributes can be modified only after channels are disabled.
- Interlaced pictures do not support mirroring and flipping. If the function is enabled, the sequence of capturing top and bottom fields is reversed, which may affect the picture quality when pictures are processed by other modules. As the MPP does not check whether the function is enabled, you need to disable the function in interlaced mode.
- **stCapRect** is valid only for the physical channel, and the width and height defined in **stCapRect** are static attributes.
- As the VI channel does not support scaling, **stDestSize** must be the same as the width and height defined in **stCapRect** in dual-field capture mode, and the height (static attribute) must be half of the height defined in **stCapRect** in single-field capture mode.
- **enCapSel** is a dynamic attribute.

[See Also]

- [HI_MPI_VI_SetChnAttr](#)
- [HI_MPI_VI_GetChnAttr](#)

VI_USERPIC_MODE_E

[Description]

Defines the user picture mode.

[Syntax]

```
typedef enum hi_VI_USERPIC_MODE_E
{
    VI_USERPIC_MODE_PIC = 0,
    VI_USERPIC_MODE_BGC,
    VI_USERPIC_MODE_BUTT,
} VI_USERPIC_MODE_E;
```

[Member]



Member	Description
VI_USERPIC_MODE_PIC	YUV picture
VI_USERPIC_MODE_BGC	Background picture only with a color

[Note]

None

[See Also]

None

VI_USERPIC_BGC_S

[Description]

Defines the information about a background picture only with a color.

[Syntax]

```
typedef struct hiVI_USERPIC_BGC_S
{
    HI_U32 u32BgColor;
} VI_USERPIC_BGC_S;
```

[Member]

Member	Description
u32BgColor	Filled data

[Note]

None

[See Also]

None

VI_USERPIC_ATTR_S

[Description]

Defines the information about a user picture.

[Syntax]

```
typedef struct hiVI_USERPIC_ATTR_S
{
    HI_BOOL          bPub;
    VI_USERPIC_MODE_E enUsrPicMode;
    union
```



```
{  
    VIDEO_FRAME_INFO_S    stUsrPicFrm;  
    VI_USERPIC_BGC_S     stUsrPicBg;  
} unUsrPic;  
} VI_USERPIC_ATTR_S;
```

[Member]

Member	Description
bPub	Whether the user picture information is shared by all VI devices and channels
enUsrPicMode	User picture mode
stUsrPicFrm	Information about a YUV picture
stUsrPicBg	Information about a user picture in single-color background mode Value range: [0x0, 0xFFFFFFF]

[Note]

Currently, the user picture information is shared by all devices and channels. That is, bPub must be set to HI_TRUE. Otherwise, the kernel returns an error.

[See Also]

None

VI_CHN_STAT_S

[Description]

Defines the structure of VI channel information.

[Syntax]

```
typedef struct hIVI_CHN_STAT_S  
{  
    HI_BOOL bEnable; /* Whether this channel is enabled */  
    HI_U32 u32IntCnt; /* The video frame interrupt count */  
    HI_U32 u32FrmRate; /* current frame rate */  
    HI_U32 u32LostInt; /* The interrupt is received but nobody care */  
    HI_U32 u32VbFail; /* Video buffer malloc failure */  
    HI_U32 u32PicWidth; /* current picture width */  
    HI_U32 u32PicHeight; /* current picture height */  
} VI_CHN_STAT_S;
```

[Member]



Member	Description
bEnable	Channel enable
u32IntCnt	Interrupt count
u32FrmRate	Average frame rate that is calculated every 10 seconds. This value may be inaccurate. This member is meaningless in online mode.
u32LostInt	Lost interrupt count. This member is meaningless in online mode.
u32VbFail	Number of times that VBs fail to be obtained. This member is meaningless in online mode.
u32PicWidth	Picture width
u32PicHeight	Picture height

[Note]

- The interrupt count is used to check whether interrupts are generated.
- The frame rate is the average frame rate that is calculated every 10 seconds. Therefore, the average frame rate is inaccurate.
- If the queried lost interrupt count continuously increases, an exception occurs in the VI channel.

[See Also]

None

VI_DATA_PATH_E

[Description]

Defines the VI data channel enumeration.

[Syntax]

```
typedef enum hiVI_DATA_PATH_E
{
    VI_PATH_BYPASS      = 0,          /* ISP bypass */
    VI_PATH_ISP         = 1,          /* ISP enable */
    VI_PATH_RAW         = 2,
    VI_PATH_BUTT
} VI_DATA_PATH_E;
```

[Member]

Member	Description
VI_PATH_BYPASS	VIU internal ISP bypass
VI_PATH_ISP	VIU internal ISP enable



Member	Description
VI_PATH_RAW	This member is reserved and not supported.

[Note]

- When the input YUV data is from the sensor with an ISP, set this data type to VI_PATH_BYPASS.
- When the input data is from the sensor without ISP, set this data type to VI_PATH_ISP.

[See Also]

- [VI_DEV_ATTR_S](#)
- [VI_DEV_ATTR_EX_S](#)

VI_DATA_TYPE_E

[Description]

Defines the VI data type enumeration.

[Syntax]

```
typedef enum hiVI_DATA_TYPE_E
{
    VI_DATA_TYPE_YUV = 0,
    VI_DATA_TYPE_RGB = 1,
    VI_DATA_TYPE_BUTT
} VI_DATA_TYPE_E;
```

[Member]

Member	Description
VI_DATA_TYPE_YUV	The input data format is YUV and the ADC is connected at the VI front-end.
VI_DATA_TYPE_RGB	The input data format is RGB and the sensor is connected at the VI front-end.

[Note]

None

[See Also]

- [VI_DEV_ATTR_S](#)
- [VI_DEV_ATTR_EX_S](#)

VI_EXT_CHN_ATTR_S

[Description]



Defines the attributes of an extended VI channel.

[Syntax]

```
typedef struct hiVI_EXT_CHN_ATTR_S
{
    VI_CHN    s32BindChn;
    SIZE_S     stDestSize;
    HI_S32    s32SrcFrameRate;
    HI_S32    s32DstFrameRate;
    PIXEL_FORMAT_E enPixelFormat;
    COMPRESS_MODE_E enCompressMode;
}VI_EXT_CHN_ATTR_S;
```

[Member]

Member	Description
s32BindChn	Channel to which an extended channel is bound (static attribute).
stDestSize	Target picture size (static attribute). Minimum width: VIU_EXTCHN_MIN_WIDTH Maximum width: VIU_EXTCHN_MAX_WIDTH Minimum height: VIU_EXTCHN_MIN_HEIGHT Maximum height: VIU_EXTCHN_MAX_HEIGHT
s32SrcFrameRate	Source frame rate. It cannot be less than -1. You are advised to set it the same as the frame rate of the interconnected camera.
s32DstFrameRate	Target frame rate. It must be less than or equal to the source frame rate, and cannot be less than -1. The source frame rate and target frame rate must be both -1 or neither is -1. If both the source frame rate and target frame rate are -1, the frame rate is not controlled.
enPixelFormat	Pixel storage format.
enCompressMode	Whether to compress. The non-compression mode and 256-byte-segment-based compression mode are supported.

[Difference]

Chip Type	Supported Pixel Format
Hi3516A	PIXEL_FORMAT_YUV_SEMIPLANAR_422 PIXEL_FORMAT_YUV_SEMIPLANAR_420
Hi3518EV200	PIXEL_FORMAT_YUV_SEMIPLANAR_422 PIXEL_FORMAT_YUV_SEMIPLANAR_420 PIXEL_FORMAT_YUV_400



Chip Type	Supported Pixel Format
Hi3519V100	PIXEL_FORMAT_YUV_SEMIPLANAR_422 PIXEL_FORMAT_YUV_SEMIPLANAR_420 PIXEL_FORMAT_YUV_400

[Note]

The target picture size is the width and height of the output picture of the extended channel.

[See Also]

- [HI_MPI_VI_SetExtChnAttr](#)
- [HI_MPI_VI_GetExtChnAttr](#)

LDC_VIEW_TYPE_E

[Description]

Defines the LDC mode.

[Syntax]

```
typedef enum hiLDC_VIEW_TYPE_E
{
    LDC_VIEW_TYPE_ALL = 0,
    LDC_VIEW_TYPE_CROP,
    LDC_VIEW_TYPE_BUTT,
} LDC_VIEW_TYPE_E;
```

[Member]

Member	Description
LDC_VIEW_TYPE_ALL	Full mode. The maximum rectangle is obtained based on the edges of the corrected picture, and then the rectangle is zoomed out to the source picture size. Picture edges may be retained.
LDC_VIEW_TYPE_CROP	Crop mode

[Difference]

Chip	LDC_VIEW_TYPE_CROP
Hi3516A/Hi3518E V200	The corrected picture is cropped and the size of the cropped region is equal to the size of the source picture. However, picture edges may be lost.
Hi3519 V100	LDC is not supported.



[Note]

None

[See Also]

- [LDC_ATTR_S](#)
- [VI_LDC_ATTR_S](#)

LDC_ATTR_S

[Description]

Defines the LDC attribute.

[Syntax]

```
typedef struct hiLDC_ATTR_S
{
    LDC_VIEW_TYPE_E enViewType;
    HI_S32 s32CenterXOffset;
    HI_S32 s32CenterYOffset;
    HI_S32 s32Ratio;
} LDC_ATTR_S;
```

[Member]

Member	Description
enViewType	LDC mode, including crop mode and full mode
s32CenterXOffset	Horizontal offset of the distortion center relative to the picture center
s32CenterYOffset	Vertical offset of the distortion center relative to the picture center
s32Ratio	Distortion ratio

[Note]

Hi3516A:

- The value range of **s32CenterXOffset** is [-75, +75].
- The value range of **s32CenterYOffset** is [-75, +75].
- When the resolution of the captured VI picture is not greater than D1, the value range of **s32Ratio** is [0, 480].
- When the resolution of the captured VI picture is greater than D1 but not greater than 720p, the value range of **s32Ratio** is [0, 433].
- When the resolution of the captured VI picture is greater than 720p but not greater than 1080p, the value range of **s32Ratio** is [0, 400].
- When the resolution of the captured VI picture is greater than 1080p but not greater than 2304 x 1536, the value range of **s32Ratio** is [0, 300].



- When the resolution of the captured VI picture is greater than 2304 x 1536 but not greater than 5 megapixels, the value range of **s32Ratio** is [0, 168].

Hi3518E V200:

- The value range of **s32CenterXOffset** is [-75, +75].
- The value range of **s32CenterYOffset** is [-75, +75].
- When the resolution of the captured VI picture is not greater than D1, the value range of **s32Ratio** is [0, 150].
- When the resolution of the captured VI picture is greater than D1 but not greater than 720p, the value range of **s32Ratio** is [0, 150].
- When the resolution of the captured VI picture is greater than 720p but not greater than 1080p, the value range of **s32Ratio** is [0, 100].
- When the resolution of the captured VI picture is greater than 1080p but not greater than 3 megapixels, the value range of **s32Ratio** is [0, 50].

Hi3519 V100 does not support LDC.

[See Also]

[LDC_VIEW_TYPE_E](#)

VI_LDC_ATTR_S

[Description]

Defines the VI LDC attribute.

[Syntax]

```
typedef struct hiVI_LDC_ATTR_S
{
    HI_BOOL bEnable;
    LDC_ATTR_S stAttr;
}VI_LDC_ATTR_S;
```

[Member]

Member	Description
bEnable	LDC enable
stAttr	LDC attribute

[Note]

None

[See Also]

- [LDC_VIEW_TYPE_E](#)
- [LDC_ATTR_S](#)
- [HI_MPI_VI_SetLDCAttr](#)
- [HI_MPI_VI_GetLDCAttr](#)



VI_CSC_TYPE_E

[Description]

Defines the YUV CSC standard.

[Syntax]

```
typedef enum hiVI_CSC_TYPE_E
{
    VI_CSC_TYPE_601 = 0,
    VI_CSC_TYPE_709,
    VI_CSC_TYPE_USER,
    VI_CSC_TYPE_BUTT,
} VI_CSC_TYPE_E;
```

[Member]

Member	Description
VI_CSC_TYPE_601	BT.601 CSC standard
VI_CSC_TYPE_709	BT.709 CSC standard
VI_CSC_TYPE_USER	User-defined color space coefficient

[Note]

The YUV CSC standards are mainly classified into two types: BT.601 and BT.709. This enumeration defines the CSC standard that is used when RGB pictures are converted into YUV pictures. Typically, the BT.601 standard is used for the SD device, and the BT.709 standard is used for the HD device. Besides, when this enumeration is defined as VI_CSC_TYPE_USER, the user can define the coefficients of the CSC matrix.

[See Also]

None

VI_CSC_MATRIX_S

[Description]

Defines the coefficients of the CSC matrix.

[Syntax]

```
typedef struct hiVI_CSC_MATRIX_S
{
    HI_S32 s32CSCIdc[VIU_CSC_IDC_NUM];
    HI_S32 s32CSCODc[VIU_CSC_ODC_NUM];
    HI_S32 s32CSCCoef[VIU_CSC_COEF_NUM];
} VI_CSC_MATRIX_S;
```

[Member]



Member	Description
s32CSCIdc[VIU_CSC_IDC_NUM]	Offset of the input DC component. s32CSCIdc[0] , s32CSCIdc[1] , and s32CSCIdc[2] correspond to the DC offsets of the input R component, input G component, and input B component, respectively. Value range: [-256, +255]
s32CSCOdc[VIU_CSC_ODC_NUM]	Offset of the output DC component. s32CSCOdc[0] , s32CSCOdc[1] , s32CSCOdc[2] correspond to the DC offsets of the output Y component, output U component, and output V component, respectively. Value range: [-256, +255]
s32CSCCoef[VIU_CSC_COE_F_NUM]	CSC coefficient matrix. The value range of each element in this array is [-16384, +16383].

[Note]

The coefficients are valid only when VI_CSC_TYPE_E is VI_CSC_TYPE_USER.

[See Also]

- [VI_CSC_ATTR_S](#)
- [VI_CSC_TYPE_E](#)
- [HI_MPI_VI_SetCSCAttr](#)
- [HI_MPI_VI_GetCSCAttr](#)

VI_CSC_ATTR_S

[Description]

Defines the CSC attribute.

[Syntax]

```
typedef struct hIVI_CSC_ATTR_S
{
    VI_CSC_TYPE_E      enViCscType;          /* 601 or 709 or user */
    HI_U32 u32LumaVal;                      /*Luminance: [0-100] */
    HI_U32 u32ContrVal;                     /*Contrast: [0-100] */
    HI_U32 u32HueVal;                      /*Hue: [0-100] */
    HI_U32 u32SatuVal;                      /*Saturation: [0-100] */
    HI_BOOL bTVModeEn;                      /* TV Mode: true or false */
    VI_CSC_MATRIX_S stCSCMatrix;           /* The coefficient matrix of CSC */
} VI_CSC_ATTR_S;
```

[Member]



Member	Description
enViCscType	YUV CSC standard
u32LumaVal	Luminance adjustment Default value: 50 Value range: [0, 100]
u32ContrVal	Contrast adjustment Default value: 50 Value range: [0, 100]
u32HueVal	Hue adjustment Default value: 50 Value range: [0, 100]
u32SatuVal	Saturation adjustment Default value: 50 Value range: [0, 100]
bTVModeEn	When bTVModeEn is HI_FALSE , the value of the constant coefficient matrix for the color space conversion ranges from 0 to 255. When bTVModeEn is HI_TRUE , the value of the constant coefficient matrix for the color space conversion ranges from 16 to 235.
stCSCMatrix	User-defined CSC matrix. This parameter is valid only when enViCscType is VI_CSC_TYPE_USER .

[Note]

None

[See Also]

- [VI_CSC_TYPE_E](#)
- [VI_CSC_MATRIX_S](#)
- [HI_MPI_VI_SetCSCAttr](#)
- [HI_MPI_VI_GetCSCAttr](#)

VI_FLASH_MODE_E

[Description]

Defines the camera flash mode.

[Syntax]

```
typedef enum hiVI_FLASH_MODE_E
{
    VI_FLASH_ONCE = 0,           /*Blink once*/
    VI_FLASH_CONTINUOUS,        /*Blink continuously*/
    VI_FLASH_STROBE,            /*Strobe*/
    VI_FLASH_TORCH,             /*Torch*/
    VI_FLASH_TORCH_STROBE,      /*Torch + Strobe*/
    VI_FLASH_TORCH_STROBE_CONT /*Torch + Strobe + Continuous*/
}
```



```
    VI_FLASH_FREQ = 1,           /*Blink frequently*/
    VI_FLASH_MODE_BUTT
}VI_FLASH_MODE_E;
```

[Member]

Member	Description
VI_FLASH_ONCE	Single-blink mode. The camera flash blinks only once.
VI_FLASH_FREQ	Multi-blink mode. The camera flash blinks periodically.

[Note]

None

[See Also]

[VI_FLASH_CONFIG_S](#)

VI_FLASH_CONFIG_S

[Description]

Defines camera flash settings.

[Syntax]

```
typedef struct hivi_FLASH_CONFIG_S
{
    VI_FLASH_MODE_E    enFlashMode;
    HI_U32             u32StartTime;
    HI_U32             u32Duration;
    HI_U32             u32CapFrmIndex;
    HI_U32             u32Interval;
}VI_FLASH_CONFIG_S;
```

[Member]

Member	Description
enFlashMode	Camera flash mode
u32StartTime	Start time of the camera flash. Its unit is clock cycle.
u32Duration	Duration in which the camera flash is on. Its unit is clock cycle.
u32CapFrmIndex	Frame to be captured. This member is used to set the frame to be captured after the camera flash blinks. The value 0 indicates the current frame is the one to be captured. Value range: [0, 5]
u32Interval	Blink interval, in frame



[Note]

- When the next frame is received after HI_MPI_VI_FlashTrigger is called, the VI device starts to trigger the camera flash signal **u32StartTime** clock cycles later after the field start interrupt is generated. The camera flash is on for **u32Duration** clock cycles. When the camera flash is set based on sensor timings, u32StartTime must include the clock cycle of the blanking area. The clock cycles are equal to the number of pixels.
- The sum of **u32StartTime** and **u32Duration** cannot be greater than the total clock cycles of a frame.
- In multi-blink mode, the camera flash signal is triggered for multiple times based on the interval defined in **u32Interval**. In single-blink mode, the camera flash signal is triggered only once, and **u32Interval** is invalid.
- In multi-blink mode, **u32Interval** must be greater than **u32CapFrmIndex**.

[See Also]

- [VI_FLASH_MODE_E](#)
- [HI_MPI_VI_SetFlashConfig](#)
- [HI_MPI_VI_GetFlashConfig](#)

VI_CHN_LUM_S

[Description]

Defines the channel luminance statistics.

[Syntax]

```
typedef struct hiVI_CHN_LUM_S
{
    HI_U32 u32Lum;
    HI_U64 u64Pts;
} VI_CHN_LUM_S;
```

[Member]

Member	Description
u32Lum	Luminance statistics
u64Pts	PTS of the obtained frame

[Note]

None

[See Also]

[HI_MPI_VI_GetChnLuma](#)



VI_RAW_DATA_INFO_S

[Description]

Defines raw data attributes.

[Syntax]

```
typedef struct
{
    VIDEO_FRAME_INFO_S stFrame;
}VI_RAW_DATA_INFO_S;
```

[Member]

Member	Description
stFrame	Raw data information

[Note]

- When the raw data is stored in the memory, 16 bits can be allocated for each pixel, the valid pixel is stored in the upper bits, and the pixel format is configured as **PIXEL_FORMAT_RGB_BAYER**. In addition, the memory can be allocated based on actual pixel bit. Each pixel must be stored in a compact way, and the pixel formats **PIXEL_FORMAT_RGB_BAYER_8BPP**, **PIXEL_FORMAT_RGB_BAYER_10BPP**, **PIXEL_FORMAT_RGB_BAYER_12BPP**, **PIXEL_FORMAT_RGB_BAYER_14BPP**, and **PIXEL_FORMAT_RGB_BAYER** are supported.
- During memory allocation, each row must be 16-byte-aligned.

[See Also]

- [VI_RAW_DATA_INFO_S](#)
- [HI_MPI_VI_SendBayerData](#)

VI_DCI_PARAM_S

[Description]

Defines DCI parameters.

During DCI, the picture luminance and contrast are dynamically adjusted based on the luminance distribution of the current picture. The captured picture may be too dark or too bright, or the picture contrast may be low due to environment interference. During DCI, the luminance of the bright picture is decreased, the luminance of the dark picture is increased, the contrast of the picture with small luminance variance is increased, and the contrast of the picture with large luminance variance is decreased.

[Syntax]

```
typedef struct hiVI_DCI_PARAM_S
{
    HI_BOOL      bEnable;
```



```
    HI_U32      u32BlackGain;      /*u32BlackGain: [0, 63]*/
    HI_U32      u32ContrastGain;   /*u32ContrastGain: [0, 63]*/
    HI_U32      u32LightGain;     /*u32LightGain: [0, 63]*/
} VI_DCI_PARAM_S;
```

[Member]

Member	Description
bEnable	DCI enable
u32BlackGain	Dark region gain. A larger value indicates a larger improvement in the dark region contrast. Value range: [0, 63]
u32ContrastGain	Gain of the medium-bright region. A larger value indicates a larger improvement in the contrast of the medium-bright region. Value range: [0, 63]
u32LightGain	Bright region gain. A larger value indicates a larger improvement in the bright region contrast. Value range: [0, 63]

[Note]

During DCI, the luminance and contrast are adjusted automatically. In general, the three gains do not need to be adjusted. If the gains are adjusted inappropriately, the picture may be too bright or too dark, the picture contrast may be too low or too high, some previously invisible details may be added, or some previously visible details are lost. The corresponding MPI can be called to transfer manually configured gains. Then the effect of these gains is overlapped during DCI. The three parameters are independent of each other and can be configured separately. The adjustment effect is related to the actual scenario. The effect of adjusting the three gains differs in different scenarios.

[See Also]

- [HI_MPI_VI_SetDCIParam](#)
- [HI_MPI_VI_GetDCIParam](#)

VI_DUMP_ATTR_S

[Description]

Defines the attribute of the data to be dumped.

[Syntax]

```
typedef struct hiVI_DUMP_ATTR_S
{
    VI_DUMP_TYPE_E enDumpType;
    PIXEL_FORMAT_E enPixelFormat;
    CROP_INFO_S    stCropInfo;
```



```
}VI_DUMP_ATTR_S;
```

[Member]

Member	Description
enDumpType	Type of the data to be dumped for the VI device
enPixelFormat	Pixel format of the dump data
stCropInfo	Member used to dump the data in some regions. The effect is equivalent to that of cropping. For details, see chapter 2 "System Control". Width: [VIU_DEV_MIN_WIDTH , VIU_DEV_MAX_WIDTH] Height: [2, VIU_DEV_MAX_HEIGHT]

[Note]

- In **stCropInfo**, the value of (**s32X + u32Width**) cannot be greater than the width of the actual input picture, and the value of (**s32Y + u32Height**) cannot be greater than the height of the actual input picture. Otherwise, the dump operation fails.
- **stCropInfo.stRect** must be 2-pixel-aligned.
- The effectiveness of **stCropInfo.stRect** is checked only when the crop function is enabled.

[See Also]

- [VI_DUMP_TYPE_E](#)
- [PIXEL_FORMAT_E](#)
- [CROP_INFO_S](#)
- [HI_MPI_VI_SetDevDumpAttr](#)
- [HI_MPI_VI_GetDevDumpAttr](#)

VI_DUMP_TYPE_E

[Description]

Defines the type of the data to be dumped for the VI device.

[Syntax]

```
typedef enum hIVI_DUMP_TYPE_E
{
    VI_DUMP_TYPE_RAW    =  0,
    VI_DUMP_TYPE_IR     =  1,
    VI_DUMP_TYPE_YUV    =  2,
    VI_DUMP_TYPE_RGB    =  3,
    VI_DUMP_TYPE_BUTT
} VI_DUMP_TYPE_E;
```

[Member]



Member	Description
VI_DUMP_TYPE_RAW	The data to be dumped is raw data.
VI_DUMP_TYPE_IR	The data to be dumped is IR data. This member is not supported by the Hi3516A and Hi3519 V100.
VI_DUMP_TYPE_YUV	The data to be dumped is YUV data. This member is not supported.
VI_DUMP_TYPE_RGB	The data to be dumped is RGB data. This member is not supported.

[Note]

None

[See Also]

- [VI_DUMP_ATTR_S](#)
- [HI_MPI_VI_SetDevDumpAttr](#)
- [HI_MPI_VI_GetDevDumpAttr](#)

VI_WDR_ATTR_S

[Description]

Defines WDR attributes.

[Syntax]

```
typedef struct hivI_ISP_WDR_ATTR_S
{
    WDR_MODE_E enWDRMode;
    HI_BOOL     bCompress;
}VI_WDR_ATTR_S;
```

[Member]

Member	Description
enWDRMode	WDR working modes, including the frame mode, line mode, and non-WDR mode
bCompress	Whether to compress. HI_FALSE indicates non-compression, and HI_TRUE indicates compression.

[Note]

None

[See Also]

- [HI_MPI_VI_SetWDRAttr](#)



- [HI_MPI_VI_GetWDRAttr](#)

VI_MOD_PARAM_S

[Description]

Defines the parameters of the VI module.

[Syntax]

```
typedef struct hiVI_MOD_PARAM_S
{
    HI_BOOL bLumaExtendEn;
    HI_BOOL bContrastModeEn;
}VI_MOD_PARAM_S;
```

[Member]

Member	Description
bLumaExtendEn	When bLumaExtendEn is HI_FALSE , the adjustment range of the luminance is -32 to +32. When bLumaExtendEn is HI_TRUE , the adjustment range of the luminance is from -128 to +128. Note: The adjustment range is not the value range of the parameter. For details about the value range, see VI_CSC_ATTR_S .
bContrastModeEn	When bContrastModeEn is HI_FALSE , the luminance changes according to the contrast adjustment. When bContrastModeEn is HI_TRUE , the luminance does not change according to the contrast adjustment.

[Note]

None

[See Also]

- [HI_MPI_VI_SetModParam](#)
- [HI_MPI_VI_GetModParam](#)

VI_DIS_CONFIG_S

[Description]

Defines the DIS configuration information.

[Syntax]

```
typedef struct hiVI_DIS_CONFIG_S
{
    VI_DIS_ACCURACY_E      enAccuracy;
    VI_DIS_CAMERA_MODE_E   enCameraMode;
```



```
    VI_DIS_MOTION_TYPE_E     enMotionType;
    HI_U32                  u32FixLevel;
    HI_U32                  u32RollingShutterCoef;
    HI_U32                  u32BufNum;
    HI_U32                  u32CropRatio;
    HI_S32                  s32FrameRate;
    HI_BOOL                 bScale;
    HI_U32                  u32DelayFrmNum;
    HI_U32                  u32RetCenterStrength;
    HI_U32                  u32GyroWeight;
}VI_DIS_CONFIG_S;
```

[Member]

Member	Description
enAccuracy	DIS accuracy. There are three levels of DIS accuracy: high, middle, and low.
enCameraMode	DIS scenario mode. There are two modes: normal mode and IPC scenario mode.
enMotionType	Multi-axis multi-degree DIS algorithm. There are seven algorithms in total.
u32FixLevel	DIS strength level. A higher level indicates better DIS effect but higher risk of misjudgment. The level needs to be configured based on the application scenario. Value range: [0, 7]
u32RollingShutterCoef	Parameter used to correct the rolling shutter Value range: [0, 100] Recommended value: 80
u32BufNum	Number of buffers used by the DIS to store pictures. The number of buffers can be increased if frame loss occurs for the DIS output frames. Value range: [5, 8]
u32CropRatio	Cropping ratio of the DIS output picture Value range: [50, 98]
s32FrameRate	Real-time VI output frame rate, which needs to be configured based on the VI output frame rate Value range: [1, 120]
bScale	Whether to scale up the output picture after cropping. The current DIS function allows the user to choose whether to scale up the output picture after cropping. If the user does not want to use the scale-up function of DIS, the scale-up operation can be performed in the back-end VPSS. If bScale is false , the output picture width/height is the input width/height multiplied by u32CropRatio , and is 4-pixel-



Member	Description
	aligned.
u32DelayFrmNum	Number of delayed output frames. The current DIS algorithm supports the delay of one output frame, which improves the anti-jitter effect. In consideration of the real-time performance of the IPC, it is recommended that u32DelayFrmNum be set to 0 in the IPC scenario, and be set to 1 in the DV scenario. Value range: [0, 1]
u32RetCenterStrength	Strength of pulling the picture back to the center point. When the picture is rotated, the picture can be quickly pulled back to the center point by setting this parameter. A smaller value indicates greater strength of pulling the picture back to the center point. Value range: [0, 100]
u32GyroWeight	Weight value of dependence on the gyroscope data in the mixed algorithm. A larger weight value indicates a higher probability of using the gyroscope data. However, the anti-jitter effect of the gyroscope algorithm is weaker than that of the software algorithm. Therefore, both the anti-jitter effect and anti-interference capability need to be considered when this parameter is configured. This member should be set to 0 when the 8dof_hard or soft algorithm is used. Value range: [0, 100]

[Note]

When the video input resolution is less than 1920 x 1080 (for example, 1280 x 720), the minimum value of **u32CropRatio** is **80**. If **u32CropRatio** is set to a value less than 80, a parameter error will be returned.

[See Also]

- [HI_MPI_VI_SetDISConfig](#)
- [HI_MPI_VI_GetDISConfig](#)

VI_DIS_ATTR_S

[Description]

Defines the DIS attributes.

[Syntax]

```
typedef struct hiVI_DIS_ATTR_S
{
    HI_BOOL             bEnable;
    HI_U32              u32MovingSubjectLevel;
    HI_U32              u32NoMovementLevel;
    HI_U32              u32Timelag;
    VI_DIS_ANGLE_TYPE_E enAngleType;
```



```
    HI_U32          u32Vangle;
    HI_BOOL         bStillCrop;
}VI_DIS_ATTR_S;
```

[Member]

Member	Description
bEnable	DIS enable
u32MovingSubjectLevel	Level for determining whether the object is moving. A higher level indicates weaker DIS effect. A smaller value indicates higher stability during the motion process but a higher probability of offset occurrence. A larger value indicates that the background jitter amplitude is large during the motion process but the picture offset is alleviated. The level needs to be configured based on the application scenario. Value range: [0, 7]
u32NoMovementLevel	Level for determining whether the camera is static. A higher level indicates weaker DIS effect. A smaller value indicates better DIS effect but a higher probability of offset occurrence. A larger value indicates that the picture offset is alleviated. The level needs to be configured based on the application scenario. Value range: [0, 7]
u32TimeLag	Time difference between the frame start time and the time for capturing the gyroscope data
enAngleType	Type of the lens view angle
u32Vangle	Lens view angle Value range: [100, 1380]
bStillCrop	Enable switch that disables the DIS function but ensures that the picture is still output based on the cropping ratio

[Note]

- The recommended setting of **enAngleType** is **VI_DIS_ANGLE_TYPE_DIAGONAL**.
- The configured value of **u32Vangle** is an integer. However, the actual value is a floating number with one decimal place. For example, to set the angle to 80.7°, you need to set **u32Vangle** to **807**. The value needs to be fine-tuned based on the anti-jitter effect.
- **u32TimeLag** needs to be configured when the gyroscope is used. The value of **u32TimeLag** is generally the time of a frame. For example, in the 1080p60fps scenario, **u32TimeLag** is generally set to **16666** μs. The value needs to be fine-tuned based on the anti-jitter effect.

[See Also]

- [HI_MPI_VI_SetDISAttr](#)



- [HI_MPI_VI_GetDISAttr](#)

VI_DIS_ANGLE_TYPE_E

[Description]

Defines the type of the lens view angle used in DIS.

[Syntax]

```
typedef enum hiVI_DIS_ANGLE_TYPE_E
{
    VI_DIS_ANGLE_TYPE_HORIZONTAL,
    VI_DIS_ANGLE_TYPE_VERTICAL,
    VI_DIS_ANGLE_TYPE_DIAGONAL,
    VI_DIS_ANGLE_TYPE_BUTT,
}VI_DIS_ANGLE_TYPE_E;
```

[Member]

Member	Description
VI_DIS_ANGLE_TYPE_HORIZONTAL	Horizontal view angle
VI_DIS_ANGLE_TYPE_VERTICAL	Vertical view angle
VI_DIS_ANGLE_TYPE_DIAGONAL	Diagonal view angle

[Note]

None

[See Also]

- [HI_MPI_VI_SetDISConfig](#)
- [HI_MPI_VI_GetDISConfig](#)

VI_DIS_ACCURACY_E

[Description]

Defines the DIS accuracy.

[Syntax]

```
typedef enum hiVI_DIS_ACCURACY_E
{
    VI_DIS_ACCURACY_HIGH,
    VI_DIS_ACCURACY_MIDDLE,
    VI_DIS_ACCURACY_LOW,
    VI_DIS_ACCURACY_BUTT
}VI_DIS_ACCURACY_E;
```



[Member]

Member	Description
VI_DIS_ACCURACY_HIGH	High-accuracy DIS. The DIS effect is good and the requirement on the performance is high.
VI_DIS_ACCURACY_MIDDLE	Middle-accuracy DIS. The DIS effect is common.
VI_DIS_ACCURACY_LOW	Low-accuracy DIS. The DIS effect is weak but the performance consumption is low.

[Note]

None

[See Also]

- [HI_MPI_VI_SetDISConfig](#)
- [HI_MPI_VI_GetDISConfig](#)

VI_DIS_CAMERA_MODE_E

[Description]

Defines the DIS scenario mode.

[Syntax]

```
typedef enum hivi_DIS_CAMERA_MODE_E
{
    VI_DIS_CAMERA_MODE_NORMAL,
    VI_DIS_CAMERA_MODE_IPC,
    VI_DIS_CAMERA_MODE_BUTT
}VI_DIS_CAMERA_MODE_E;
```

[Member]

Member	Description
VI_DIS_CAMERA_MODE_NORMAL	Normal mode (used in the normal scenario)
VI_DIS_CAMERA_MODE_IPC	IPC scenario mode (used in the IPC scenario)

[Note]

None

[See Also]

- [HI_MPI_VI_SetDISConfig](#)
- [HI_MPI_VI_GetDISConfig](#)



VI_DIS_MOTION_TYPE_E

[Description]

Defines the multi-DOF DIS algorithm.

[Syntax]

```
typedef enum hiVI_DIS_MOTION_TYPE_E
{
    VI_DIS_MOTION_4DOF_SOFT,
    VI_DIS_MOTION_6DOF_SOFT,
    VI_DIS_MOTION_6DOF_HYBRID,
    VI_DIS_MOTION_8DOF_HARD,
    VI_DIS_MOTION_BUTT,
}VI_DIS_MOTION_TYPE_E;
```

[Member]

Member	Description
VI_DIS_MOTION_4DOF_SOFT	4-axis software algorithm without using the gyroscope
VI_DIS_MOTION_6DOF_SOFT	6-axis software algorithm without using the gyroscope
VI_DIS_MOTION_6DOF_HYBRID	6-axis hybrid algorithm (software algorithm+gyroscope)
VI_DIS_MOTION_8DOF_HARD	8-axis gyroscope data algorithm without using the software algorithm

[Note]

None

[See Also]

- [HI_MPI_VI_SetDISConfig](#)
- [HI_MPI_VI_GetDISConfig](#)

VI_DIS_GYRO_DATA_S

[Description]

Defines the structure of the gyroscope data used by the DIS.

[Syntax]

```
typedef struct hiVI_DIS_GYRO_DATA_S
{
    HI_DOUBLE      *pdRotX;
    HI_DOUBLE      *pdRotY;
```



```
    HI_DOUBLE      *pdRotZ;
    HI_S64         *ps64Time;
    HI_U32         u32Num;
}VI_DIS_GYRO_DATA_S;
```

[Member]

Member	Description
pdRotX	Pointer to the rotation angular velocity array of the X axis
pdRotY	Pointer to the rotation angular velocity array of the Y axis
pdRotZ	Pointer to the rotation angular velocity array of the Z axis
ps64Time	Pointer to the time stamp array
u32Num	Number of available gyroscope data segments

[Note]

None

[See Also]

[HI_MPI_VI_RegisterDISCallback](#)

VI_DIS_CALLBACK_S

[Description]

Defines the structure that registers the callback functions with the DIS.

[Syntax]

```
typedef struct hiVI_DIS_CALLBACK_S
{
    HI_S32 (*pfnGetGyroDataCallback) (HI_U64 u64BeginPts, HI_U64 u64EndPts,
                                       VI_DIS_GYRO_DATA_S* pstDISGyroData);
    HI_S32 (*pfnReleaseGyroDataCallback) (HI_VOID);
}VI_DIS_CALLBACK_S;
```

[Member]

Member	Description
pfnGetGyroDataCallback	Pointer to the callback function for obtaining the gyroscope data
pfnReleaseGyroDataCallback	Pointer to the callback function for releasing the gyroscope data after the data is used

[Note]



In the callback function for obtaining the gyroscope data, **pfnGetGyroDataCallback** needs to allocate memory for the gyroscope data and release the memory after DIS uses the data.

[See Also]

[HI_MPI_VI_RegisterDISCallback](#)

3.6 Error Codes

Table 3-4 describes the error codes of VI APIs.

Table 3-4 Error codes of VI APIs

Error Code	Macro Definition	Description
0xA0108001	HI_ERR_VI_INVALID_DEVID	The VI device ID is invalid.
0xA0108002	HI_ERR_VI_INVALID_CHNID	The VI channel ID is invalid.
0xA0108003	HI_ERR_VI_INVALID_PARA	The VI parameter is invalid.
0xA0108006	HI_ERR_VI_INVALID_NULL_PTR	The pointer of the input parameter is null.
0xA0108007	HI_ERR_VI_FAILED_NOTCONFIG	The attributes of the video device are not set.
0xA0108008	HI_ERR_VI_NOT_SUPPORT	The operation is not supported.
0xA0108009	HI_ERR_VI_NOT_PERM	The operation is forbidden.
0xA010800C	HI_ERR_VI_NOMEM	The memory fails to be allocated.
0xA010800E	HI_ERR_VI_BUF_EMPTY	The VI buffer is empty.
0xA010800F	HI_ERR_VI_BUF_FULL	The VI buffer is full.
0xA0108010	HI_ERR_VI_SYS_NOTREADY	The VI system is not initialized.
0xA0108012	HI_ERR_VI_BUSY	The VI system is busy.
0xA0108040	HI_ERR_VI_FAILED_NOTENABLE	The VI device or VI channel is not enabled.
0xA0108041	HI_ERR_VI_FAILED_NOTDISABLE	The VI device or VI channel is not disabled.
0xA0108042	HI_ERR_VI_FAILED_CHNOTDISABLE	The VI channel is not disabled.
0xA0108043	HI_ERR_VI_CFG_TIMEOUT	The video attribute configuration times out.
0xA0108044	HI_ERR_VI_NORM_UNMATCH	Mismatch occurs.
0xA0108045	HI_ERR_VI_INVALID_WAYID	The video channel ID is invalid.
0xA0108046	HI_ERR_VI_INVALID_PHYCHNID	The physical video channel ID is invalid.
0xA0108047	HI_ERR_VI_FAILED_NOTBIND	The video channel is not bound.
0xA0108048	HI_ERR_VI_FAILED_BINDED	The video channel is bound.



Error Code	Macro Definition	Description
0xA0108049	HI_ERR_VI_DIS_PROCESS_FAIL	The DIS fails to run.



Contents

4 VO	4-1
4.1 Overview	4-1
4.2 Important Concepts	4-1
4.3 API Reference	4-5
4.4 Data Structures	4-80
4.5 Error Codes	4-100



Figures

Figure 4-1 Concepts related to video layer attributes	4-87
Figure 4-2 Video layer scenario A (cluster memory).....	4-88
Figure 4-3 Video layer scenario B (non-cluster memory)	4-88
Figure 4-4 Principle of partial zoom-in	4-97



Tables

Table 4-1 Supported display devices, graphics layers, and cursor layers	4-1
Table 4-2 Device specifications of the Hi3516A/Hi3518E V200, and Hi3519 V100	4-2
Table 4-3 Video layer specifications of the Hi3516A/Hi3518E V200, and Hi3519 V100	4-2
Table 4-4 Error codes of VO MPIS	4-100



4 VO

4.1 Overview

The video output unit (VOU) reads video and graphics data from certain positions in the memory and then outputs the data by using corresponding display devices. [Table 4-1](#) describes the display devices, graphics layers, and cursor layers.

Table 4-1 Supported display devices, graphics layers, and cursor layers

Chip	Supported Display Device		Graphics Layer	Hardware Cursor Layer
	HD Device	SD Device		
Hi3516A	-	One SD device (DSD) The SD video layer supports a maximum of 32 pictures.	One graphics layer (G0)	-
	-	One SD device (DSD) The SD video layer supports a maximum of 8 pictures.		-
Hi3518E V200	-	One SD device (DSD) The SD video layer supports a maximum of 8 pictures.	One graphics layer (G0)	-
	-	One SD device (DSD) The SD video layer supports a maximum of 8 pictures.		-
Hi3519 V100	-	One SD device (DSD) The SD video layer supports a maximum of 8 pictures.	One graphics layer (G0)	-
	-	One SD device (DSD) The SD video layer supports a maximum of 8 pictures.		-

4.2 Important Concepts

- HD and SD display devices

In the SDK, each HD device is named DHDx, and each SD device is named DSDx. x is numbered from 0. For example, HD display device 0 is DHD0, and SD display device 0 is DSD0. The HD and SD display devices are called HD and SD devices for short respectively. The HD device may be bound to two video layers at the same time.



- Video layer

The video layer that is always bound to an HD device is called VHDx, and the video layer that is always bound to an SD device is called VSDx. The video layer that can be dynamically bound is called PIP. For details about the supported display devices, see [Table 4-1](#). For details about the function differences between HD and SD devices, see [Table 4-2](#). For details about the function differences among the VHD, VSD, and PIP video layers, see [Table 4-3](#).

- Channel

In the SDK, channels are managed by video layers. The channels at all video layers are independent of each other. The channel IDs are independent of each layer. For example, in the electronic magnification scenario, channel 0 at VHD0 displays the enlarged picture, and channel 0 at the PIP layer displays the pictures in all scenarios. That is, the two channels are irrelevant.

When sys_bind is called, the binding relationship needs to be set by using the video layer and channel ID. When the video layer is written back, the binding relationship needs to be set by using the ID of channel 0 and the ID of the device where the writeback contents are located.

Table 4-2 Device specifications of the Hi3516A/Hi3518E V200, and Hi3519 V100

Chip		Maximum Output Timing	Output Interface	Display After Overlay and Combination
Hi3516A	SD	1920 x 1080@60	BT.1120/CVBS/ BT.656	Video graphic subsystem (VGS) overlay is supported.
Hi3518E V200	SD	PAL/NTSC	BT.656/LCD	VGS overlay is supported.
Hi3519 V100	SD	1920 x 1080@60	BT.1120/CVBS/ BT.656/LCD	VGS overlay is supported.

Table 4-3 Video layer specifications of the Hi3516A/Hi3518E V200, and Hi3519 V100

Chip	Dynamical Binding	Scaling	Number of Supported Channels		CSC
			For Software	For Hardware	
Hi3516A	VSD	Not supported (VSD is always bound to the SD device)	Not supported	32	1
Hi3518E V200	VSD	Not supported (VSD is always bound to the	Not supported	8	1



Chip	Dynamical Binding	Scaling	Number of Supported Channels		CSC
			For Software	For Hardware	
		SD device)			
Hi3519 V100	VSD	Not supported (VSD is always bound to the SD device)	Not supported	8	1 Supported

- Scaling and displaying

Each VHD or PIP layer has x channels. For details about the value of x , see [Table 4-3](#). The channels at video layers cannot be overlapped. As the channels at HD video layers do not support scaling, the external video processing subsystem (VPSS) is used for scaling. After the source pictures in the video input (VI) channel or video decoding (VDEC) channel are scaled by the VPSS, the pictures are output to a VO channel. If the output picture size is greater than the region size, the picture is cropped. The entire VHD can be zoomed in but cannot be cropped. The PIP video layer cannot be zoomed in or cropped.

The VSD has x channels. For details about the value of x , see [Table 4-1](#). The channels of at the VSD video layer are scaled by using the VGS. The pictures in the channels are overlaid as one picture and output to the display device. The VSD can be cropped but cannot be scaled. For details, see [Table 4-2](#).



CAUTION

- Cropping indicates that when the width or height of the canvas size (**stImageSize**) is greater than the width or height of the display device resolution (**stDispRect**), the canvas is cropped to fit into the display device resolution.
- Scaling indicates that when the width or height of the display device resolution (**stDispRect**) is greater than the width or height of the canvas (**stImageSize**), the canvas is zoomed in to fit into the display device resolution.
- Channel priority
 - Setting the channel priority has no effect for HD devices. The channel position is uniquely specified by the related video layer. That is, the channel data is displayed at the VHDx or PIP layer.
 - The SD device allows multiple channels to output pictures at the same time. The pictures are overlaid based on the channel priority. When the channel pictures are overlapped, the picture with the highest priority is displayed on the top. If multiple channels have the same priority, the channel with the largest ID has the highest priority by default.
- PIP overlay



The PIP function enables the picture at the PIP layer of the HD device to be overlaid with the picture of the original video layer. The original video layer of the HD device VHD layer and PIP layer do not support channel overlay. The video layer display priority can be configured to determine the output position of the video layer. The picture at the video layer with a higher priority is displayed on the top.

- Resolution

Resolutions are categorized into:

- Device resolution. It is the number of active output pixels of a device, which depends on the device timing.
- Display resolution. It is the valid display region on a display device.
- Picture resolution. It is the number of active pixels of a picture.

- Partial enlargement

- The SD device supports partial enlargement on a picture. The enlarged original area is clipped from the original picture, and the size of the enlarged target area is the size of the display channel.
- The HD or PIP layer can partially enlarge the picture of a channel only when it works with the VPSS, because only the VPSS supports scaling.
- At most 16x partial enlargement is supported in the horizontal or vertical direction.

- Binding the graphics layer

The chip allows a graphics layer or cursor layer to be bound to a device.

The Hi3516A/Hi3518E V200/Hi3519 V100 supports one graphics layer (G0) that is always bound to DSD0.

Note

- HC0 can be bound to only one device at the same time. For example, if HC0 is bound to DHD0, it cannot be bound to DHD1 or DSD1.
- HC0 can be bound to DHD0, DHD1, or DSD0. However, the device to be bound cannot be switched in real time. You must disable the bound graphics layer before switching the device to be bound. For example, if HC0 is bound to DHD0 and you want to bind HC0 to DHD1, you must disable HC0 and then bind HC0 to DHD1.

- Binding the video layer

The PIP video layer can be bound to an HD device and is overlaid with the original video layer of the HD device. In this case, the PIP layer is an overlay layer of the HD device. If the PIP layer is bound to a device and you want to bind the PIP layer to another device, you must disable the PIP layer before switching the device to be bound. For example, if the PIP layer is bound to DHD0 and you want to bind the PIP layer to DHD1, you must disable the PIP layer, unbind the PIP layer from DHD0, and then bind the PIP layer to DHD1. The PIP layer is bound to DHD0 by default.

The Hi3516A/Hi3518E V200/Hi3519 V100 VOU does not support the PIP layer.

- Single-picture straight-through mode

The single-picture straight-through mode indicates that data streams are not processed by the VGS. In this way, the VGS transfer operation and the memory requested by the VOU are reduced. The straight-through mode is selected only when all the following conditions are met:

- VO channels are enabled and only the pictures from only one channel are displayed.
- OSD and COVER are not displayed.
- Borders are disabled for VO.
- Anti-flicker is disabled for VO.



- Electronic magnification is disabled for VO.
- The pixel format of the input pictures of the VOU is the same as the configured pixel format.
- When the input picture resolution of the VOU is different from that of the channel Rect, the display buffer length is **0**. If the input picture resolution is greater than the channel Rect, the picture is cropped to fit into the channel Rect; if the input picture resolution is less than the channel Rect, the actual picture is displayed.

Apart from the preceding conditions, one more condition must be met for Hi3518E V200, that is, the VO input pictures cannot be compressed pictures.

If the preceding conditions are not met, the VOU does not allocate memory by default. Therefore, you must call `HI_MPI_VO_SetDispBufLen` to configure the buffer length before enabling the video layer. This ensures that the memory is allocated. Otherwise, the VOU does not output videos.

- Decompression

The Hi3516A/Hi3519 V100 VOU supports the input source pictures that are compressed by 256-byte segment. The VOU can decompress such pictures for displaying. However, the VOU of Hi3518E V200 does not support decompression. When the input source pictures are compressed pictures, the pictures must be decompressed by the VGS module and then be displayed.

- Low-power strategy

The VOU uses the low-power strategy. The VOU clock is disabled after the VOU is loaded or the case is exited. If the VOU registers are manually read or written at this time, the read/write error or suspension may occur.

4.3 API Reference

The VOU supports multiple functions including enabling a VO device or VO channel and transmitting video data to an output channel.

The VOU provides the following MPP programming interfaces (MPIs).

The following are the MPIs related to device operations:

- `HI_MPI_VO_Enable`: Enables a VO device.
- `HI_MPI_VO_Disable`: Disables a VO device.
- `HI_MPI_VO_SetPubAttr`: Sets the public attributes of a VO device.
- `HI_MPI_VO_GetPubAttr`: Obtains the public attributes of a VO device.
- `HI_MPI_VO_CloseFd`: Disables the file descriptors (FDs) of all the VO devices.
- `HI_MPI_VO_SetDevFrameRate`: Sets the frame rate of a device when the user timing is used.
- `HI_MPI_VO_GetDevFrameRate`: Obtains the frame rate of a device.

The following are the MPIs related to video layer operations:

- `HI_MPI_VO_EnableVideoLayer`: Enables a video layer.
- `HI_MPI_VO_DisableVideoLayer`: Disables a video layer.
- `HI_MPI_VO_SetVideoLayerAttr`: Sets the attributes of a video layer.
- `HI_MPI_VO_GetVideoLayerAttr`: Obtains the attributes of the PIP layer.
- `HI_MPI_VO_SetVideoLayerPriority`: Sets the priority of a video layer.



- [HI_MPI_VO_GetVideoLayerPriority](#): Obtains the priority of a video layer.
- [HI_MPI_VO_SetVideoLayerPartitionMode](#): Sets the split mode of a video layer.
- [HI_MPI_VO_GetVideoLayerPartitionMode](#): Obtains the split mode of a video layer.
- [HI_MPI_VO_SetVideoLayerCSC](#): Sets the CSC attributes of a video layer.
- [HI_MPI_VO_GetVideoLayerCSC](#): Obtains the CSC attributes of a video layer.
- [HI_MPI_VO_SetAttrBegin](#): Starts channel attribute settings of a video layer.
- [HI_MPI_VO_SetAttrEnd](#): Ends channel attribute settings of a video layer.
- [HI_MPI_VO_SetPlayToleration](#): Sets the play tolerance of a video layer.
- [HI_MPI_VO_GetPlayToleration](#): Obtains the play tolerance of a video layer.
- [HI_MPI_VO_GetScreenFrame](#): Obtains the output frames of a video layer.
- [HI_MPI_VO_ReleaseScreenFrame](#): Releases the output frames of a video layer.
- [HI_MPI_VO_SetDispBufLen](#): Sets the display buffer length of a video layer.
- [HI_MPI_VO_GetDispBufLen](#): Obtains the display buffer length of a video layer.

The following are the MPIs related to channel operations:

- [HI_MPI_VO_EnableChn](#): Enables a specified VO channel.
- [HI_MPI_VO_DisableChn](#): Disables a specified VO channel.
- [HI_MPI_VO_SetChnAttr](#): Sets the attributes of a specified VO channel.
- [HI_MPI_VO_GetChnAttr](#): Obtains the attributes of a specified VO channel.
- [HI_MPI_VO_SetChnField](#): Sets the frame/field display policy of a specified VO channel
- [HI_MPI_VO_GetChnField](#): Obtains the frame/field display policy of a specified VO channel.
- [HI_MPI_VO_GetChnFrame](#): Obtains frames from a VO channel.
- [HI_MPI_VO_ReleaseChnFrame](#): Releases the buffer for storing the pictures of a VO channel.
- [HI_MPI_VO_SendFrame](#): Sends video pictures to a specified VO channel.
- [HI_MPI_VO_SetChnFrameRate](#): Sets the display frame rate of a specified VO channel.
- [HI_MPI_VO_GetChnFrameRate](#): Obtains the display frame rate of a specified VO channel.
- [HI_MPI_VO_PauseChn](#): Pauses a specified VO channel.
- [HI_MPI_VO_ResumeChn](#): Resumes a specified VO channel.
- [HI_MPI_VO_StepChn](#): Plays the streams in a specified VO channel by frame.
- [HI_MPI_VO_RefreshChn](#): Refreshes a specified VO channel.
- [HI_MPI_VO_ShowChn](#): Shows a specified channel.
- [HI_MPI_VO_HideChn](#): Hides a specified channel.
- [HI_MPI_VO_SetZoomInWindow](#): Sets a VO zoom-in window.
- [HI_MPI_VO_GetZoomInWindow](#): Obtains the parameters of a VO zoom-in window.
- [HI_MPI_VO_GetChnPts](#): Obtains the presentation timestamp (PTS) of the current picture in a specified VO channel.
- [HI_MPI_VO_QueryChnStat](#): Queries the status of a VO channel.
- [HI_MPI_VO_ClearChnBuffer](#): Clears the buffer data in a specified VO channel.
- [HI_MPI_VO_SetChnBorder](#): Sets channel border attributes.
- [HI_MPI_VO_GetChnBorder](#): Obtains channel border attributes.



- [HI_MPI_VO_SetChnReceiveThreshold](#): Sets the display threshold for a VO channel.
- [HI_MPI_VO_GetChnReceiveThreshold](#): Obtains the display threshold for a VO channel.
- [HI_MPI_VO_GetChnRegionLuma](#): Obtains region luminance.

The following are the MPIS related to graphics layer operations:

- [HI_MPI_VO_SetGraphicLayerCSC](#): Sets the effect of the output pictures from a graphics layer.
- [HI_MPI_VO_GetGraphicLayerCSC](#): Obtains the effect of the output pictures from a graphics layer.

HI_MPI_VO_Enable

[Description]

Enables a VO device.

[Syntax]

```
HI_S32 HI_MPI_VO_Enable (VO_DEV VoDev);
```

[Parameter]

Parameter	Description	Input/Output
VoDev	VO device ID	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Value Range of VoDev
Hi3516A/Hi3518E V200/Hi3519 V100	[0, VO_MAX_DEV_NUM)

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]

- As the VO device is not enabled during system initialization, you must enable the VO device before using the video output function.



- Before enabling a VO device, you must configure its public attributes. Otherwise, an error code is returned, indicating that the VO device is not configured.
- To ensure that the startup screen and the operation screen are smoothly switched, you need to check whether the VO device is enabled. If the VO device is enabled, a code indicating success is returned, and the existing timing is used. If you want to change the VO timing configuration, you need to call [HI_MPI_VO_Disable](#) to disable the VO device and then call [HI_MPI_VO_Enable](#) to enable the VO device again.
- For details about each device, see the description of [VO_DEV](#).

[Example]

```
HI_S32 s32Ret;
VO_DEV VoDev = 0;
VO_PUB_ATTR_S stPubAttr;
s32Ret = HI_MPI_VO_GetPubAttr(VoDev, &stPubAttr);
if (s32Ret != HI_SUCCESS)
{
    printf("Get device attributes failed with error code %#x!\n", s32Ret);
    return HI_FAILURE;
}
stPubAttr.u32BgColor = 0xff;
stPubAttr.enIntfType = VO_INTF_VGA;
stPubAttr.enIntfSync = VO_OUTPUT_1280 x 1024_60;
s32Ret = HI_MPI_VO_SetPubAttr(VoDev, &stPubAttr);
if (s32Ret != HI_SUCCESS)
{
    printf("Set device attributes failed with errno %#x!\n", s32Ret);
    return HI_FAILURE;
}
s32Ret = HI_MPI_VO_Enable(VoDev);
if (s32Ret != HI_SUCCESS)
{
    printf("Enable vo dev %d failed with errno %#x!\n", VoDev, s32Ret);
    return HI_FAILURE;
}
s32Ret = HI_MPI_VO_Disable(VoDev);
if (s32Ret != HI_SUCCESS)
{
    printf("Enable vo dev %d failed with errno %#x!\n", VoDev, s32Ret);
    return HI_FAILURE;
}
s32Ret = HI_MPI_VO_CloseFd();
if (s32Ret != HI_SUCCESS)
{
    printf("Some device is not disable with errno %#x!\n", VoDev, s32Ret);
    return HI_FAILURE;
}
```



}

[See Also]

[HI_MPI_VO_Disable](#)

HI_MPI_VO_Disable

[Description]

Disables a VO device.

[Syntax]

```
HI_S32 HI_MPI_VO_Disable(VO_DEV VoDev);
```

[Parameter]

Parameter	Description	Input/Output
VoDev	VO device ID	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Value Range of VoDev
Hi3516A/Hi3518E V200/Hi3519 V100	[0, VO_MAX_DEV_NUM)

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpci.a

[Note]

- Before disabling a VO device, you must disable its video layer.
- If the WBC function is enabled, you must disable it before disabling a VO device.
- If you do not call HI_MPI_VO_Disable after calling HI_MPI_VO_Enable, the VO device is always enabled, and the device attributes that are set next time do not take effect.
- If a device is disabled, it can be enabled only after public attributes are set again.



[Example]

See the example of [HI_MPI_VO_Enable](#).

[See Also]

[HI_MPI_VO_Enable](#)

HI_MPI_VO_SetPubAttr

[Description]

Sets the public attributes of a VO device.

[Syntax]

```
HI_S32 HI_MPI_VO_SetPubAttr(VO_DEV VoDev, const VO_PUB_ATTR_S  
*pstPubAttr);
```

[Parameter]

Parameter	Description	Input/Output
VoDev	VO device ID	Input
pstPubAttr	Pointer to data structure of the public attributes of a VO device Static attribute	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Value Range of VoDev
Hi3516A/Hi3519 V100	[0, VO_MAX_DEV_NUM) Automatic load detection is supported during CVBS output.
Hi3518E V200	[0, VO_MAX_DEV_NUM)

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpci.a

[Note]



- The attributes of a VO device are static attributes. Therefore, you must set the attributes before calling [HI_MPI_VO_Enable](#).
- For details about each device, see the description of [VO_DEV](#).
- For details about how to use the attributes of a VO device, see the description of [VO_PUB_ATTR_S](#).

[Example]

See the example of [HI_MPI_VO_Enable](#).

[See Also]

[HI_MPI_VO_GetPubAttr](#)

[HI_MPI_VO_GetPubAttr](#)

[Description]

Obtains the public attributes of a VO device.

[Syntax]

```
HI_S32 HI_MPI_VO_GetPubAttr(VO_DEV VoDev, VO_PUB_ATTR_S *pstPubAttr);
```

[Parameter]

Parameter	Description	Input/Output
VoDev	VO device ID	Input
pstPubAttr	Pointer to the data structure of the public attributes of a VO device	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Value Range of VoDev
Hi3516A/Hi3518E V200/Hi3519 V100	[0, VO_MAX_DEV_NUM)

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpvi.a



[Note]

You are advised to obtain the attributes of a VO device before setting public attributes. This enables you to set only the items to be changed.

[Example]

See the example of [HI_MPI_VO_Enable](#).

[See Also]

[HI_MPI_VO_SetPubAttr](#)

HI_MPI_VO_CloseFd

[Description]

Disables the FDs of all the VO devices.

[Syntax]

```
HI_S32 HI_MPI_VO_CloseFd(HI_VOID);
```

[Parameter]

None

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]

Before calling this MPI, you must disable all devices.

[Example]

See the example of [HI_MPI_VO_Enable](#).

[See Also]

[HI_MPI_VO_Enable](#)

HI_MPI_VO_SetDevFrameRate

[Description]

Sets the frame rate of a device when the user timing is used.

[Syntax]



```
HI_S32 HI_MPI_VO_SetDevFrameRate(VO\_DEV VoDev, HI_U32 u32FrameRate);
```

[Parameter]

Parameter	Description	Input/Output
VoDev	VO device ID	Input
u32FrameRate	Frame rate of a device The frame rate must be 25, 30, 50, or 60. The default value is 60.	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Value Range of VoDev
Hi3516A/Hi3518E V200/Hi3519 V100	[0, VO_MAX_DEV_NUM)

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]

- This MPI can be called only when the user timing is used.
- This MPI must be called after [HI_MPI_VO_SetPubAttr](#) is called and before [HI_MPI_VO_Enable](#) is called.

[Example]

None

[See Also]

[HI_MPI_VO_GetDevFrameRate](#)

HI_MPI_VO_GetDevFrameRate

[Description]

Obtains the frame rate of a device.

[Syntax]



```
HI_S32 HI_MPI_VO_GetDevFrameRate(VO_DEV VoDev, HI_U32 *pu32FrameRate);
```

[Parameter]

Parameter	Description	Input/Output
VoDev	VO device ID	Input
pu32FrameRate	u32 pointer	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Value Range of VoDev
Hi3516A/Hi3518E V200/Hi3519 V100	[0, VO_MAX_DEV_NUM)

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]

None

[Example]

None

[See Also]

[HI_MPI_VO_SetDevFrameRate](#)

HI_MPI_VO_EnableVideoLayer

[Description]

Enables a video layer.

[Syntax]

```
HI_S32 HI_MPI_VO_EnableVideoLayer (VO_LAYER VoLayer);
```

[Parameter]



Parameter	Description	Input/Output
VoLayer	VO layer ID	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Value Range of VoLayer
Hi3516A/Hi3518E V200/Hi3519 V100	[0, VO_MAX_LAYER_NUM)

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]

- Before enabling a video layer, enable the device that is bound to the video layer.
- Configure a video layer before enabling it.
- For details about each video layer, see the description of [VO_LAYER](#).

[Example]

```
HI_S32 s32Ret;
VoLayer VoLayer = 0;
RECT_S stRect;
VO_VIDEO_LAYER_ATTR_S stLayerAttr;

s32Ret = HI_MPI_VO_GetVideoLayerAttr(VoLayer, &stLayerAttr);
if (s32Ret != HI_SUCCESS)
{
    printf("Get video layer attributes failed with errno %#x!\n", s32Ret);
    return HI_FAILURE;
}
stLayerAttr.stDispRect.s32X      = 0;
stLayerAttr.stDispRect.s32Y      = 0;
stLayerAttr.stDispRect.u32Width  = 720;
stLayerAttr.stDispRect.u32Height = 576;
```



```
stLayerAttr.stImageSize.u32Width = 720;
stLayerAttr.stImageSize.u32Height = 576;
stLayerAttr.u32DispFrmRt = 25;
stLayerAttr.enPixelFormat = PIXEL_FORMAT_YUV_SEMIPLANAR_422;
stLayerAttr.bDoubleFrame = HI_TRUE;
stLayerAttr.bClusterMode = HI_FALSE;

s32Ret = HI_MPI_VO_SetVideoLayerAttr(VoLayer, &stLayerAttr);
if (s32Ret != HI_SUCCESS)
{
    printf("Set video layer attributes failed with errno %#x!\n", s32Ret);
    return HI_FAILURE;
}

s32Ret = HI_MPI_VO_EnableVideoLayer(VoLayer);
if (s32Ret != HI_SUCCESS)
{
    printf("Enable video layer failed with errno %#x!\n", s32Ret);
    return HI_FAILURE;
}

s32Ret = HI_MPI_VO_DisableVideoLayer(VoLayer);
if (s32Ret != HI_SUCCESS)
{
    printf("Disable video layer failed with errno %#x!\n", s32Ret);
    return HI_FAILURE;
}
```

[See Also]

[HI_MPI_VO_DisableVideoLayer](#)

HI_MPI_VO_DisableVideoLayer

[Description]

Disables a video layer.

[Syntax]

```
HI_S32 HI_MPI_VO_DisableVideoLayer(VO\_LAYER VoLayer);
```

[Parameter]

Parameter	Description	Input/Output
VoLayer	VO layer ID	Input



[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Value Range of VoLayer
Hi3516A/Hi3518E V200/Hi3519 V100	[0, VO_MAX_LAYER_NUM)

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]

- Before disabling a video layer, disable all the channels at the video layer.
- If you do not release the video buffer (VB) that stores the pictures obtained from the VO channel when disabling a video layer, the error code HI_ERR_VB_BUSY is returned, indicating that the VB created by the VOU is not released. This error typically occurs when you obtain screen pictures but the VB is not released.
- If the video layer is disabled and the WBC data source is in VO_WBC_DATASOURCE_VIDEO mode, ensure that the WBC function is disabled.

[Example]

See the example of [HI_MPI_VO_EnableVideoLayer](#).

[See Also]

[HI_MPI_VO_EnableVideoLayer](#)

HI_MPI_VO_SetVideoLayerAttr

[Description]

Sets the attributes of a video layer.

[Syntax]

```
HI_S32 HI_MPI_VO_SetVideoLayerAttr(VO_LAYER VoLayer, const  
VO_VIDEO_LAYER_ATTR_S *pstLayerAttr);
```

[Parameter]



Parameter	Description	Input/Output
VoLayer	VO video layer	Input
pstLayerAttr	Pointer to the attributes of a video layer	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Value Range of VoLayer
Hi3516A/Hi3518E V200/Hi3519 V100	[0, VO_MAX_LAYER_NUM)

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]

- The attributes of a video layer can be set only when the video layer is disabled.
- Ensure that the video layer is bound to a device and the device is enabled before setting the video layer attributes. This requirement applies to the PIP layer that can be dynamically bound. If the PIP layer is unbound from the device, an error indicating no binding relationship occurs when this MPI is called.
- For details about how to use the attributes of the video layer, see the description of [VO_VIDEO_LAYER_ATTR_S](#).

[Example]

See the example of [HI_MPI_VO_EnableVideoLayer](#).

[See Also]

[HI_MPI_VO_GetVideoLayerAttr](#)

HI_MPI_VO_GetVideoLayerAttr

[Description]

Obtains the attributes of the PIP layer.

[Syntax]



```
HI_S32 HI_MPI_VO_GetVideoLayerAttr(VO_LAYER VoLayer,  
VO_VIDEO_LAYER_ATTR_S *pstLayerAttr);
```

[Parameter]

Parameter	Description	Input/Output
VoLayer	VO layer ID	Input
pstLayerAttr	Pointer to the data structure of the attributes of a video layer	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Value Range of VoLayer
Hi3516A/Hi3518E V200/Hi3519 V100	[0, VO_MAX_LAYER_NUM)

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]

- The default attributes of a video layer can be obtained even [HI_MPI_VO_EnableVideoLayer](#) and [HI_MPI_VO_SetVideoLayerAttr](#) are not called.
- You are advised to obtain the attributes of a video layer by calling [HI_MPI_VO_GetVideoLayerAttr](#) before setting its attributes.

[Example]

See the example of [HI_MPI_VO_EnableVideoLayer](#).

[See Also]

[HI_MPI_VO_SetVideoLayerAttr](#)

HI_MPI_VO_SetVideoLayerPriority

[Description]

Sets the priority of a video layer.



[Syntax]

```
HI_S32 HI_MPI_VO_SetVideoLayerPriority(VO_LAYER VoLayer, HI_U32  
u32Priority);
```

[Parameter]

Parameter	Description	Input/Output
VoLayer	VO layer ID	Input
u32Priority	Priority of a video layer	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Description
Hi3516A/Hi3518E V200/Hi3519 V100	The Hi3516A/Hi3518E V200/Hi3519 V100 does not support this MPI.

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]

- Before calling this MPI, ensure that the video layer is bound to a device.
- When a video layer is bound to a device, a default priority is set for the video layer. The default priority is set from high to low. To be specific, the default priority of the video layer is the highest priority.
- If **VoLayer** is bound to **VoDev** and the priority **u32Priority** to be set for **VoLayer** has been set for **VoLayer_old**, **u32Priority** is set for **VoLayer**, and the default priority of **VoLayer** is set for **VoLayer_old**. The following example assumes that the PIP layer with priority 1 and VHD1 with priority 0 are bound to DHD1. If you set the priority of the PIP layer to 0 by calling this MPI, priority conflict occurs. In this case, the priority to be set prevails. To be specific, the priority of the PIP layer changes to 0, and the priority of VHD1 changes to the original priority 1 of the PIP layer.

[Example]

None



[See Also]

[HI_MPI_VO_GetVideoLayerPriority](#)

HI_MPI_VO_GetVideoLayerPriority

[Description]

Obtains the priority of a video layer.

[Syntax]

```
HI_S32 HI_MPI_VO_GetVideoLayerPriority(VO\_LAYER VoLayer, HI_U32  
*pu32Priority);
```

[Parameter]

Parameter	Description	Input/Output
VoLayer	VO layer ID	Input
pu32Priority	Pointer to a priority of a video layer	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Value Range of VoLayer
Hi3516A/Hi3518E V200/Hi3519 V100	The Hi3516A/Hi3518E V200/Hi3519 V100 does not support this MPI.

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]

Before calling this MPI, ensure that the video layer is bound to a device.

[Example]

None

[See Also]



HI_MPI_VO_SetVideoLayerPriority

HI_MPI_VO_SetVideoLayerPartitionMode

[Description]

Sets the split mode of a video layer.

[Syntax]

```
HI_S32 HI_MPI_VO_SetVideoLayerPartitionMode(VO_LAYER VoLayer,  
VO_PART_MODE_E enPartMode);
```

[Parameter]

Parameter	Description	Input/Output
VoLayer	VO layer ID	Input
enPartMode	Split mode of a video layer	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Description
Hi3516A/Hi3518E V200/Hi3519 V100	The Hi3516A/Hi3518E V200/Hi3519 V100 does not support this MPI.

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h, hiDefines.h
- Library file: libmpi.a

[Note]

- Before calling this MPI, ensure that the video layer is disabled.
- The split mode of a video layer indicates the mode of configuring regions when a picture is configured for a display device. In VO_PART_MODE_MULTI mode, the number of display information groups to be configured is the same as the number of picture regions. In VO_PART_MODE_SINGLE mode, only an information group is configured no matter there is a single-region picture or multi-region pictures.

If a video layer is processed in VO_PART_MODE_MULTI mode, the maximum number of display channels is restricted by hardware. The VHD displays at most 16 pictures,



whereas the VSD or VPIP video layer displays only one picture. If a video layer is processed in VO_PART_MODE_SINGLE mode, regions are combined by the VGS, and a maximum of 64 pictures are supported.

- In VO_PART_MODE_SINGLE mode, the video layer channel cannot be bound to the system binding source of the VPSS in automatic mode, because the regions of the video layer are combined by the VGS.
- When the system is initialized, the default split mode of the VHD and PIP layer is VO_PART_MODE_MULTI, and the default split mode of the VSD device video layer is VO_PART_MODE_SINGLE.

[Example]

None

[See Also]

[HI_MPI_VO_GetVideoLayerPartitionMode](#)

HI_MPI_VO_GetVideoLayerPartitionMode

[Description]

Obtains the split mode of a video layer.

[Syntax]

```
HI_S32 HI_MPI_VO_GetVideoLayerPartitionMode(VO_LAYER VoLayer,  
VO_PART_MODE_E *penPartMode);
```

[Parameter]

Parameter	Description	Input/Output
VoLayer	VO layer ID	Input
penPartMode	Pointer to the split mode of a video layer	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Description
Hi3516A/Hi3518E V200/Hi3519 V100	The Hi3516A/Hi3518E V200/Hi3519 V100 does not support this MPI.



[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]

None

[Example]

None

[See Also]

[HI_MPI_VO_SetVideoLayerPartitionMode](#)

HI_MPI_VO_SetVideoLayerCSC

[Description]

Sets the effect of the output pictures from a device.

[Syntax]

```
HI_S32 HI_MPI_VO_SetVideoLayerCSC(VO_LAYER VoLayer, const VO_CSC_S
*pstVideoCSC);
```

[Parameter]

Parameter	Description	Input/Output
VoLayer	VO layer ID	Input
pstDevCSC	Pointer to the data structure of the picture output effect	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Description
Hi3516A	The value range of VoLayer is [0, VO_MAX_LAYER_NUM). The value range of u32Luma , u32Contrast , u32Hue , or u32Satuature is [0, 100], and the default value is 50. The default value of enCscMatrix is an identity matrix. The following matrices are supported:



Chip	Description
	<ul style="list-style-type: none">• VO_CSC_MATRIX_IDENTITY• VO_CSC_MATRIX_BT601_TO_BT709• VO_CSC_MATRIX_BT709_TO_BT601
Hi3518E V200/Hi3519 V100	<p>The value range of VoLayer is [0, VO_MAX_LAYER_NUM]. The value range of u32Luma, u32Contrast, u32Hue, or u32Saturation is [0, 100], and the default value is 50.</p> <p>The default value of enCscMatrix is an identity matrix. The following matrices are supported:</p> <ul style="list-style-type: none">• VO_CSC_MATRIX_IDENTITY• VO_CSC_MATRIX_BT601_TO_BT709• VO_CSC_MATRIX_BT709_TO_BT601• VO_CSC_MATRIX_BT601_TO_RGB_PC• VO_CSC_MATRIX_BT709_TO_RGB_PC

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]

- This MPI is used to adjust the effect of output pictures, including the luminance, contrast, hue, and saturation. The effect value ranges from 0 to 100.
- When a video layer is disabled, the luminance, contrast, hue, saturation, and CSC matrix are restored to default values.
- The format of the output data of the LCD interface is RGB. However, the default format of the actual output data is YUV. Therefore, when the output interface is LCD, this MPI must be called to convert the output data (in YUV format) into RGB data by using the **VO_CSC_MATRIX_BT601_TO_RGB_PC** or **VO_CSC_MATRIX_BT709_TO_RGB_PC** conversion matrix.

[Example]

None

[See Also]

[HI_MPI_VO_GetVideoLayerCSC](#)

HI_MPI_VO_GetVideoLayerCSC

[Description]

Obtains the effect of the output pictures from a device.

[Syntax]

```
HI_S32 HI_MPI_VO_GetVideoLayerCSC(VO_LAYER VoLayer, VO_CSC_S
*pstVideoCSC);
```



[Parameter]

Parameter	Description	Input/Output
VoLayer	VO layer ID	Input
pstDevCSC	Pointer to the data structure of the picture output effect	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Value Range of VoLayer
Hi3516A/Hi3518E V200/Hi3519 V100	[0, VO_MAX_LAYER_NUM)

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]

This MPI is used to obtain the effect of output pictures, including the luminance, contrast, hue, and saturation. The effect value ranges from 0 to 100.

[Example]

None

[See Also]

[HI_MPI_VO_SetVideoLayerCSC](#)

HI_MPI_VO_SetAttrBegin

[Description]

Starts attribute setting.

[Syntax]

```
HI_S32 HI_MPI_VO_SetAttrBegin(VO_LAYER VoLayer);
```

[Parameter]



Parameter	Description	Input/Output
VoLayer	VO layer ID	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Value Range of VoLayer
Hi3516A/Hi3518E V200/Hi3519 V100	[0, VO_MAX_LAYER_NUM)

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]

- The BEGIN and END MPIs must be used in pairs. Otherwise, the channel attributes are invalid after the BEGIN MPI is called.
- This MPI can be used to process dynamic operations on the channel in batches. The dynamic operations include setting the channel attribute, showing, hiding, enabling, or disabling a channel, binding a cascade region to a channel, and unbinding a cascade region from a channel. When the picture display mode is switched or a VO channel is shown or hidden, a buffer needs to be obtained again and the picture display mode needs to be switched in a timely manner. Therefore, you are advised to call the BEGIN MPI, MPIs supporting the operations in batches (such as the MPIs for setting channel attributes, showing a channel, and hiding a channel), and the END MPI in sequence.
- The batch channel operations take effect only after [HI_MPI_VO_SetAttrEnd](#) is called. You need to call appropriate MPIs between [HI_MPI_VO_SetAttrBegin](#) and [HI_MPI_VO_SetAttrEnd](#). The following describes how to disable channels and video layers in batches. [HI_MPI_VO_SetAttrBegin](#), [HI_MPI_VO_DisableChn](#), [HI_MPI_VO_SetAttrEnd](#), and [HI_MPI_VO_DisableVideoLayer](#) must be called in sequence, because the operation of disabling the video layer must take effect after the channel is successfully disabled and the operation of disabling the channel must take effect after [HI_MPI_VO_SetAttrEnd](#) is called. If [HI_MPI_VO_SetAttrBegin](#), [HI_MPI_VO_DisableChn](#), [HI_MPI_VO_DisableVideoLayer](#), and [HI_MPI_VO_SetAttrEnd](#) are called in sequence, disabling the video layer fails.

[Example]

```
HI_S32 s32ret;
```



```
HI_U32 i;
VO_DEV VoDev = 0;
VO_LAYER VoLayer = 0;
VO_PUB_ATTR_S VoAttr;
VO_CHN_ATTR_S VoChnAttr[4];

memset(&VoAttr, 0, sizeof(VO_PUB_ATTR_S));
VoAttr.enIntfType = VO_INTF_CVBS; /*CVBS */
VoAttr.enIntfSync = VO_OUTPUT_PAL ; /*PAL Mode*/
VoAttr.u32BgColor = 0x0;

/* set public attribute of vo device */
s32ret = HI_MPI_VO_SetPubAttr(VoDev, &VoAttr);
if (HI_SUCCESS != s32ret)
{
    printf("vo set pub attr failed! \n");
    return s32ret;
}

/* enable vo device */
s32ret = HI_MPI_VO_Enable(VoDev);
if (HI_SUCCESS != s32ret)
{
    printf("enable vo device failed! \n");
    return s32ret;
}
/* configure and enable vo Layer 0 */  

.....  

/* configure and enable vo channel */  

for (i = 0; i < 4; i++)
{
    VoChnAttr[i].bDeflicker = HI_FALSE;
    VoChnAttr[i].u32Priority = 1;
    VoChnAttr[i].stRect.s32X = 360*(i%2);
    VoChnAttr[i].stRect.s32Y = 288*(i/2);
    VoChnAttr[i].stRect.u32Width = 360;
    VoChnAttr[i].stRect.u32Height = 288;

    /* set attribute of vo chn*/
    s32ret = HI_MPI_VO_SetChnAttr(VoLayer, i, &VoChnAttr[i]);
    if (HI_SUCCESS != s32ret)
    {
        printf("vo set dev %d chn %d attr failed! \n", VoLayer, i);
        return s32ret;
    }
}
```



```
}

/* enable vo chn */
s32ret = HI_MPI_VO_EnableChn(VoLayer, i);
if (HI_SUCCESS != s32ret)
{
    printf("enable vo Layer %d chn %d failed! \n", VoLayer,i);
    return s32ret;
}

/* do something else */ 
sleep(20);

/* set channel attributes begin */ 
s32ret = HI_MPI_VO_SetAttrBegin(VoLayer);
if (HI_SUCCESS != s32ret)
{
    printf("set attributes begin failed!\n");
    return s32ret;
}

/* reset channel attributes
 * we just show channel 0 and 1 on screen, and scale them as vertical
 * half D1
 */
for (i = 0; i < 2; i++)
{
    VoChnAttr[i].bDeflicker = HI_FALSE;
    VoChnAttr[i].u32Priority = 1;
    VoChnAttr[i].stRect.s32X = 352*(i%2);
    VoChnAttr[i].stRect.s32Y = 288*(i/2);
    VoChnAttr[i].stRect.u32Width = 360;
    VoChnAttr[i].stRect.u32Height = 576;

    /* set attribute of vo chn*/
    s32ret = HI_MPI_VO_SetChnAttr(VoLayer, i, &VoChnAttr[i]);
    if (HI_SUCCESS != s32ret)
    {
        printf("vo set chn %d attr failed! \n", i);
        return s32ret;
    }
}
```



```
/* hide channel 2 and 3
 * we do this just for saving bandwidth
 */
for (i = 2; i < 4; i++)
{
    s32ret = HI_MPI_VO_HideChn(VoLayer, i);
    if (s32ret != HI_SUCCESS)
    {
        printf("vo hide channel %d failed!\n", i);
        return HI_FAILURE;
    }
}

/* set channel attributes end
s32ret = HI_MPI_VO_SetAttrEnd(VoLayer);
if (HI_SUCCESS != s32ret)
{
    printf("set attributes end failed!\n");
    return s32ret;
}

/* do something else
sleep(20);

/* set channel attributes begin
s32ret = HI_MPI_VO_SetAttrBegin(VoLayer);
if (HI_SUCCESS != s32ret)
{
    printf("set attributes begin failed!\n");
    return s32ret;
}
/* reset channel attributes
 * now we set them back as 4 cif on screen
 */
for (i = 0; i < 4; i++)
{
    VoChnAttr[i].bDeflicker = HI_FALSE;
    VoChnAttr[i].u32Priority = 1;
    VoChnAttr[i].stRect.s32X = 360*(i%2);
    VoChnAttr[i].stRect.s32Y = 288*(i/2);
    VoChnAttr[i].stRect.u32Width = 360;
    VoChnAttr[i].stRect.u32Height = 288;
    /* set attribute of vo chn*/
    s32ret = HI_MPI_VO_SetChnAttr(VoLayer, i, &VoChnAttr[i]);
}
```



```
if (HI_SUCCESS != s32ret)
{
    printf("vo set chn %d attr failed! \n", i);
    return s32ret;
}
/*
 * show channel 2 and 3
for (i = 2; i < 4; i++)
{
    s32ret = HI_MPI_VO_ShowChn(VoLayer, i);
    if (s32ret != HI_SUCCESS)
    {
        printf("vo show channel %d failed!\n", i);
        return HI_FAILURE;
    }
}
/*
 * set channel attributes end
s32ret = HI_MPI_VO_SetAttrEnd(VoLayer);
if (HI_SUCCESS != s32ret)
{
    printf("set attributes end failed!\n");
    return s32ret;
}

/*
 * do something else
sleep(20);
/*
 * disable all channels
for (i = 0; i < 4; i++)
{
    s32ret = HI_MPI_VO_DisableChn(VoLayer, i);
    if (HI_SUCCESS != s32ret)
    {
        printf("vo disable chn %d failed!\n", i);
        return s32ret;
    }
}
/*
 * disable VoLayer
...
/*
 * disable vrou
(void)HI_MPI_VO_Disable(VoDev);
```

[See Also]

[HI_MPI_VO_SetAttrEnd](#)



HI_MPI_VO_SetAttrEnd

[Description]

Ends attribute setting.

[Syntax]

```
HI_S32 HI_MPI_VO_SetAttrEnd (VO_LAYER VoLayer);
```

[Parameter]

Parameter	Description	Input/Output
VoLayer	VO layer ID	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Value Range of VoLayer
Hi3516A/Hi3518E V200/Hi3519 V100	[0, VO_MAX_LAYER_NUM)

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]

- Before calling **HI_MPI_VO_SetAttrEnd**, ensure that **HI_MPI_VO_SetAttrBegin** is called.
- The BEGIN and END MPIs must be used in pairs. Otherwise, the channel attributes are invalid after the BEGIN MPI is called.

[Example]

See the example of [HI_MPI_VO_SetAttrBegin](#).

[See Also]

[HI_MPI_VO_SetAttrBegin](#)



HI_MPI_VO_SetPlayToleration

[Description]

Sets the play tolerance.

[Syntax]

```
HI_S32 HI_MPI_VO_SetPlayTolerance(VO_LAYER VoLayer, HI_U32  
u32Toleration);
```

[Parameter]

Parameter	Description	Input/Output
VoLayer	VO device ID	Input
u32Toleration	Play tolerance Value range: [1, 100000]	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Value Range of VoLayer
Hi3516A/Hi3518E V200/Hi3519 V100	The Hi3516A/Hi3518E V200/Hi3519 V100 does not support this MPI.

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]

- Before calling this MPI, ensure that a video layer is enabled.
- The unit of the play tolerance is millisecond, and the default value is 10000.
- If the absolute value of the difference between the PTSs of two decoded frames is greater than the play tolerance, the system will reset the PTSs and control the frame rate for decoded data based on the PTS of the current frame. The play tolerance is valid for play control during picture decoding rather than picture previewing.

[Example]



```
HI_S32 s32Ret;
VO_DEV VoDev = 1;
VO_LAYER VoLayer =1;
VO_PUB_ATTR_S stPubAttr;
VO_VIDEO_LAYER_ATTR_S stLayerAttr;
HI_U32 u32Toleration = 5000;

s32Ret = HI_MPI_VO_GetPubAttr(VoDev, &stPubAttr);
if (s32Ret != HI_SUCCESS)
{
    printf("Get device attributes failed with error code %#x!\n", s32Ret);
    return HI_FAILURE;
}
stPubAttr.u32BgColor = 0xff;
stPubAttr.enIntfType = VO_INTF_VGA;
stPubAttr.enIntfSync = VO_OUTPUT_1280x1024_60;
s32Ret = HI_MPI_VO_SetPubAttr(VoDev, &stPubAttr);
if (s32Ret != HI_SUCCESS)
{
    printf("Set device attributes failed with errno %#x!\n", s32Ret);
    return HI_FAILURE;
}
s32Ret = HI_MPI_VO_Enable(VoDev);
if (s32Ret != HI_SUCCESS)
{
    printf("Enable vo dev %d failed with errno %#x!\n", VoDev, s32Ret);
    return HI_FAILURE;
}
/* configure and enable vo Layer 1 */  
.....
s32Ret = HI_MPI_VO_SetPlayToleration (VoLayer, u32Toleration);
if (s32Ret != HI_SUCCESS)
{
    printf("Set play toleration failed with error code %#x!\n", s32Ret);
    return HI_FAILURE;
}
```

[See Also]

[HI_MPI_VO_GetPlayToleration](#)

HI_MPI_VO_GetPlayToleration

[Description]



Obtains the play tolerance.

[Syntax]

```
HI_S32 HI_MPI_VO_GetPlayToleration(VO_LAYER VoLayer, HI_U32  
*pu32Toleration);
```

[Parameter]

Parameter	Description	Input/Output
VoLayer	VO layer ID	Input
pu32Toleration	u32 pointer	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Value Range of VoLayer
Hi3516A/Hi3518E V200/Hi3519 V100	The Hi3516A/Hi3518E V200/Hi3519 V100 does not support this MPI.

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]

Before calling this MPI, ensure that a video layer is enabled.

[Example]

```
HI_S32 s32Ret;  
VO_DEV VoDev = 0;  
VO_LAYER VoLayer = 0;  
HI_U32 u32Toleration;  
  
s32Ret = HI_MPI_VO_GetPlayToleration (VoLayer, & u32Toleration);  
if (s32Ret != HI_SUCCESS)  
{  
    printf("Get play toleration failed with error code %#x!\n", s32Ret);
```



```
        return HI_FAILURE;
    }
    printf("Get dev %d play toleration is %d.\n", VoLayer, u32Toleration);
```

[See Also]

[HI_MPI_VO_SetPlayToleration](#)

HI_MPI_VO_GetScreenFrame

[Description]

Obtains the pictures displayed on the screen.

[Syntax]

```
HI_S32 HI_MPI_VO_GetScreenFrame(VO_LAYER VoLayer,
VIDEO_FRAME_INFO_S *pstVFrame, HI_S32 s32MilliSec);
```

[Parameter]

Parameter	Description	Input/Output
VoLayer	VO layer ID	Input
pstVFrame	Pointer to the data structure of the obtained information about the pictures displayed on the screen	Output
s32MilliSec	Timeout period, in ms	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Value Range of VoLayer
Hi3516A/Hi3518E V200/Hi3519 V100	[0, VO_MAX_LAYER_NUM)

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]



- Before calling this MPI, ensure that a video layer is enabled.
- Screen pictures can be obtained in block mode by calling this MPI. If **s32MilliSec** is **-1**, the MPI is blocked until there are pictures displayed on the screen; if **s32MilliSec** is **0**, screen pictures are obtained in non-block mode; if **s32MilliSec** is greater than 0, the value indicates the waiting time, and the MPI waits until the time elapses.
- This MPI can be repeatedly called. After calling **HI_MPI_VO_GetScreenFrame**, you must call **HI_MPI_VO_ReleaseScreenFrame** in time. The buffer for storing the pictures displayed on the screen must be released immediately for the physical VO device.
- The pictures obtained by calling **HI_MPI_VO_GetScreenFrame** contain PTSs. The PTSs show the time of combining captured pictures. You can modify the PTSs for encoding.
- This MPI supports only SD devices.
- This MPI is used only for debugging.

[Example]

None

[See Also]

[HI_MPI_VO_ReleaseScreenFrame](#)

HI_MPI_VO_ReleaseScreenFrame

[Description]

Releases the buffer for storing the pictures displayed on the screen.

[Syntax]

```
HI_S32 HI_MPI_VO_ReleaseScreenFrame(VO_LAYER VoLayer,  
VIDEO_FRAME_INFO_S *pstVFrame);
```

[Parameter]

Parameter	Description	Input/Output
VoLayer	VO device ID	Input
pstVFrame	Pointer to the data structure of the information about the released pictures displayed on the screen	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]



Chip	Value Range of VoDev
Hi3516A/Hi3518E V200/Hi3519 V100	[0, VO_MAX_LAYER_NUM)

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]

- Before disabling the video layer, you must release the buffer for storing the obtained screen pictures.
- This MPI can be repeatedly called. After calling [HI_MPI_VO_GetScreenFrame](#), you must call [HI_MPI_VO_ReleaseScreenFrame](#).

[Example]

None

[See Also]

[HI_MPI_VO_GetScreenFrame](#)

HI_MPI_VO_SetDispBufLen

[Description]

Sets the display buffer length of a video layer.

[Syntax]

```
HI_S32 HI_MPI_VO_SetDispBufLen(VO_LAYER VoLayer, HI_U32 u32BufLen);
```

[Parameter]

Parameter	Description	Input/Output
VoLayer	VO device ID	Input
u32BufLen	Display buffer length Value range: [0] and [3, 15]	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."



[Difference]

Chip	Value Range of VoLayer
Hi3516A/Hi3518E V200/Hi3519 V100	[0, VO_MAX_LAYER_NUM)

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]

- Before calling this MPI, ensure that the VO layer is disabled.
- The default buffer length is 0. The default display mode is VO bypass mode.
- When the conditions for VO bypass are not met, this MPI needs to be called to set the buffer length. Otherwise, the VO cannot work properly.

[Example]

```
HI_S32 s32Ret;
VO_DEV VoDev = 1;
VO_LAYER VoLayer = 1;
VO_PUB_ATTR_S stPubAttr;
HI_U32 u32DispBufLen = 10;

s32Ret = HI_MPI_VO_SetDispBufLen(VoLayer, u32DispBufLen);
if (s32Ret != HI_SUCCESS)
{
    printf("Set display buf len failed with error code %#x!\n", s32Ret);
    return HI_FAILURE;
}

s32Ret = HI_MPI_VO_GetPubAttr(VoDev, &stPubAttr);
if (s32Ret != HI_SUCCESS)
{
    printf("Get device attributes failed with error code %#x!\n", s32Ret);
    return HI_FAILURE;
}
stPubAttr.u32BgColor = 0xff;
stPubAttr.enIntfType = VO_INTF_VGA;
stPubAttr.enIntfSync = VO_OUTPUT_1280 x 1024_60;
s32Ret = HI_MPI_VO_SetPubAttr(VoDev, &stPubAttr);
if (s32Ret != HI_SUCCESS)
{
    printf("Set device attributes failed with errno %#x!\n", s32Ret);
    return HI_FAILURE;
}
```



```
    }
    s32Ret = HI_MPI_VO_Enable(VoDev);
    if (s32Ret != HI_SUCCESS)
    {
        printf("Enable vo dev %d failed with errno %#x!\n", VoDev, s32Ret);
        return HI_FAILURE;
    }
```

[See Also]

[HI_MPI_VO_GetDispBufLen](#)

HI_MPI_VO_GetDispBufLen

[Description]

Obtains the display buffer length of a video layer.

[Syntax]

```
HI_S32 HI_MPI_VO_GetDispBufLen(VO_LAYER VoLayer, HI_U32 *pu32BufLen);
```

[Parameter]

Parameter	Description	Input/Output
VoLayer	VO device ID	Input
pu32BufLen	u32 pointer	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Value Range of VoLayer
Hi3516A/Hi3518E V200/Hi3519 V100	[0, VO_MAX_LAYER_NUM)

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a



[Note]

None

[Example]

```
HI_S32 s32Ret;
VO_DEV VoDev = 1;
VO_LAYER VoLayer = 1;
VO_PUB_ATTR_S stPubAttr;
HI_U32 u32DispBufLen;

s32Ret = HI_MPI_VO_GetDispBufLen(VoLayer, &u32DispBufLen);
if (s32Ret != HI_SUCCESS)
{
    printf("Get display buf len failed with error code %#x!\n", s32Ret);
    return HI_FAILURE;
}
printf("Get VoLayer %d disp buf len is %d.\n", VoLayer, u32DispBufLen);
```

[See Also]

[HI_MPI_VO_SetDispBufLen](#)

HI_MPI_VO_EnableChn

[Description]

Enables a specified VO channel.

[Syntax]

```
HI_S32 HI_MPI_VO_EnableChn (VO_LAYER VoLayer, VO_CHN VoChn);
```

[Parameter]

Parameter	Description	Input/Output
VoLayer	VO layer ID	Input
VoChn	VO channel ID Value range: [0, VO_MAX_CHN_NUM)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."



[Difference]

Chip	Value Range of VoLayer
Hi3516A/Hi3518E V200/Hi3519 V100	[0, VO_MAX_LAYER_NUM)

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]

- Before calling this MPI, ensure that the video layer is bound to a device. Otherwise, an error code indicating failure is returned.
- Before calling this MPI, you must enable the video layer of the corresponding device.
- Before enabling a channel, you must configure it. Otherwise, an error is returned, indicating that the channel is not configured.
- A VO channel can be repeatedly enabled, and no error code indicating failure is returned.

[Example]

```
VO_DEV VoDev = 0;
VO_LAYER VoLayer = 0;
VO_CHN VoChn = 0;
VO_CHN_ATTR_S stChnAttr;

s32Ret = HI_MPI_VO_GetChnAttr(VoLayer, VoChn, &stChnAttr);
if (s32Ret != HI_SUCCESS)
{
    printf("Get channel attr failed with errno %#x!\n", s32Ret);
    return HI_FAILURE;
}

stChnAttr.u32Priority = 0;
stChnAttr.stRect.s32X = 0;
stChnAttr.stRect.s32Y = 0;
stChnAttr.stRect.u32Width = 720;
stChnAttr.stRect.u32Height = 576;

s32Ret = HI_MPI_VO_SetChnAttr(VoLayer, VoChn, &stChnAttr);
if (s32Ret != HI_SUCCESS)
{
    printf("Set channel attr failed with errno %#x!\n", s32Ret);
    return HI_FAILURE;
}
```



```
s32Ret = HI_MPI_VO_EnableChn(VoLayer, VoChn);  
if (s32Ret != HI_SUCCESS)  
{  
    printf("Enable channel failed with errno %#x!\n", s32Ret);  
    return HI_FAILURE;  
}  
  
s32Ret = HI_MPI_VO_DisableChn(VoLayer, VoChn);  
if (s32Ret != HI_SUCCESS)  
{  
    printf("Disable channel failed with errno %#x!\n", s32Ret);  
    return HI_FAILURE;  
}
```

[See Also]

[HI_MPI_VO_DisableChn](#)

HI_MPI_VO_DisableChn

[Description]

Disables a specified VO channel.

[Syntax]

```
HI_S32 HI_MPI_VO_DisableChn(VO_LAYER VoLayer, VO_CHN VoChn);
```

[Parameter]

Parameter	Description	Input/Output
VoLayer	VO layer ID	Input
VoChn	VO channel ID Value range: [0, VO_MAX_CHN_NUM)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]



Chip	Value Range of VoLayer
Hi3516A/Hi3518E V200/Hi3519 V100	[0, VO_MAX_LAYER_NUM)

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]

- A VO channel can be repeatedly disabled, and no error code indicating failure is returned.
- If a VO channel of the HD device is bound to a VPSS channel, you are advised to disable the VO channel by calling this MPI and then unbind the VO channel from the VPSS channel. Otherwise, an error indicating [HI_ERR_VO_BUSY](#) is returned.

[Example]

See the example of [HI_MPI_VO_EnableChn](#).

[See Also]

[HI_MPI_VO_EnableChn](#)

HI_MPI_VO_SetChnAttr

[Description]

Sets the attributes of a specified VO channel.

[Syntax]

```
HI_S32 HI_MPI_VO_SetChnAttr(VO_LAYER VoLayer, VO_CHN VoChn, const  
VO_CHN_ATTR_S *pstChnAttr);
```

[Parameter]

Parameter	Description	Input/Output
VoLayer	VO layer ID	Input
VoChn	VO channel ID Value range: [0, VO_MAX_CHN_NUM)	Input
pstAttr	Pointer to the attributes of a VO channel	Input

[Return Value]

Return Value	Description
0	Success



Return Value	Description
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Description
Hi3516A/Hi3518E V200/Hi3519 V100	<ul style="list-style-type: none">Value range of VoLayer: [0, VO_MAX_LAYER_NUM)The range of the SD device priority is [0, VO_MAX_CHN_NUM).The width or height of an SD device channel must be greater than or equal to 32 pixels.

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]

- A larger value indicates a higher SD device priority.
The priority range is [0, VO_MAX_CHN_NUM – 1]. When the display regions of multiple channels overlap, the picture of the channel with a lower priority is overlaid with the picture of the channel with a higher priority. When the display regions of multiple channels with the same priority are overlaid, the pictures from the channel with a smaller ID are overlaid with the pictures from the channel with a larger ID by default.
- The size of the channel display region cannot exceed the canvas size **stImageSize** (one of video layer attributes).
- HI_MPI_VO_SetChnAttr** is a dynamic MPI. That is, you can call it when a VO device is enabled and the corresponding video layer is configured.
- The channel attribute **stRect** of the HD device must be 2-pixel-aligned, and the width and height must be greater than or equal to 32 pixels. If the interlaced timing is used and the pixel format is SEMIPLANAR_420, the height in **stRect** must be 4-pixel-aligned. It is recommended that the vertical coordinate of the start point be 4-pixel-aligned. Otherwise, unexpected picture quality issues may occur. The width or height of an HD channel must be greater than or equal to 32 pixels.
- If the video layer of an HD device is zoomed in, the start position, width, and height defined in the **stRect** attribute are the values before the video layer is zoomed in. After the video layer is zoomed in, the display start position shifts, and the width and height increase based on the zoom-in ratio of the video layer.
- For the channel attribute **stRect** of the SD device, the channel width and height must be 2-pixel-aligned.

[Example]

See the example of [HI_MPI_VO_EnableChn](#).

[See Also]



HI_MPI_VO_GetChnAttr

[Description]

Obtains the attributes of a specified VO channel.

[Syntax]

```
HI_S32 HI_MPI_VO_GetChnAttr(VO_LAYER VoLayer, VO_CHN VoChn, VO_CHN_ATTR_S *pstChnAttr);
```

[Parameter]

Parameter	Description	Input/Output
VoLayer	VO layer ID	Input
VoChn	VO channel ID Value range: [0, VO_MAX_CHN_NUM)	Input
pstChnAttr	Pointer to the attributes of a VO channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Value Range of VoLayer
Hi3516A/Hi3518E V200/Hi3519 V100	[0, VO_MAX_LAYER_NUM)

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]

None

[Example]

See the example of [HI_MPI_VO_EnableChn](#).



[See Also]

[HI_MPI_VO_SetChnAttr](#)

HI_MPI_VO_SetChnField

[Description]

Sets the frame/field display policy of a specified VO channel.

There are three application scenarios.

- When CIF pictures are previewed, you can configure the VI device to output the two-field CIF pictures and configure the corresponding VO channel to display two-field pictures. The display of 2-field pictures is smoother than the display of 1-field pictures.
- When 2-field pictures are previewed at a low frame rate (for example, D1 pictures are previewed at a low frame rate), you need to configure the corresponding VO channel to display 1-field pictures. Otherwise, the screen flickers upwards and downwards.
- If VI channels capture D1 and CIF pictures at the same time, you need to call this MPI to capture fields of D1 pictures. Otherwise, the screen flickers upwards and downwards.

[Syntax]

```
HI_S32 HI_MPI_VO_SetChnField(VO_LAYER VoLayer, VO_CHN VoChn, const  
VO_DISPLAY_FIELD_E enField);
```

[Parameter]

Parameter	Description	Input/Output
VoLayer	VO device ID Value range: [0, VO_MAX_LAYER_NUM)	Input
VoChn	VO channel ID Value range: [0, VO_MAX_CHN_NUM)	Input
enField	Frame/Field display policy of a video channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Description
Hi3516A/Hi3518E V200/Hi3519 V100	The Hi3516A/Hi3518E V200/Hi3519 V100 does not support this MPI.



[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]

- If the MPI is not called explicitly, the VO channel displays two fields (that is, **VO_FIELD_BOTH**) by default.
- If the picture that is sent to a VO channel is a frame picture (for example, a single-field CIF picture), calling this MPI has no effect on picture display.
- This MPI is valid only for SD devices.

[Example]

```
/*The following is an example of previewing pictures at a low frame rate.  
The VI frame rate is set to 8 fps, and the display mode of corresponding  
VO channel is set to bottom field.*/
VO_DISPLAY_FIELD_E enField;
VI_CHN_ATTR_S stViChnAttr;
...
stViChnAttr.s32SrcFrameRate = 25;
stViChnAttr.s32FrameRate = 8;
...
if (HI_SUCCESS!= HI_MPI_VI_SetChnAttr(0, & stViChnAttr))
{
    printf("HI_MPI_VI_SetChnAttr failed !\n");
    return HI_FAILURE;
}
if (HI_SUCCESS!=HI_MPI_VO_SetChnField(2, 0, VO_FIELD_BOTTOM) )
{
    printf("HI_MPI_VO_SetChnField failed !\n");
    return HI_FAILURE;
}
if (HI_SUCCESS!=HI_MPI_VO_GetChnField(2, 0, &enField) )
{
    printf("HI_MPI_VO_GetChnField failed !\n");
    return HI_FAILURE;
}
```

[See Also]

[HI_MPI_VO_GetChnField](#)

HI_MPI_VO_GetChnField

[Description]

Obtains the frame/field display policy of a specified VO channel.



[Syntax]

```
HI_S32 HI_MPI_VO_GetChnField(VO_LAYER VoLayer, VO_CHN VoChn,  
VO_DISPLAY_FIELD_E *pField);
```

[Parameter]

Parameter	Description	Input/Output
VoLayer	VO device ID Value range: [0, VO_MAX_LAYER_NUM)	Input
VoChn	VO channel ID Value range: [0, VO_MAX_CHN_NUM)	Input
pField	Pointer to the attributes of the channel display position	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Description
Hi3516A//Hi3518E V200/Hi3519 V100	The Hi3516A/Hi3518E V200/Hi3519 V100 does not support this MPI.

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]

If the display policy is not set, calling this MPI returns the default policy **VO_FIELD_BOTH**.

[Example]

See the example of [HI_MPI_VO_SetChnField](#).

[See Also]

[HI_MPI_VO_SetChnField](#)

HI_MPI_VO_GetChnFrame

[Description]



Obtains frames from a VO channel.

[Syntax]

```
HI_S32 HI_MPI_VO_GetChnFrame(VO_LAYER VoLayer, VO_CHN VoChn,  
VIDEO_FRAME_INFO_S *pstFrame, HI_S32 s32MilliSec);
```

[Parameter]

Parameter	Description	Input/Output
VoLayer	VO layer ID	Input
VoChn	VO channel ID Value range: [0, VO_MAX_CHN_NUM)	Input
pstVFrame	Pointer to the information about video data	Output
s32MilliSec	Timeout period (in ms)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Value Range of VoLayer
Hi3516A//Hi3518E V200/Hi3519 V100	[0, VO_MAX_LAYER_NUM)

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]

- Before calling the MPI, ensure that a VO channel is enabled.
- Channel frames can be obtained in block mode by calling this MPI. If **s32MilliSec** is **-1**, the MPI is blocked until there are pictures in the channel; if **s32MilliSec** is **0**, frames are obtained in non-block mode; if **s32MilliSec** is greater than 0, the value indicates the waiting time, and the MPI waits until the time elapses.
- The frames need to be released in a timely manner after they are obtained. The channel frames can be obtained again only after [HI_MPI_VO_ReleaseChnFrame](#) is called to release the channel frames.



[Example]

None

[See Also]

None

HI_MPI_VO_ReleaseChnFrame

[Description]

Releases the buffer for storing the pictures of a VO channel.

[Syntax]

```
HI_S32 HI_MPI_VO_ReleaseChnFrame(VO_LAYER VoLayer, VO_CHN VoChn, const  
VIDEO_FRAME_INFO_S *pstFrame);
```

[Parameter]

Parameter	Description	Input/Output
VoLayer	VO layer ID	Input
VoChn	VO channel ID Value range: [0, VO_MAX_CHN_NUM)	Input
pstFrame	Pointer to the data structure of the information about the released pictures of a VO channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Value Range of VoLayer
Hi3516A/Hi3518E V200/Hi3519 V100	[0, VO_MAX_LAYER_NUM)

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]



- HI_MPI_VO_GetChnFrame and HI_MPI_VO_ReleaseChnFrame must be called in pairs.
- This MPI cannot be called repeatedly.

[Example]

None

[See Also]

[HI_MPI_VO_GetChnFrame](#)

HI_MPI_VO_SendFrame

[Description]

Sends pictures to a specified VO channel.

[Syntax]

```
HI_S32 HI_MPI_VO_SendFrame(VO_LAYER VoLayer, VO_CHN VoChn,  
VIDEO_FRAME_INFO_S *pstVFrame, HI_S32 s32MilliSec);
```

[Parameter]

Parameter	Description	Input/Output
VoLayer	VO layer ID	Input
VoChn	VO channel ID Value range: [0, VO_MAX_CHN_NUM)	Input
pstVFrame	Pointer to the information about video data	Input
s32MilliSec	Timeout period, in ms	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Value Range of VoLayer
Hi3516A/Hi3518E V200/Hi3519 V100	[0, VO_MAX_LAYER_NUM)

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h



- Library file: libmpi.a

[Note]

- Before calling the MPI, ensure that a VO channel is enabled.
- Pictures can be sent in block mode by calling **HI_MPI_VO_SendFrame**. If **s32MilliSec** is **-1**, the MPI is blocked until there are pictures to be sent; if **s32MilliSec** is **0**, pictures are sent in non-block mode; if **s32MilliSec** is greater than **0**, the MPI is blocked and waited until the time elapses.
- The input video data information must meet the requirements on the VO data. To be specific, the width and height in the input information must be consistent with the actual picture width and height, be greater than or equal to 32, and be 2-pixel-aligned. The pixel format can be SPYCbCr420, SPYCbCr422, or single component. The compression mode can be non-compression mode or 256-segment compression mode. Only the linear video format is supported.

[Example]

None

[See Also]

None

HI_MPI_VO_SetChnFrameRate

[Description]

Sets the display frame rate of a specified VO channel.

[Syntax]

```
HI_S32 HI_MPI_VO_SetChnFrameRate(VO_LAYER VoLayer, VO_CHN VoChn, HI_S32 s32ChnFrmRate);
```

[Parameter]

Parameter	Description	Input/Output
VoLayer	VO layer ID	Input
VoChn	VO channel ID Value range: [0, VO_MAX_CHN_NUM)	Input
s32VoFramerate	Display frame rate of a video channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."



[Difference]

Chip	Value Range of VoLayer
Hi3516A/Hi3518E V200/Hi3519 V100	The Hi3516A/Hi3518E V200/Hi3519 V100 does not support this MPI.

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]

- This MPI can be called only after the public attributes of a VO device are set. Otherwise, an error code indicating failure is returned.
- The frame rate can be set to a multiple of the negative Nx for rewind. N is an integer ranging from -64 to $+64$, and x is the frame rate in PAL or NTSC standard. In this case, you need to send the pictures in inverse sequence (pictures with descending PTSs) to the VO channel.
- Calling [HI_MPI_VO_EnableChn](#) enables the channel to play streams at the normal speed. Therefore, you are advised to call [HI_MPI_VO_SetChnFrameRate](#) to control playback after the VO channel is enabled.
- If you call [HI_MPI_VO_SetChnAttr](#) when the VO channel is disabled, the channel frame rate is reset to the display frame rate. The channel frame rate is the frame rate of the input picture, and the display frame rate is the final display frame rate of the channel. The display frame rate can be set when video layer attributes are set.

[Example]

```
VO_DEV VoDev = 0;
VO_LAYER VoLayer = 0;
VO_CHN VoChn = 0;
HI_S32 s32ChnFrmRate = 30;

s32Ret = HI_MPI_VO_SetChnFrameRate(VoLayer, VoChn, s32ChnFrmRate);
if (s32Ret != HI_SUCCESS)
{
    printf("Set channel %d frame rate failed with errno %#x!\n", VoChn,
s32Ret);
    return HI_FAILURE;
}
```

[See Also]

None

HI_MPI_VO_GetChnFrameRate

[Description]

Obtains the display frame rate of a specified VO channel.



[Syntax]

```
HI_S32 HI_MPI_VO_GetChnFrameRate(VO_LAYER VoLayer, VO_CHN VoChn, HI_S32  
*ps32ChnFrmRate);
```

[Parameter]

Parameter	Description	Input/Output
VoLayer	VO layer ID	Input
VoChn	VO channel ID Value range: [0, VO_MAX_CHN_NUM)	Input
ps32VoFramerate	Display frame rate of a video channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Value Range of VoLayer
Hi3516A/Hi3518E V200/Hi3519 V100	The Hi3516A/Hi3518E V200/Hi3519 V100 does not support this MPI.

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]

The obtained frame rate is the configured value. If no frame rate is configured, the full frame rate is returned.

[Example]

```
VO_DEV VoDev = 0;  
VO_LAYER VoLayer = 0;  
VO_CHN VoChn = 0;  
HI_S32 s32ChnFrmRate;  
  
s32Ret = HI_MPI_VO_GetChnFrameRate(VoLayer, VoChn, &s32ChnFrmRate);  
if (s32Ret != HI_SUCCESS)
```



```
{  
    printf("Set channel %d frame rate failed with errno %#x!\n", VoChn,  
s32Ret);  
    return HI_FAILURE;  
}  
printf("Channel %d frame rate is %d.\n", VoChn, s32ChnFrmRate);
```

[See Also]

None

HI_MPI_VO_PauseChn

[Description]

Pauses a specified VO channel.

[Syntax]

```
HI_S32 HI_MPI_VO_PauseChn (VO_LAYER VoLayer, VO_CHN VoChn);
```

[Parameter]

Parameter	Description	Input/Output
VoLayer	VO layer ID	Input
VoChn	VO channel ID Value range: [0, VO_MAX_CHN_NUM)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Value Range of VoLayer
Hi3516A/Hi3518E V200/Hi3519 V100	The Hi3516A/Hi3518E V200/Hi3519 V100 does not support this MPI.

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a



[Note]

- This MPI is not recommended during VI-VO preview.
- A channel can be paused repeatedly and no error code indicating failure is returned.

[Example]

```
HI_S32 s32ret;
VO_DEV VoDev = 0;
VO_LAYER VoLayer = 0;
VO_CHN VoChn = 0;
...
/* enable vo chn */
s32ret = HI_MPI_VO_EnableChn(VoLayer, VoChn);
if (HI_SUCCESS != s32ret)
{
    printf("enable vo chn failed! \n");
    return s32ret;
}
/* pause current vo channel */
s32ret = HI_MPI_VO_PauseChn(VoLayer, VoChn);
if (HI_SUCCESS != s32ret)
{
    printf("pause vo chn failed! \n");
    return s32ret;
}
/* resume current vo channel */
s32ret = HI_MPI_VO_ResumeChn(VoLayer, VoChn);
if (HI_SUCCESS != s32ret)
{
    printf("resume vo chn failed! \n");
    return s32ret;
}
while (getchar() != 'q')
{
    /* step forward current vo channel */
    s32ret = HI_MPI_VO_StepChn(VoLayer, VoChn);
    if (HI_SUCCESS != s32ret)
    {
        printf("step play vo chn failed! \n");
        return s32ret;
    }
}

/* resume current vo channel */
s32ret = HI_MPI_VO_ResumeChn(VoLayer, VoChn);
```



```
if (HI_SUCCESS != s32ret)
{
    printf("resume vo chn failed! \n");
    return s32ret;
}
(void) HI_MPI_VO_DisableChn(VoLayer, VoChn);
```

[See Also]

[HI_MPI_VO_ResumeChn](#)

HI_MPI_VO_ResumeChn

[Description]

Resumes a specified VO channel.

[Syntax]

```
HI_S32 HI_MPI_VO_ResumeChn(VO_LAYER VoLayer, VO_CHN VoChn);
```

[Parameter]

Parameter	Description	Input/Output
VoLayer	VO layer ID	Input
VoChn	VO channel ID Value range: [0, VO_MAX_CHN_NUM)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Value Range of VoLayer
Hi3516A/Hi3518E V200/Hi3519 V100	The Hi3516A/Hi3518E V200/Hi3519 V100 does not support this MPI.

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a



[Note]

A channel can be resumed repeatedly and no error code indicating failure is returned.

[Example]

See the example of [HI_MPI_VO_PauseChn](#).

[See Also]

None

HI_MPI_VO_StepChn

[Description]

Plays the streams in a specified VO channel by frame.

[Syntax]

```
HI_S32 HI_MPI_VO_StepChn(VO_LAYER VoLayer, VO_CHN VoChn);
```

[Parameter]

Parameter	Description	Input/Output
VoLayer	VO layer ID	Input
VoChn	VO channel ID Value range: [0, VO_MAX_CHN_NUM)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Value Range of VoLayer
Hi3516A/Hi3518E V200/Hi3519 V100	The Hi3516A/Hi3518E V200/Hi3519 V100 does not support this MPI.

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]



- You can resume normal play by calling [HI_MPI_VO_ResumeChn](#).
- This MPI is not recommended during VI-VO preview.

[Example]

See the example of [HI_MPI_VO_PauseChn](#).

[See Also]

[HI_MPI_VO_ResumeChn](#)

HI_MPI_VO_RefreshChn

[Description]

Refreshes a specified VO channel.

[Syntax]

```
HI_S32 HI_MPI_VO_RefreshChn(VO_LAYER VoLayer, VO_CHN VoChn);
```

[Parameter]

Parameter	Description	Input/Output
VoLayer	VO layer ID	Input
VoChn	VO channel ID Value range: [0, VO_MAX_CHN_NUM)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Value Range of VoLayer
Hi3516A/Hi3518E V200/Hi3519 V100	The Hi3516A/Hi3518E V200/Hi3519 V100 does not support this MPI.

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]



- This MPI applies to the electronic magnification scenario when the HD device is paused. Electronic magnification for the HD device is implemented by calling HI_MPI_VPSS_SetCropCfg. When the HD device is paused, the pictures clipped by the VPSS cannot be sent to the VO channel. You only need to refresh the VO channel by calling HI_MPI_VO_ChnRefresh.
- This MPI applies to the electronic magnification scenario even the decoding frame rate is low.
- Because the settings by calling HI_MPI_VO_ChnRefresh take effect immediately, you must call HI_MPI_VO_RefreshChn after calling HI_MPI_VPSS_SetCropCfg. Otherwise, the pictures before cropping are obtained.
- This MPI does not take effect in normal play mode including forward and slow play.
- This MPI does not apply to the SD device and VI-VO previewing.

[Example]

```
HI_S32 s32ret;
VO_DEV VoDev = 0;
VO_LAYER VoLayer = 0;
VO_CHN VoChn = 0;
...
/* enable vo chn */
s32ret = HI_MPI_VO_EnableChn(VoLayer, VoChn);
if (HI_SUCCESS != s32ret)
{
    printf("enable vo chn failed! \n");
    return s32ret;
}

/* pause current vo channel */
s32ret = HI_MPI_VO_PauseChn(VoLayer, VoChn);
if (HI_SUCCESS != s32ret)
{
    printf("pause vo chn failed! \n");
    return s32ret;
}

/* set vpss clip config */
...
/* refresh current vo channel */
s32ret = HI_MPI_VO_RefreshChn(VoLayer, VoChn);
if (HI_SUCCESS != s32ret)
{
    printf("refresh vo chn failed! \n");
    return s32ret;
}
```



```
(void) HI_MPI_VO_DisableChn(VoLayer, VoChn);
```

[See Also]

[HI_MPI_VO_ResumeChn](#)

HI_MPI_VO_ShowChn

[Description]

Shows a specified channel.

[Syntax]

```
HI_S32 HI_MPI_VO_ShowChn(VO_LAYER VoLayer, VO_CHN VoChn);
```

[Parameter]

Parameter	Description	Input/Output
VoLayer	VO layer ID	Input
VoChn	VO channel ID Value range: [0, VO_MAX_CHN_NUM)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Value Range of VoLayer
Hi3516A/Hi3518E V200/Hi3519 V100	[0, VO_MAX_LAYER_NUM)

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpapi.a

[Note]

- Before calling the MPI, ensure that the video layer of the VO channel is enabled.
- The channel is shown by default.



[Example]

See the example of [HI_MPI_VO_SetAttrBegin](#).

[See Also]

[HI_MPI_VO_HideChn](#)

HI_MPI_VO_HideChn

[Description]

Hides a specified channel.

[Syntax]

```
HI_S32 HI_MPI_VO_HideChn(VO_LAYER VoLayer, VO_CHN VoChn);
```

[Parameter]

Parameter	Description	Input/Output
VoLayer	VO layer ID	Input
VoChn	VO channel ID Value range: [0, VO_MAX_CHN_NUM)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Value Range of VoLayer
Hi3516A/Hi3518E V200/Hi3519 V100	[0, VO_MAX_LAYER_NUM)

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]

When a VO channel of the HD channel is bound to a VPSS channel, you are advised to call this MPI to hide the VO channel, and then unbind the VO channel from the VPSS channel. Otherwise, the error code HI_ERR_VO_BUSY is returned.



[Example]

See the example of [HI_MPI_VO_SetAttrBegin](#).

[See Also]

[HI_MPI_VO_ShowChn](#)

HI_MPI_VO_SetZoomInWindow

[Description]

Sets a VO zoom-in window.

[Syntax]

```
HI_S32 HI_MPI_VO_SetZoomInWindow(VO_LAYER VoLayer, VO_CHN VoChn, const  
VO_ZOOM_ATTR_S *pstZoomAttr);
```

[Parameter]

Parameter	Description	Input/Output
VoLayer	VO layer ID	Input
VoChn	VO channel ID Value range: [0, VO_MAX_CHN_NUM)	Input
pstZoomAttr	Data structure of the partial zoom-in attribute	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Value Range of VoLayer
Hi3516A/Hi3518E V200/Hi3519 V100	[0, VO_MAX_LAYER_NUM)

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]

- Before calling the MPI, you must enable a VO device and set public VO attributes.



- Before calling the MPI, ensure that the video layer is configured.
- This MPI has different meanings for the HD device and SD device.

The HD device can scale the entire video layer but not the picture of a single channel. For example, if the size of a source VI picture is D1 and the size of the corresponding VO channel is CIF, the picture is scaled by the VPSS. For the HD device, this MPI is used to crop the picture of the VO channel and display the cropped part on the display region without scaling. The vacancy region is displayed in background color.

The SD device can scale the picture of a single channel by using the VGS. For the SD device, this MPI is used to crop the picture of the VO channel and scale the cropped part to fit into the channel size, implementing the partial zoom-in function.

- The source data of a VO channel can be cropped by rectangle or ratio. For details, see the description of [VO_ZOOM_ATTR_S](#).
- For SD devices, if video inputs are D1 and 2CIF (two fields) pictures, you need to set **pstZoomAttr** based on the small picture. The SDK automatically processes the large picture, ensuring that the fields of vision of large and small pictures are consistent.
- In video cascade mode, the master chip can select regions from the picture obtained from the cascaded slave chip and combine the regions by using the partial zoom-in function.

[Example]

```
/* define input parameter */
HI_S32 s32Ret;
VO_DEV VoDev = 0;
VO_LAYER VoLayer = 0;
VO_CHN VoChn = 0;
VO_ZOOM_ATTR_S stZoomWindow, stZoomWindowGet;

/*
 * we should enable VO and VO channel first!
 * TODO: enable operation.
 */

stZoomAttr.enZoomType = VOU_ZOOM_IN_RECT;
stZoomWindow.stZoomRect.s32X = 128;
stZoomWindow.stZoomRect.s32Y = 128;
stZoomWindow.stZoomRect.u32Width = 200;
stZoomWindow.stZoomRect.u32Height = 200;

/* set zoom window */
s32Ret = HI_MPI_VO_SetZoomInWindow(VoLayer, VoChn, &stZoomWindow);
if (s32Ret != HI_SUCCESS)
{
    printf("Set zoom attribute failed, ret = %#x.\n", s32Ret);
    return HI_FAILURE;
}
/* get current zoom window parameter */
s32Ret = HI_MPI_VO_GetZoomInWindow(VoLayer, VoChn, &stZoomWindowGet);
```



```
if (s32Ret != HI_SUCCESS)
{
    printf("Get zoom attribute failed, ret = %#x.\n", s32Ret);
    return HI_FAILURE;
}
else
{
    printf("Current zoom window is (%d,%d,%d,%d) !\n",
           stZoomWindowGet.stZoomRect.s32X,
           stZoomWindowGet.stZoomRect.s32Y,
           stZoomWindowGet.stZoomRect.u32Width,
           stZoomWindowGet.stZoomRect.u32Height);
}
/*
 * TODO: something else ...
 */
stZoomWindow.stZoomRect.s32X = 0;
stZoomWindow.stZoomRect.s32Y = 0;
stZoomWindow.stZoomRect.u32Width = 0;
stZoomWindow.stZoomRect.u32Height = 0;

/* cancel zoom window, we use (0,0,0,0) to recover */
s32Ret = HI_MPI_VO_SetZoomInWindow(VoLayer, VoChn, &stZoomWindow);
if (s32Ret != HI_SUCCESS)
{
    printf("Recover zoom attribute failed, ret = %#x.\n", s32Ret);
    return HI_FAILURE;
}
```

[See Also]

[HI_MPI_VO_GetZoomInWindow](#)

HI_MPI_VO_GetZoomInWindow

[Description]

Obtains the parameters of a VO zoom-in window.

[Syntax]

```
HI_S32 HI_MPI_VO_GetZoomInWindow(VO_LAYER VoLayer, VO_CHN VoChn,
                                  VO_ZOOM_ATTR_S *pstZoomAttr);
```

[Parameter]

Parameter	Description	Input/Output
VoLayer	VO layer ID	Input



Parameter	Description	Input/Output
VoChn	VO channel ID Value range: [0, VO_MAX_CHN_NUM)	Input
pstZoomAttr	Pointer to the data structure of partial zoom-in attributes	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Value Range of VoLayer
Hi3516A/Hi3518E V200/Hi3519 V100	[0, VO_MAX_LAYER_NUM)

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]

If the zoom-in window is not set, the setting (0, 0, 0, 0) is obtained by default.

[Example]

See the example of [HI_MPI_VO_SetZoomInWindow](#).

[See Also]

[HI_MPI_VO_SetZoomInWindow](#)

HI_MPI_VO_GetChnPts

[Description]

Obtains the PTS of the current picture in a specified VO channel.

[Syntax]

```
HI_S32 HI_MPI_VO_GetChnPts(VO_LAYER VoLayer, VO_CHN VoChn, HI_U64  
*pu64ChnPts);
```

[Parameter]



Parameter	Description	Input/Output
VoLayer	VO layer ID	Input
VoChn	VO channel ID Value range: [0, VO_MAX_CHN_NUM)	Input
pu64VoChnPts	Pointer to the channel PTS	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Value Range of VoLayer
Hi3516A/Hi3518E V200/Hi3519 V100	The Hi3516A/Hi3518E V200/Hi3519 V100 does not support this MPI.

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]

- Before calling this MPI, ensure that the video layer and VO channel are enabled.
- When the decoding module is bound to the VOU by the system for transmitting video frames and the PTS of the decoded frame is 0, the VOU marks frames with PTSs starting from 0 based on play interval. This ensures that every frame is played once. In other cases, the PTSs of original videos are not changed.

[Example]

```
VO_DEV VoDev = 0;
VO_LAYER VoLayer = 0;
VO_CHN VoChn = 0;
HI_U64 u64ChnPts;

s32Ret = HI_MPI_VO_GetChnPts(VoLayer, VoChn, &u64ChnPts);
if (s32Ret != HI_SUCCESS)
{
    printf("Get channel %d pts failed with errno %#x!\n", VoChn, s32Ret);
```



```
        return HI_FAILURE;
    }

printf("Channel %d pts is %llu.\n", VoChn, u64ChnPts);
```

[See Also]

None

HI_MPI_VO_QueryChnStat

[Description]

Queries the status of a VO channel.

[Syntax]

```
HI_S32 HI_MPI_VO_QueryChnStat(VO_LAYER VoLayer, VO_CHN VoChn,
                                VO_QUERY_STATUS_S *pstStatus);
```

[Parameter]

Parameter	Description	Input/Output
VoLayer	VO layer ID	Input
VoChn	VO channel ID Value range: [0, VO_MAX_CHN_NUM)	Input
pstStatus	Pointer to the data structure of the channel status	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Value Range of VoLayer
Hi3516A/Hi3518E V200/Hi3519 V100	[0, VO_MAX_LAYER_NUM)

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpci.a

[Note]



- Before calling this MPI, ensure that a VO device is enabled.
- Before calling the MPI, ensure that the VO channel whose status is to be queried is enabled.
- This MPI can be repeatedly called.

[Example]

```
HI_S32 s32Ret;
VO_DEV VoDev = 0;
VO_LAYER VoLayer = 0;
VO_CHN VoChn = 0;
VO_QUERY_STATUS_S stVoQueryStatus;

/* enable vo device and vo channel */
...
s32Ret = HI_MPI_VO_QueryChnStat(VoLayer, VoChn, &stVoQueryStatus);
if (HI_SUCCESS != s32Ret)
{
    printf("Query channel status failed with errorcode %#x!\n", s32Ret);
    return HI_FAILURE;
}
printf("Current vo dev %d channel %d has %d VB occupied!\n",
VoDev, VoChn, stVoQueryStatus .u32ChnBufUsed );
```

[See Also]

None

HI_MPI_VO_ClearChnBuffer

[Description]

Clears the data in a specified VO channel buffer.

[Syntax]

```
HI_S32 HI_MPI_VO_ClearChnBuffer(VO_LAYER VoLayer, VO_CHN VoChn, HI_BOOL
bClrAll);
```

[Parameter]

Parameter	Description	Input/Output
VoLayer	VO layer ID	Input
VoChn	VO channel ID Value range: [0, VO_MAX_CHN_NUM)	Input



Parameter	Description	Input/Output
bClrAll	<p>Whether to clear all the data in the channel buffer. The following describes the value definitions:</p> <ul style="list-style-type: none">• HI_TRUE: clears all the data in the channel buffer. In this case, no picture is displayed in the area corresponding to this channel until new pictures are received.• HI_FALSE: retains a picture in the channel buffer and clears other data.	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Value Range of VoLayer
Hi3516A/Hi3518E V200/Hi3519 V100	[0, VO_MAX_LAYER_NUM)

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]

You can call this MPI to clear a VO channel buffer.

[Example]

```
VO_DEV VoDev = 0;
VO_LAYER VoLayer = 0;
VO_CHN VoChn = 0;
HI_BOOL bClearAll = HI_TRUE;

s32Ret = HI_MPI_VO_ClearChnBuffer (VoLayer, VoChn, bClearAll);
if (s32Ret != HI_SUCCESS)
{
    printf("Clear channel %d buf failed with errno %#x!\n", VoChn, s32Ret);
    return HI_FAILURE;
}
```



[See Also]

None

HI_MPI_VO_SetChnBorder

[Description]

Sets channel border attributes.

[Syntax]

```
HI_S32 HI_MPI_VO_SetChnBorder(VO_LAYER VoLayer, VO_CHN VoChn, const  
VO_BORDER_S *pstBorder);
```

[Parameter]

Parameter	Description	Input/Output
VoLayer	VO layer ID	Input
VoChn	VO channel ID Value range: [0, VO_MAX_CHN_NUM)	Input
pstBorder	Pointer to video channel border attributes	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Value range of VoLayer
Hi3516A/Hi3518E V200/Hi3519 V100	[0, VO_MAX_LAYER_NUM)

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]

- This MPI applies only to the channels of the video layer that is processed in VO_PART_MODE_SINGLE mode. In VO_PART_MODE_SINGLE mode, channel borders are processed when the VOU calls the VGS to combine pictures. In VO_PART_MODE_MULTI mode, channel borders are processed by the VPSS. If this



MPI is called for the video layer that is processed in VO_PART_MODE_MULTI mode, a code indicating success is returned.

- In the VO border attributes, the width must be an even number that falls within [2, 8]. The widths of all the lines of the VO border are the same, and are equal to the width of the upper line of the border.

[See Also]

[HI_MPI_VO_SetVideoLayerPartitionMode](#)

HI_MPI_VO_GetChnBorder

[Description]

Obtains channel border attributes.

[Syntax]

```
HI_S32 HI_MPI_VO_GetChnBorder(VO_LAYER VoLayer, VO_CHN VoChn, VO_BORDER_S
*pstBorder);
```

[Parameter]

Parameter	Description	Input/Output
VoLayer	VO layer ID	Input
VoChn	VO channel ID Value range: [0, VO_MAX_CHN_NUM)	Input
pstBorder	Pointer to channel border attributes	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Value Range of VoLayer
Hi3516A/Hi3518E V200/Hi3519 V100	[0, VO_MAX_LAYER_NUM)

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a



[Note]

None

[Example]

None

[See Also]

None

HI_MPI_VO_SetChnReceiveThreshold

[Description]

Sets the display threshold for a VO channel. In PCIV cascade mode, when the slave chip transfers pictures to the master chip for previewing or displaying, the number of pictures to be displayed in the VO channel may be greater than the default threshold because the transfer interval is uneven. As a result, frames are lost or intermittence occurs during previewing or playback. In this case, this MPI is used to set the threshold to a larger value.

[Syntax]

```
HI_S32 HI_MPI_VO_SetChnReceiveThreshold(VO_LAYER VoLayer, VO_CHN VoChn,  
HI_U32 u32Threshold);
```

[Parameter]

Parameter	Description	Input/Output
VoLayer	VO layer ID	Input
VoChn	VO channel ID Value range: [0, VO_MAX_CHN_NUM)	Input
u32Threshold	Display threshold for a VO channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Value Range of VoLayer
Hi3516A/Hi3518E V200/Hi3519 V100	[0, VO_MAX_LAYER_NUM)



[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]

- The display threshold range for HD devices is [3, 8]. The default threshold 3 indicates that there are three frames to be displayed in the VO channel. The display threshold range for SD devices of the Hi3516A and Hi3519V100 is [3, 8], and the default threshold is 3. The display threshold range for SD devices of Hi3518E V200 is [2, 8]. The default threshold 2 indicates that the channel buffer queue can receive at most one frame. Because there is a displaying frame or a frame to be displayed, two frames are to be displayed in total.
- A smaller threshold indicates a smaller preview delay, and vice versa. Therefore, you are advised not to increase the threshold typically.
- A larger threshold indicates that there are more pictures to be displayed in the buffer queue and the memory usage may be higher. The threshold can be decreased to reduce the memory usage.
- The threshold can be dynamically adjusted. It is recommended that the threshold be set before the VO channel is enabled. You can call this MPI to restore the threshold to the default value.

[Example]

None

[See Also]

[HI_MPI_VO_GetChnReceiveThreshold](#)

HI_MPI_VO_GetChnReceiveThreshold

[Description]

Obtains the display threshold for a VO channel.

[Syntax]

```
HI_S32 HI_MPI_VO_GetChnReceiveThreshold(VO_LAYER VoLayer, VO_CHN VoChn,  
HI_U32 *pu32Threshold);
```

[Parameter]

Parameter	Description	Input/Output
VoLayer	VO layer ID	Input
VoChn	VO channel ID Value range: [0, VO_MAX_CHN_NUM)	Input
pu32Threshold	Display threshold for a VO channel pointer	Output

[Return Value]



Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Value Range of VoLayer
Hi3516A/Hi3518E V200/Hi3519 V100	[0, VO_MAX_LAYER_NUM)

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]

If you call HI_MPI_VO_GetChnReceiveThreshold after calling [HI_MPI_VO_SetChnReceiveThreshold](#), the threshold configured by calling [HI_MPI_VO_SetChnReceiveThreshold](#) is obtained.

[Example]

None

[See Also]

[HI_MPI_VO_SetChnReceiveThreshold](#)

HI_MPI_VO_GetChnRegionLuma

[Description]

Obtains the total luminance of the regions in a specified picture. This MPI is used to collect statistics on the luminance of cover regions. Then colors are inverted based on the luminance statistics, making the cover regions more obvious.

[Syntax]

```
HI_S32 HI_MPI_VO_GetChnRegionLuma(VO_LAYER VoLayer, VO_CHN VoChn,  
VO_REGION_INFO_S *pstRegionInfo, HI_U32 *pu32LumaData, HI_S32 s32MilliSec);
```

[Parameter]

Parameter	Description	Input/Output
VoLayer	VO layer ID	Input
VoChn	VO channel ID Value range: [0, VO_MAX_CHN_NUM)	Input



Parameter	Description	Input/Output
pstRegionInfo	Region information. pstRegionInfo->pstRegion indicates the attributes of statistics regions including the start position, width, and height. pstRegionInfo->u32RegionNum indicates the number of statistics regions.	Input
pu32LumaData	Pointer to memory for storing total region luminance information. The memory size must be greater than or equal to sizeof(HI_U32) multiplied by pstRegionInfo->u32RegionNum .	Output
s32MilliSec	Timeout period. If s32MilliSec is -1 , this MPI is a block MPI; if s32MilliSec is 0 , this MPI is a non-block MPI; If s32MilliSec is greater than 0, the value indicates the timeout period, and its unit is ms.	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Value Range of VoLayer
Hi3516A/Hi3518E V200/Hi3519 V100	The Hi3516A/Hi3518E V200/Hi3519 V100 does not support this MPI.

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]

- Enable a VO channel before calling this MPI.
- This MPI applies only to the video layer channels in VO_PART_MODE_SINGLE mode; otherwise, an error code is returned. The luminance information in VO_PART_MODE_MULTI mode can be obtained from the VPSS.
- A maximum of 64 statistics regions is supported.
- The start coordinates of the statistics regions are determined as follows:
 - If the crop function is disabled, the start coordinates are determined based on the coordinates of the upper left corner of the original picture.



- If the crop function is enabled, the start coordinates are determined based on the coordinates of the upper left corner of the cropped picture.

[Example]

None

[See Also]

[VO_REGION_INFO_S](#)

HI_MPI_VO_SetGraphicLayerCSC

[Description]

Sets the effect of the output pictures from a graphics layer.

[Syntax]

```
HI_S32 HI_MPI_VO_SetGraphicLayerCSC(GRAPHIC_LAYER GraphicLayer, const  
VO_CSC_S *pstCSC);
```

[Parameter]

Parameter	Description	Input/Output
GraphicLayer	Graphics layer ID Value range: [0, VOU_GRAPHICS_LAYER_NUM)	Input
pstCSC	Pointer to the output effect of a graphics layer The value range of u32Luma , u32Contrast , u32Hue , or u32Satuature is [0, 100], and the default value is 50. The default value of enCscMatrix is VO_CSC_MATRIX_RGB_TO_BT601_PC.	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]

Chip	Description
Hi3516A/Hi3518E V200/Hi3519 V100	The Hi3516A/Hi3518E V200/Hi3519 V100 does not support this MPI.



[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]

- This MPI is used to adjust the output effect of a graphics layer, including the luminance, contrast, hue, and saturation. The effect value ranges from 0 to 100.
- The CSC matrix must be **VO_CSC_MATRIX_RGB_TO_BT601_PC** or **VO_CSC_MATRIX_RGB_TO_BT709_PC**.
- If the video layer is disabled, the luminance, contrast, hue, and saturation are restored to default values.

[Example]

None

[See Also]

[HI_MPI_VO_GetGraphicLayerCSC](#)

HI_MPI_VO_GetGraphicLayerCSC

[Description]

Obtains the effect of the output pictures from a graphics layer.

[Syntax]

```
HI_S32 HI_MPI_VO_GetGraphicLayerCSC(GRAPHIC_LAYER GraphicLayer, VO_CSC_S
*pstCSC);
```

[Parameter]

Parameter	Description	Input/Output
GraphicLayer	Graphics layer ID Value range: [0, VOU_GRAPHICS_LAYER_NUM)	Input
pstCSC	Pointer to the output effect of a graphics layer	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 4.5 "Error Codes."

[Difference]



Chip	Description
Hi3516A/Hi3518E V200/Hi3519 V100	The Hi3516A/Hi3518E V200/Hi3519 V100 does not support this MPI.

[Requirement]

- Header files: mpi_vo.h, hi_comm_vo.h
- Library file: libmpi.a

[Note]

This MPI is used to obtain the output effect of a graphics layer, including the luminance, contrast, hue, and saturation. The effect value ranges from 0 to 100.

[Example]

None

[See Also]

[HI_MPI_VO_SetGraphicLayerCSC](#)

4.4 Data Structures

The VO data structures are as follows:

- [VO_MAX_DEV_NUM](#): Defines the maximum number of VO devices.
- [VO_MAX_LAYER_NUM](#): Defines the maximum number of video layers.
- [VO_MAX_CHN_NUM](#): Defines the maximum number of VO channels.
- [VO_DEV](#): Defines the device ID.
- [VO_LAYER](#): Defines the video layer ID.
- [VO_PUB_ATTR_S](#): Defines the attributes of a VO device.
- [VO_VIDEO_LAYER_ATTR_S](#): Defines the attributes of a video layer.
- [VO_PART_MODE_E](#): Defines the split mode of a video layer.
- [VO_CHN_ATTR_S](#): Defines the attributes of a VO channel.
- [VO_DISPLAY_FIELD_E](#): Defines the field mode during video output.
- [VO_QUERY_STATUS_S](#): Defines the status of a VO channel.
- [VO_BORDER_S](#): Defines border attributes.
- [VOU_ZOOM_IN_E](#): Defines the partial zoom-in type.
- [VO_ZOOM_RATIO_S](#): Defines the partial zoom-in ratio.
- [VO_ZOOM_ATTR_S](#): Defines the partial zoom-in attributes.
- [VO_CSC_S](#): Defines the picture output effect.
- [VO_CSC_MATRIX_E](#): Defines the CSC matrix.

VO_MAX_DEV_NUM

[Description]



Defines the maximum number of VO devices.

[Syntax]

```
#define VO_MAX_DEV_NUM      1
```

[Note]

None

[See Also]

None

VO_MAX_LAYER_NUM

[Description]

Defines the maximum number of video layers.

[Syntax]

```
#define VO_MAX_LAYER_NUM      1
```

[Note]

None

[See Also]

None

VO_MAX_CHN_NUM

[Description]

Defines the maximum number of VO channels.

[Syntax]

Hi3516A:

```
#define VO_MAX_CHN_NUM      32
```

Hi3518E V200/Hi3519 V100:

```
#define VO_MAX_CHN_NUM      8
```

[Difference]

Chip	Description
Hi3516A	At most 32 channels are supported.
Hi3518E V200/Hi3519 V100	At most eight channels are supported.

[Note]

None



[See Also]

None

VO_DEV

[Description]

Defines the device ID.

[Syntax]

```
typedef HI_S32 VO_DEV;
```

[Member]

Chip	Description
Hi3516A/Hi3518E V200/Hi3519 V100	The VOU supports only one VO device. 0: DSD0

[Difference]

Chip	Description
Hi3516A	The SD device (DSD0) can output BT.1120, CVBS, and BT.656 signals. The BT.1120 signals must be output separately. The CVBS signals support the PAL and NTSC standards, and the BT.656 signals support the PAL and NTSC standards. In the PAL or NTSC standard, CVBS and BT.656 signals can be output at the same time.
Hi3518E V200	The SD device (DSD0) can output BT.656 and LCD signals. The BT.656 signals support the PAL and NTSC standards. The LCD signals support the 6-bit or 8-bit serial output.
Hi3519 V100	The SD device (DSD0) can output BT.1120, CVBS, BT.656, and LCD signals. The BT.1120 signals can only be output separately. The CVBS and BT.656 signals support the PAL and NTSC standards. In the PAL or NTSC standard, CVBS and BT.656 signals can be output at the same time. The LCD signals support the 6-/8-bit serial output and 16-bit parallel output.

[Note]

None

[See Also]

[VO_PUB_ATTR_S](#)

VO_LAYER

[Description]

Defines the video layer ID.



[Syntax]

```
typedef HI_S32 VO_LAYER;
```

[Member]

Chip	Description
Hi3516A/Hi3518E V200/Hi3519 V100	The VOU supports only one video layer. 0: VSD0 video layer (SD video layer 0)

[Difference]

Chip	Description
Hi3516A/Hi3518E V200/Hi3519 V100	The VSD video layer is always bound to the corresponding device.

[Note]

None

[See Also]

[VO_VIDEO_LAYER_ATTR_S](#)

VO_PUB_ATTR_S

[Description]

Defines the public attributes of a VO device.

[Syntax]

```
typedef struct hivo_pub_attr_s
{
    HI_U32                 u32BgColor;           /*Background color of a device,
    in RGB format.*/
    VO_INTF_TYPE_E          enIntfType;          /*Type of a VO interface*/
    VO_INTF_SYNC_E          enIntfSync;          /*Type of a VO interface
    timing*/
    VO_SYNC_INFO_S          stSyncInfo;          /*Information about VO interface
    timings*/
} VO_PUB_ATTR_S;
```

[Member]

Member	Description
u32BgColor	Background color of a VO device, in RGB888 format. Only the lower 24 bits are valid.



Member	Description
enIntfType	<p>Typical configuration of the interface type. The prototype is defined as follows:</p> <pre>typedef HI_S32 VO_INTF_TYPE_E; #define VO_INTF_CVBS (0x01L<<0) #define VO_INTF_YPBPR (0x01L<<1) #define VO_INTF_VGA (0x01L<<2) #define VO_INTF_BT656 (0x01L<<3) #define VO_INTF_BT1120 (0x01L<<4) #define VO_INTF_HDMI (0x01L<<5) #define VO_INTF_LCD (0x01L<<6) #define VO_INTF_BT656_H (0x01L<<7) #define VO_INTF_BT656_L (0x01L<<8) #define VO_INTF_LCD_6BIT (0x01L<<9) #define VO_INTF_LCD_8BIT (0x01L<<10) #define VO_INTF_LCD_16BIT (0x01L<<11)</pre>
enIntfSync	<p>Typical configuration of the interface timing. The prototype is defined as follows:</p> <pre>typedef enum hivo_INTF_SYNC_E { VO_OUTPUT_PAL = 0, VO_OUTPUT_NTSC, VO_OUTPUT_1080P24, VO_OUTPUT_1080P25, VO_OUTPUT_1080P30, VO_OUTPUT_720P50, VO_OUTPUT_720P60, VO_OUTPUT_1080I50, VO_OUTPUT_1080I60, VO_OUTPUT_1080P50, VO_OUTPUT_1080P60, VO_OUTPUT_576P50, VO_OUTPUT_480P60, VO_OUTPUT_800x600_60, VO_OUTPUT_1024x768_60, VO_OUTPUT_1280x1024_60, VO_OUTPUT_1366x768_60, VO_OUTPUT_1440x900_60, VO_OUTPUT_1280x800_60, VO_OUTPUT_1600x1200_60, VO_OUTPUT_1680x1050_60, VO_OUTPUT_1920x1200_60, VO_OUTPUT_640x480_60,</pre>



Member	Description
	<pre>VO_OUTPUT_960H_PAL, VO_OUTPUT_960H_NTSC, VO_OUTPUT_320X240_60, VO_OUTPUT_320X240_50, VO_OUTPUT_240X320_50, VO_OUTPUT_240X320_60 VO_OUTPUT_USER, VO_OUTPUT_BUTT } VO_INTF_SYNC_E;</pre>
stSyncInfo	<p>Structure of the interface timing. The prototype is defined as follows:</p> <pre>typedef struct tagVO_SYNC_INFO_S { HI_BOOL bSynm; /* Value range: [0, 1] */ HI_BOOL bIop; /* Value range: [0, 1] */ HI_U8 u8Intfb; /* Value range: [0, 255] */ HI_U16 u16Vact; /* Value range: [0, 4096] */ HI_U16 u16Vbb; /* Value range: [0, 256] */ HI_U16 u16Vfb; /* Value range: [0, 256] */ HI_U16 u16Hact; /* Value range: [0, 4096] */ HI_U16 u16Hbb; /* Value range: [0, 65536] */ HI_U16 u16Hfb; /* Value range: [0, 65536] */ HI_U16 u16Hmid; /* Value range: [0, 65536] */ HI_U16 u16Bvact; /* Value range: [0, 4096] */ HI_U16 u16Bvbb; /* Value range: [0, 256] */ HI_U16 u16Bvfb; /* Value range: [0, 256] */ HI_U16 u16Hpw; /* Value range: [0, 65536] */ HI_U16 u16Vpw; /* Value range: [0, 256] */ HI_BOOL bIdv; /* Value range: [0, 1] */ HI_BOOL bIhs; /* Value range: [0, 1] */ HI_BOOL bIvs; /* Value range: [0, 1] */ } VO_SYNC_INFO_S;</pre>

[Difference]

Chip	Supported Timings
Hi3516A	VO_INTF_CVBS VO_INTF_BT656 VO_INTF_BT1120



Chip	Supported Timings
Hi3518E V200	VO_INTF_BT656 VO_INTF_LCD_6BIT VO_INTF_LCD_8BIT
Hi3519 V100	VO_INTF_CVBS VO_INTF_BT656 VO_INTF_BT1120 VO_INTF_LCD_6BIT VO_INTF_LCD_8BIT VO_INTF_LCD_16BIT

[Note]

- The timing structure defined in **stSyncInfo** is valid only when the interface timing is set to **VO_OUTPUT_USER**, which indicates the user-defined timing.
- If you want to use multiple interface types at the same time, you can set **enIntfSync** as follows: **enIntfSync = VO_INTF_BT1120|VO_INTF_HDMI**
- When the interface type is set to **VO_INTF_CVBS**, the value range of **enIntfSync** is [0, 1].
- When the interface type is set to **VO_INTF_BT1120**, the value range of **enIntfSync** is [2, 12].
- When the interface type is set to **VO_INTF_HDMI**, or **VO_INTF_VGA**, the value range of **enIntfSync** is [2, 22].
- When the interface type is set to **VO_INTF_BT656**, the value range of **enIntfSync** is [0, 1].
- When the interface type is set to **VO_INTF_LCD_8BIT**, the value of **enIntfSync** is 25.
- When the interface type is set to **VO_INTF_LCD_6BIT**, the value range of **enIntfSync** is [26, 27].
- When the interface type is set to **VO_INTF_LCD_16BIT**, the value of **enIntfSync** is 28.
- The configured device attributes take effect after the VO device is restarted.
- When you customize the timing, you need to ensure the timing effectiveness and configure the VDP CRG registers. For details about the configuration of CRG registers, see chapter 3 "System" of the *Hi35xx Professional HD IP Camera SoC Data Sheet*.
- The **VO_OUTPUT_960H_PAL**, **VO_OUTPUT_960H_NTSC**, **VO_OUTPUT_1600x1200_60**, and **VO_OUTPUT_1920x1200_60** standards are not supported.

[See Also]

[HI_MPI_VO_SetPubAttr](#)

VO_VIDEO_LAYER_ATTR_S

[Description]

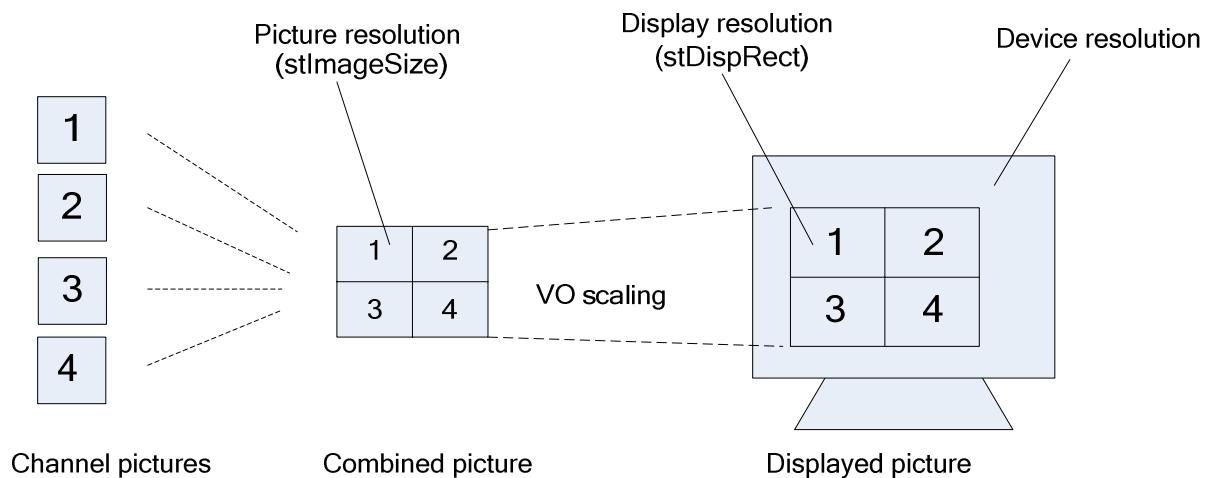
Defines the attributes of a video layer.



The attributes of a video layer include the device resolution, display resolution, and picture resolution. [Figure 4-1](#) illustrates each resolution.

- The picture resolution is the size of the canvas on which the picture of each channel is placed.
- The display resolution is the size of the display region that is obtained after the VOU scales the canvas.
- The device resolution is the same as the resolution defined in the device timing. For example, if the resolution defined in the timing is 1920 x 1080, the device resolution is 1920 x 1080.

Figure 4-1 Concepts related to video layer attributes



The memory of the video layer can be used in cluster mode and non-cluster mode.

- Cluster mode
 - The size of the allocated memory depends on the total resolutions of actual display channels.
 - This mode applies only to the VHD and PIP video layers.
 - The video layer cannot be zoomed in.
 - After pictures are combined, the display position of channel pictures can be adjusted. For the VHD and PIP video layers, the channel display position can be adjusted by calling `HI_MPI_VO_SetChnDisPos`.
- Non-cluster mode
 - The size of the allocated memory depends on the scaling ratio of the picture resolution to the display resolution and the region specified by the start coordinates (0, 0) of the displayed picture and the coordinates of the lower rightmost corner.
 - The video layer can be zoomed in. This function allows you to reduce the picture resolution at an appropriate ratio, which reduces the required memory and the picture quality if the display resolution is the same.
 - After pictures are combined, the display position of channel pictures cannot be adjusted.



Figure 4-2 Video layer scenario A (cluster memory)

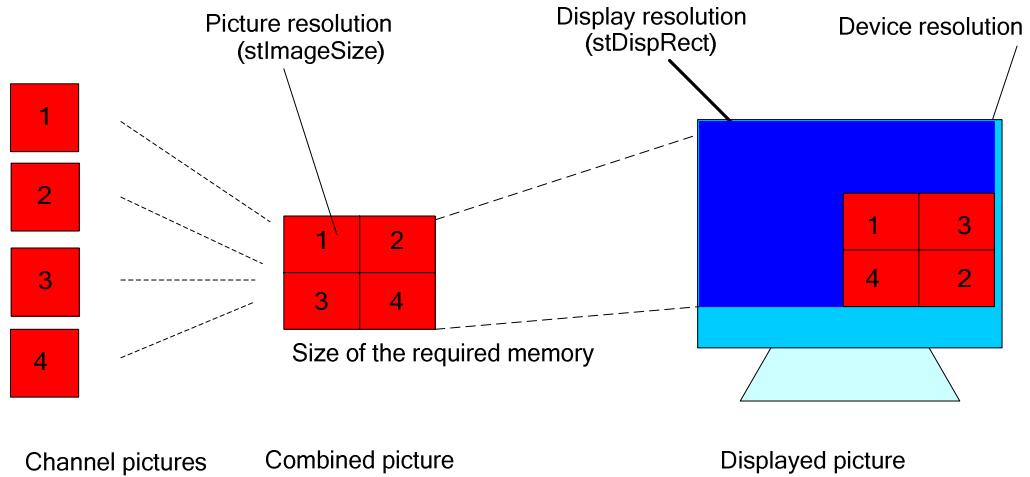
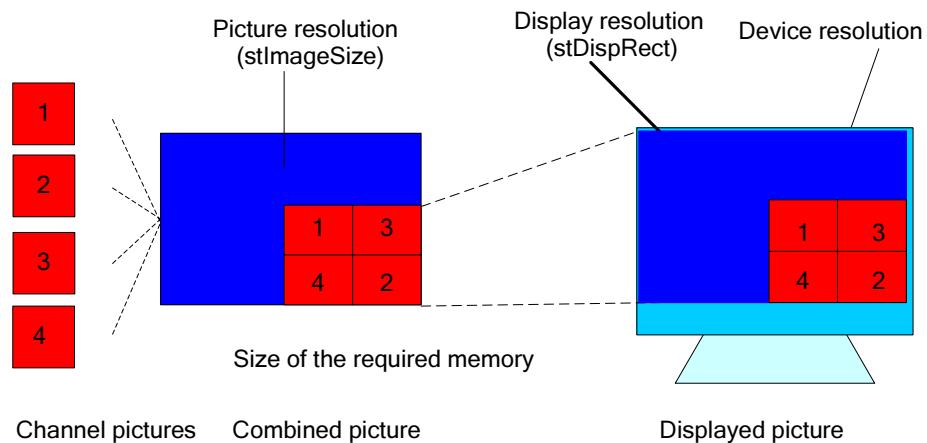


Figure 4-3 Video layer scenario B (non-cluster memory)



[Syntax]

```
typedef struct hivo_VIDEO_LAYER_ATTR_S
{
    RECT_S stDispRect;           /*Display resolution*/
    SIZE_S stImageSize;          /*Canvas size of the video layer*/
    HI_U32 u32DispFrmRt;        /*Display frame rate*/
    PIXEL_FORMAT_E enPixFormat; /*Pixel format of the video layer*/
    HI_BOOL bDoubleFrame; /* Whether to double the frame rate of the video
layer */
    HI_BOOL bClusterMode; /*Whether the channels at the video layer uses
the memory in cluster mode*/
} VO_VIDEO_LAYER_ATTR_S;
```



[Member]

Member	Description
stDispRect	Rectangular video display region
stImageSize	Picture resolution, that is, the size of the combined picture
u32DispFrmRt	Frame rate for video display
enPixelFormat	Input pixel format of the video layer
bDoubleFrame	Frame rate doubling switch of the video layer
bClusterMode	Memory cluster mode enable for the video layer

[Difference]

Chip	Display Capability	Input Pixel Format
Hi3516A	The maximum resolution of the VSD layer is 1920 x 1080 (up to 1920 x 1200 when the user timing is used). The value of stDispRect must be less than or equal to 1920 x 1080.	SPYCbCr420 or SPYCbCr422
Hi3518E V200	The maximum resolution of the VSD layer is 720 x 576. The value of stDispRect must be less than or equal to 720 x 576.	SPYCbCr420, SPYCbCr422, or single component
Hi3519 V100	The maximum resolution of the VSD layer is 1920 x 1080 (up to 1920 x 1200 when the user timing is used). The value of stDispRect must be less than or equal to 1920 x 1080.	SPYCbCr420, SPYCbCr422, or single component

[Note]

- The value of **stDispRect** cannot exceed the device resolution. **stImageSize** and **stDispRect** must be greater than or equal to the minimum resolution 32 x 32.
- The HD device supports scaling but not cropping. If **stImageSize** is less than **stDispRect**, the picture at the video layer is zoomed in from **stImageSize** to **stDispRect**. Note that **stImageSize** must be less than or equal to **stDispRect**.
- The SD device supports cropping rather than scaling. If **stImageSize** is greater than **stDispRect**, the combined picture is cropped.
- For the SD device, **stImageSize** or **stDispRect** must be 2-pixel-aligned.
- For the HD device, the values of **stImageSize** and **stDispRect** must be 2-pixel-aligned.
- In interlaced timing output mode, if the pixel format is SPYCbCr420, it is recommended that the height of **stImageSize** be 4-pixel-aligned. If the height is not 4-pixel-aligned, display exceptions may occur.



- **u32DispFrmRt** is the frame rate for video display. The value range is (0, VO_DISP_MAX_FRMRATE]. Typically, it is set to the full frame rate of the channel.
 - **u32DispFrmRt** is set to **25** if the source is in PAL standard.
 - **u32DispFrmRt** is set to **30** if the source is in NTSC standard.If the configured frame rate is greater than the source frame rate of the channel, the performance is not used efficiently.
- The SD device does not support VO scaling. Therefore, for the output of the SD device, the picture resolution is always equal to the display resolution.
- The HD device supports only VO zoom-in. That is, the picture resolution cannot be greater than the display resolution.
- The start position (**s32X**, **s32Y**) of **stDispRect** for the SD device or HD device of the Hi3516A/Hi3518E V200 is invalid. The start coordinates of the video layer are fixed at (0, 0). You can specify any display position by setting the start position of each channel.
- Hi3516A, Hi3518E V200, and Hi3519 V100 do not support the function of doubling the frame rate.

[See Also]

[HI_MPI_VO_SetVideoLayerAttr](#)

VO_PART_MODE_E

[Description]

Defines the split mode of a video layer.

[Syntax]

```
typedef enum hivo_PART_MODE_E
{
    VO_PART_MODE_SINGLE = 0,
    VO_PART_MODE_MULTI = 1,
    VO_PART_MODE_BUTT
} VO_PART_MODE_E;
```

[Member]

Member	Description
VO_PART_MODE_SINGLE	The video layer is configured based on a single region for the display device.
VO_PART_MODE_MULTI	The video layer is configured based on multiple regions for the display device.

[Note]

VO software cannot combine the pictures from multiple channels. Typically, the pictures are managed and then displayed based on multiple regions by using hardware or are internally combined by the VGS and then displayed as a full picture. The preceding processing modes correspond to VO_PART_MODE_MULTI and VO_PART_MODE_SINGLE of VO_PART_MODE_E respectively.



[See Also]

[HI_MPI_VO_SetVideoLayerPartitionMode](#)

VO_CHN_ATTR_S

[Description]

Defines the attributes of a VO channel.

[Syntax]

```
typedef struct hIVO_CHN_ATTR_S
{
    HI_U32 u32Priority;           /*Channel priority*/
    RECT_S stRect;               /*Channel display region*/
    HI_BOOL bDeflicker;          /*Anti-flicker enable*/
} VO_CHN_ATTR_S;
```

[Member]

Member	Description
u32Priority	Overlay priority of a video channel (dynamic attribute). The picture in the channel with the highest priority is displayed on the top. This attribute applies only to the SD device. Value range: [0, VO_MAX_CHN_NUM – 1]
stRect	Rectangular display region of the channel (dynamic attribute). The upper left corner of the screen acts as the coordinate origin. The value must be 2-pixel-aligned, and the rectangular area must fall within the screen range. Note that the height must be 4-pixel-aligned if the HD device timing is an interlaced timing and the display format is SPYCbCr420.
bDeflicker	Anti-flicker enable (dynamic attribute) HI_TRUE: enabled HI_FALSE: disabled Anti-flicker is valid only for the device channels whose pictures can be scaled by the VGS. That is, this parameter is valid only for SD devices.

[Difference]

Chip	Description
Hi3516A/Hi3518E V200/Hi3519 V100	<ul style="list-style-type: none">The priority range of SD devices is [0, VO_MAX_CHN_NUM – 1].The width or height of an SD device channel must be greater than or equal to 32 pixels.

[Note]



- A larger value indicates a higher priority.
For the SD device, the priority range is [0, VO_MAX_CHN_NUM – 1]. When the display regions of multiple channels overlap, the picture of the channel with a lower priority is overlaid with the picture of the channel with a higher priority. When the display regions of multiple channels with the same priority are overlaid, the pictures from the channel with a smaller ID are overlaid with the pictures from the channel with a larger ID.
- The size of the channel display region cannot exceed the canvas size **stImageSize** (one of video layer attributes).
- The channel attribute **stRect** of the HD device must be 2-pixel-aligned. If the interlaced timing is used and the pixel format is SEMIPLANAR_420, the height defined in **stRect** must be 4-pixel-aligned. The width or height of an HD channel must be greater than or equal to 32 pixels.
- If the video layer of an HD device is zoomed in, the start position, width, and height defined in the **stRect** attribute are the values before the video layer is zoomed in. After the video layer is zoomed in, the display start position shifts, and the width and height increase based on the zoom-in ratio of the video layer.
- For the channel attribute **stRect** of the SD device, the channel width and height must be 2-pixel-aligned.

[See Also]

[HI_MPI_VO_SetChnAttr](#)

VO_REGION_INFO_S

[Description]

Defines the VO region information for obtaining the total region luminance.

[Syntax]

```
typedef struct hivo_REGION_INFO_S
{
    RECT_S *pstRegion;      /*region attribute*/
    HI_U32 u32RegionNum;    /*count of the region*/
}VO_REGION_INFO_S;
```

[Member]

Member	Description
pstRegion	Region attributes, including the start position, width, and height
u32RegionNum	Number of regions

[Note]

The start position, width, and height of a region must be 2-pixel-aligned.

[See Also]

- [RECT_S](#)



- [HI_MPI_VO_GetChnRegionLuma](#)

VO_DISPLAY_FIELD_E

[Description]

Defines the field mode during video output.

[Syntax]

```
typedef enum hivo_DISPLAY_FIELD_E
{
    VO_FIELD_TOP,
    VO_FIELD_BOTTOM,
    VO_FIELD_BOTH,
    VO_FIELD_BUTT
}VO_DISPLAY_FIELD_E;
```

[Member]

Member	Description
VO_FIELD_TOP	Processing only the top field during video output
VO_FIELD_BOTTOM	Processing only the bottom field during video output
VO_FIELD_BOTH	Processing both the top field and the bottom field during video output

[Note]

None

[See Also]

- [HI_MPI_VO_SetChnField](#)
- [HI_MPI_VO_GetChnField](#)

VO_QUERY_STATUS_S

[Description]

Defines the status of a VO channel.

[Syntax]

```
typedef struct hivo_QUERY_STATUS_S
{
    HI_U32 u32ChnBufUsed;          /* channel buffer that been occupied */
} VO_QUERY_STATUS_S;
```

[Member]



Member	Description
u32ChnBufUsed	Number of VBs occupied by the VO channel

[Note]

None

[See Also]

[HI_MPI_VO_QueryChnStat](#)

VO_BORDER_S

[Description]

Defines border attributes.

[Syntax]

```
typedef struct hivo_BORDER_S
{
    HI_BOOL bBorderEn;           /*do Frame or not*/
    BORDER_S stBorder;
}VO_BORDER_S;
```

[Member]

Member	Description
bBorderEn	Border enable
stBorder	Border attributes, including the width and color of each border

[Note]

The border width must be an even number that falls within [2, 8]. The widths of all the lines of the VO border are the same, and are equal to the width of the upper line of the border.

[See Also]

- [VO_BORDER_S](#)
- [HI_MPI_VO_SetChnBorder](#)
- [HI_MPI_VO_GetChnBorder](#)

VOU_ZOOM_IN_E

[Description]

Defines the partial zoom-in type.

[Syntax]

```
typedef enum hivou_ZOOM_IN_E
```



```
{  
    VOU_ZOOM_IN_RECT = 0,  
    VOU_ZOOM_IN_RATIO,  
    VOU_ZOOM_IN_BUTT  
} VOU_ZOOM_IN_E;
```

[Member]

Member	Description
VOU_ZOOM_IN_RECT	Partial zoom-in by rectangle
VOU_ZOOM_IN_RATIO	Partial zoom-in by ratio
VOU_ZOOM_IN_BUTT	Invalid configuration

[Note]

None

[See Also]

[HI_MPI_VO_SetZoomInWindow](#)

VO_ZOOM_RATIO_S

[Description]

Defines the partial zoom-in ratio.

[Syntax]

```
typedef struct hivo_ZOOM_RATIO_S  
{  
    HI_U32 u32XRatio;  
    HI_U32 u32YRatio;  
    HI_U32 u32WRatio;  
    HI_U32 u32HRatio;  
} VO_ZOOM_RATIO_S;
```

[Member]

Member	Description
u32XRatio	Based on the screen coordinates, the ratio of the horizontal coordinate of the start point of the area to be zoomed to the width of the display channel
u32YRatio	Based on the screen coordinates, the ratio of the vertical coordinate of the start point of the area to be zoomed to the height of the display channel
u32WRatio	Based on the screen coordinates, the ratio of the width of the area to be zoomed to the width of the display channel



Member	Description
u32HRatio	Based on the screen coordinates, the ratio of the height of the area to be zoomed to the height of the display channel

[Note]

- The value range of the preceding members is [0, 1000].
- The member value is 1000 times of the ratio. For example, if the ratio of the width of the area to be zoomed to the width of the display channel is 0.6, the value of **u32WRatio** is **600**.
- To disable the partial zoom-in function, you only need to set the values of the preceding members to **0**.

[See Also]

[HI_MPI_VO_SetZoomInWindow](#)

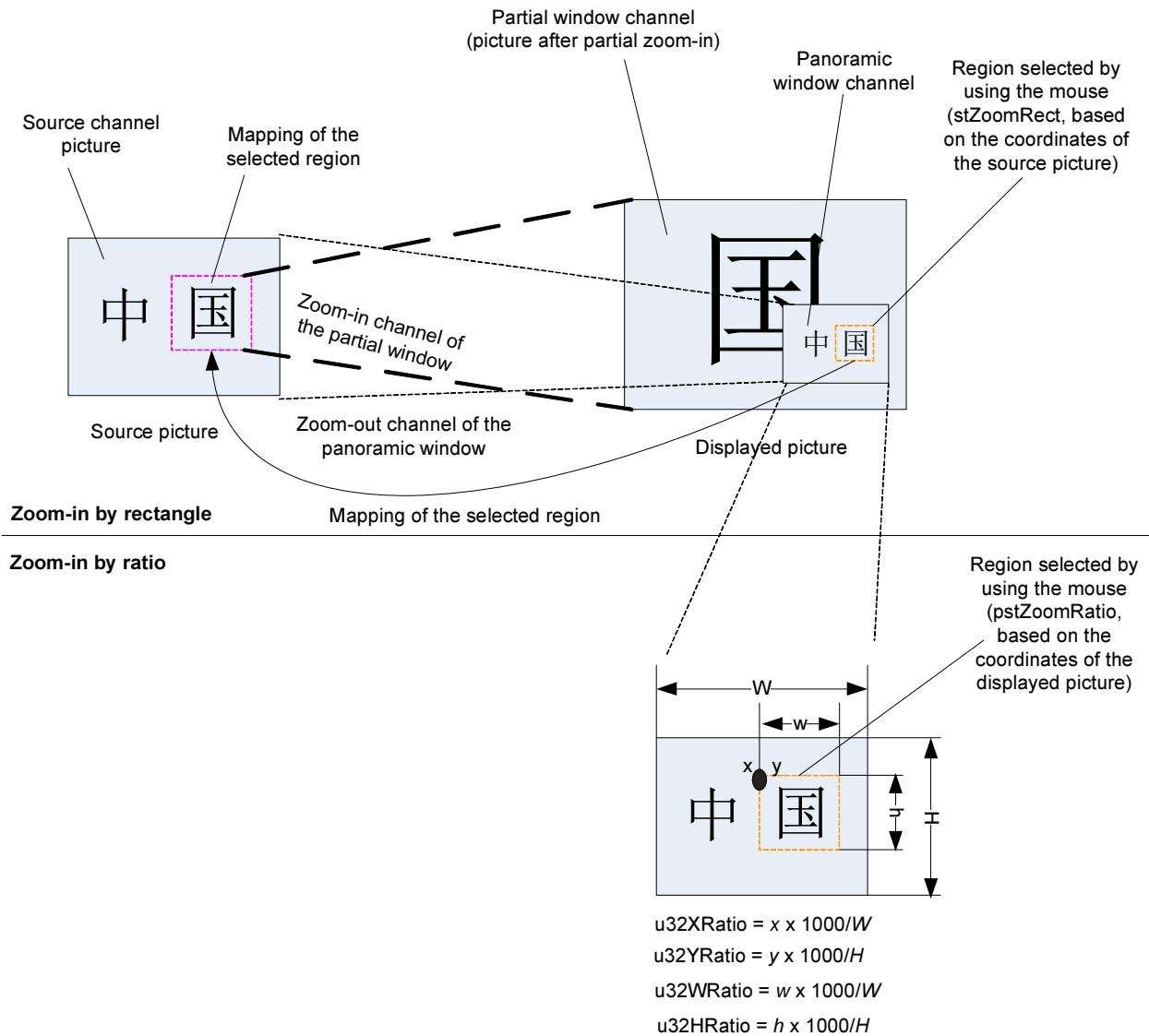
VO_ZOOM_ATTR_S

[Description]

Defines the partial zoom-in attributes. [Figure 4-4](#) shows the principle of partial zoom-in.



Figure 4-4 Principle of partial zoom-in



[Syntax]

```
typedef struct hivo_ZOOM_ATTR_S
{
    VOU_ZOOM_IN_E enZoomType;           /* choose the type of zoom in */
    union
    {
        RECT_S          stZoomRect;      /* zoom in by rect */
        VO_ZOOM_RATIO_S stZoomRatio;    /* zoom in by ratio */
    };
} VO_ZOOM_ATTR_S;
```

[Member]



Member	Description
enZoomType	Partial zoom-in type
stZoomRect	Coordinates of the rectangular partial zoom-in window
stZoomRatio	Ratio of the partial zoom-in window

[Difference]

Chip	Description
Hi3516A/Hi3518E V200/Hi3519 V100	For the SD device, the size of the partial zoom-in window must be greater than or equal to 32 x 32. That is, the width or height of the window must be greater than or equal to 32 pixels.

[Note]

- The source data of a VO channel can be cropped by rectangle or ratio during partial zoom-in.
 - When the source data is cropped by rectangle, **stZoomRect** must be set to a non-negative number and 2-pixel-aligned. The size of the partial zoom-in window must be greater than or equal to 32 x 32.
 - When the source data is cropped by ratio, **stZoomRatio** needs to be configured. The value range of **stZoomRatio** is [0, 1000]. The value **1000** indicates that the ratio is 1:1, and that the calculated width and height are 2-pixel-aligned. If the calculated width and height are less than 32, there is no zoom-in effect.
- The preceding two modes are based on the input picture but not the screen coordinates.
- To disable the partial zoom-in function, you can set **stZoomRect** or **stZoomRatio** to **(0, 0, 0, 0)**.
- If some parts of the partial zoom-in window exceed the input picture, the part where the source picture overlaps with the partial zoom-in window is truncated. If the partial zoom-in window exceeds the entire input picture in all directions (that is, the window can cover the entire input picture), there is no zoom-in effect.
- The ratio of the channel width/height to the width/height of the truncated window cannot be greater than 16.

[See Also]

[HI_MPI_VO_SetZoomInWindow](#)

VO_CSC_S

[Description]

Defines the picture output effect.

[Syntax]

```
typedef struct hivo_csc_s
{
```



```
VO_CSC_MATRIX_E enCscMatrix;  
HI_U32 u32Luma; /*Luminance: 0-100*/  
HI_U32 u32Contrast; /*Contrast: 0-100*/  
HI_U32 u32Hue; /*Hue: 0-100*/  
HI_U32 u32Satuature; /*Saturation: 0-100*/  
} VO_CSC_S;
```

[Member]

Member	Description
enCscMatrix	CSC matrix select
u32Luma	Luminance Value range: [0, 100]
u32Contrast	Contrast Value range: [0, 100]
u32Hue	Hue Value range: [0, 100]
u32Satuature	Saturation Value range: [0, 100]

[Note]

None

[See Also]

[HI_MPI_VO_SetVideoLayerCSC](#)

VO_CSC_MATRIX_E

[Description]

Defines the CSC matrix.

[Syntax]

```
typedef enum hiVO_CSC_MATRIX_E  
{  
    VO_CSC_MATRIX_IDENTITY = 0,  
    VO_CSC_MATRIX_BT601_TO_BT709,  
    VO_CSC_MATRIX_BT709_TO_BT601,  
    VO_CSC_MATRIX_BT601_TO_RGB_PC,  
    VO_CSC_MATRIX_BT709_TO_RGB_PC,  
    VO_CSC_MATRIX_RGB_TO_BT601_PC,  
    VO_CSC_MATRIX_RGB_TO_BT709_PC,  
    VO_CSC_MATRIX_BUTT
```



```
} VO_CSC_MATRIX_E;
```

[Member]

Member	Description
VO_CSC_MATRIX_IDENTITY	Identity matrix
VO_CSC_MATRIX_BT601_TO_BT709	CSC matrix from BT.601 to BT.709
VO_CSC_MATRIX_BT709_TO_BT601	CSC matrix from BT.709 to BT.601
VO_CSC_MATRIX_BT601_TO_RGB_PC	CSC matrix from BT.601 to RGB
VO_CSC_MATRIX_BT709_TO_RGB_PC	CSC matrix from BT.709 to RGB
VO_CSC_MATRIX_RGB_TO_BT601_PC	CSC matrix from RGB to BT.601
VO_CSC_MATRIX_RGB_TO_BT709_PC	CSC matrix from RGB to BT.709

[Note]

None

[See Also]

[HI_MPI_VO_SetVideoLayerCSC](#)

4.5 Error Codes

[Table 4-4](#) describes the error codes of VO MPIS.

Table 4-4 Error codes of VO MPIS

Error Code	Macro Definition	Description
0xA00F8001	HI_ERR_VO_INVALID_DEVID	The device ID does not fall within the value range.
0xA00F8002	HI_ERR_VO_INVALID_CHNID	The channel ID does not fall within the value range.
0xA00F8003	HI_ERR_VO_ILLEGAL_PARAM	The parameter value does not fall within the value range.
0xA00F8006	HI_ERR_VO_NULL_PTR	The pointer is null.
0xA00F8008	HI_ERR_VO_NOT_SUPPORT	The operation is not supported.
0xA00F8009	HI_ERR_VO_NOT_PERMIT	The operation is forbidden.
0xA00F800C	HI_ERR_VO_NO_MEM	The memory is insufficient.



Error Code	Macro Definition	Description
0xA00F8010	HI_ERR_VO_SYS_NOTREADY	The system is not initialized.
0xA00F8012	HI_ERR_VO_BUSY	The resources are unavailable.
0xA00F8040	HI_ERR_VO_DEV_NOT_CONFIG	The VO device is not configured.
0xA00F8041	HI_ERR_VO_DEV_NOT_ENABLE	The VO device is not enabled.
0xA00F8042	HI_ERR_VO_DEV_HAS_ENABLED	The VO device has been enabled.
0xA00F8043	HI_ERR_VO_DEV_HAS_BINDED	The device has been bound.
0xA00F8044	HI_ERR_VO_DEV_NOT_BINDED	The device is not bound.
0xA00F8045	HI_ERR_VO_VIDEO_NOT_ENABLE	The video layer is not enabled.
0xA00F8046	HI_ERR_VO_VIDEO_NOT_DISABLE	The video layer is not disabled.
0xA00F8047	HI_ERR_VO_VIDEO_NOT_CONFIG	The video layer is not configured.
0xA00F8048	HI_ERR_VO_CHN_NOT_DISABLE	The VO channel is not disabled.
0xA00F8049	HI_ERR_VO_CHN_NOT_ENABLE	No VO channel is enabled.
0xA00F804A	HI_ERR_VO_CHN_NOT_CONFIG	The VO channel is not configured.
0xA00F804B	HI_ERR_VO_CHN_NOT_ALLOC	No VO channel is allocated.
0xA00F804C	HI_ERR_VO_INVALID_PATTERN	The pattern is invalid.
0xA00F804D	HI_ERR_VO_INVALID_POSITION	The cascade position is invalid.
0xA00F804E	HI_ERR_VO_WAIT_TIMEOUT	Waiting times out.
0xA00F804F	HI_ERR_VO_INVALID_VFRAME	The video frame is invalid.
0xA00F8050	HI_ERR_VO_INVALID_RECT PARA	The rectangle parameter is invalid.
0xA00F8051	HI_ERR_VO_SETBEGIN_ALREADY	The SETBEGIN MPI has been configured.
0xA00F8052	HI_ERR_VO_SETBEGIN_NOTYET	The SETBEGIN MPI is not configured.
0xA00F8053	HI_ERR_VO_SETEND_ALREADY	The SETEND MPI has been configured.



Error Code	Macro Definition	Description
0xA00F8054	HI_ERR_VO_SETEND_NOTYET	The SETEND MPI is not configured.
0xA00F8065	HI_ERR_VO_GFX_NOT_DISABLE	The graphics layer is not disabled.
0xA00F8066	HI_ERR_VO_GFX_NOT_BIND	The graphics layer is not bound.
0xA00F8067	HI_ERR_VO_GFX_NOT_UNBIND	The graphics layer is not unbound.
0xA00F8068	HI_ERR_VO_GFX_INVALID_ID	The graphics layer ID does not fall within the value range.
0xA00F806b	HI_ERR_VO_CHN_AREA_OVERLAP	The VO channel areas overlap.
0xA00F806d	HI_ERR_VO_INVALID_LAYERID	The video layer ID does not fall within the value range.
0xA00F806e	HI_ERR_VO_VIDEO_HAS_BINDED	The video layer has been bound.
0xA00F806f	HI_ERR_VO_VIDEO_NOT_BINDED	The video layer is not bound



Contents

5 VPSS	5-1
5.1 Overview	5-1
5.2 Function Description	5-1
5.2.1 Basic Concepts.....	5-1
5.2.2 Function Implementation	5-2
5.2.3 Data Processing Workflow.....	5-3
5.3 API Reference	5-12
5.4 Data Structures	5-98
5.5 Error Codes	5-145



Figures

Figure 5-1 Context relationship of the VPSS	5-3
Figure 5-2 Data processing workflow of the VPSS for the Hi3516A	5-3
Figure 5-3 Data processing workflow of the VPSS for Hi3518E V200.....	5-4
Figure 5-4 Data processing workflow of the VPSS for Hi3519 V100	5-4



Tables

Table 5-1 VPSS channel specifications 1	5-4
Table 5-2 VPSS channel specifications 2	5-11
Table 5-3 Error codes of VPSS MPIs	5-145



5 VPSS

5.1 Overview

The video processing subsystem (VPSS) preprocesses an input picture (such as denoising and deinterlacing), processes the picture in each channel separately (such as scaling and sharpening), and then outputs multiple pictures in various resolutions.

The VPSS supports various picture processing functions including frame rate control (FRC), cropping, noise reduction (NR), Lens distortion correction (LDC), rotate, cover/overlay, scaling, mirror/flip and fisheye.

5.2 Function Description

5.2.1 Basic Concepts

- Group

The VPSS provides a maximum of **VPSS_MAX_GRP_NUM** available groups. The maximum number of groups varies according to the chip. The VPSS hardware is multiplexed by various groups in time division multiplexing (TDM) mode. Each group has multiple channels. The number of channels varies according to solution implementation. For details, see the channel description.

- Channel

The VPSS group channels are classified into physical channels and extended channels. The VPSS provides multiple physical channels with the scaling and cropping functions. The extended channels support scaling. They use the outputs of physical channels as inputs after being bound to physical channels, scale pictures based on the target resolution configured by users, and then output the scaled pictures.

- Frame rate control (FRC)

FRC is classified into group frame rate control and channel frame rate control:

- Group frame rate control: It applies to the VI-VPSS offline solution. The group frame rate of receiving input pictures is controlled.
- Channel frame rate control: It applies to the VI-VPSS offline and online solution. The channel frame rate of processing pictures in the physical channels is controlled.

- Crop



The cropping modes include group cropping, physical channel cropping, and extended channel cropping.

- Group cropping: The VPSS crops input pictures.
- Physical channel cropping: The VPSS crops the output pictures of each physical channel.
- Extended channel cropping: The VPSS crops the output pictures of the extended channels by calling the VGS.

- DEI

The DEI function restores the interlaced video source to the progressive video source.

- NR

By setting NR parameters configuration, the Gaussian noises are removable from pictures. This enables the pictures to become smooth and reduces the encoding bit rate.

- Scaling

The VPSS can zoom in or zoom out on pictures.

- LDC

The pictures captured by some low-end lenses are easily distorted. Such pictures need to be corrected by using the LDC function based on the distortion degree.

- Cover

The video cover region of the VPSS is used to fill the output pictures with pure color blocks.

- Overlay

The video overlay region is used to load bitmaps in ARGB4444, ARGB1555, or ARGB8888 format and refresh the background color on groups.

- Border

The VPSS can add borders to output pictures.

- Backup node

The backup node indicates that of the source picture. Each group has a backup node that is used for backing up the source picture of the frame that is to be processed by hardware. The VPSS places the frame of the head node in the cache queue in the backup node in the following situations:

- If the picture of the head node in the cache queue requires to be processed by VPSS hardware, the VPSS places this picture in the backup node and replaces the source picture.

- If the bound back-end RX module requires that the VPSS place the picture of the head node in the cache queue in the backup node, the VPSS replaces the picture in the backup node even if this picture is not processed by hardware.

- Short delay

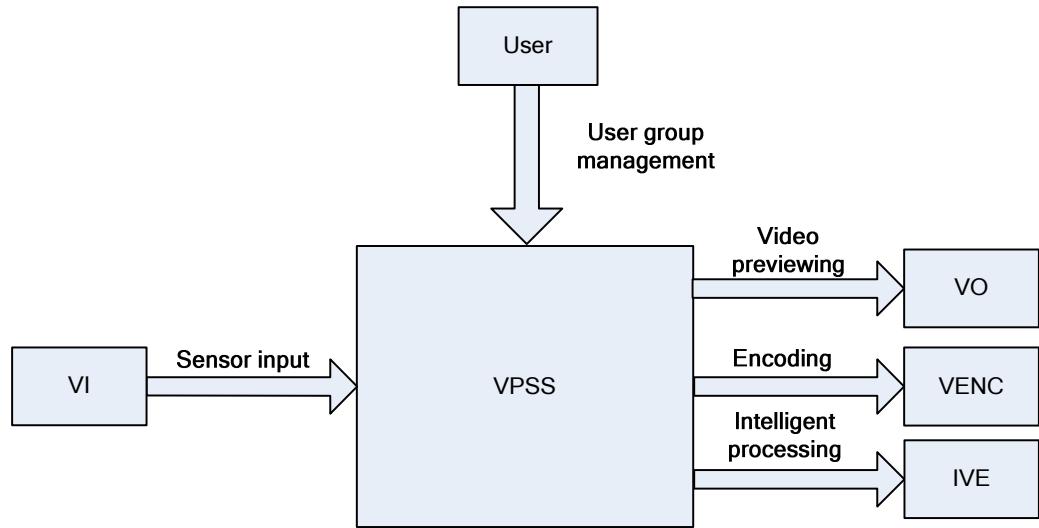
In the VI-VPSS online solution, if the encoder performance is sufficient, the VPSS can transmit pictures to the encoding module by line while capturing pictures. In this way, the VPSS does not need to transmit pictures to the encoding module only after a complete frame is processed, which reduces the delay.

5.2.2 Function Implementation

[Figure 5-1](#) shows the position of the VPSS in the system.



Figure 5-1 Context relationship of the VPSS



The VPSS can be bound to the VI and VO/VENC/IVE modules by calling the binding interface of the SYS module. The VO/VDEC modules are inputs of the VPSS, and the VO/VENC modules are receivers. You can manage groups by calling media processing platform programming interfaces (MPIs). Each group can be bound to only one input source. The physical channels of groups support automatic and user working modes. The working mode can be dynamically switched. The default working mode is automatic mode. In this mode, each channel can be bound to only one receiver. If you want to select the user mode, you need to call the related MPI and specify the required size and format of pictures. In user mode, each channel can be bound to multiple receivers. Note that the user mode is used when the pictures from a channel are encoded in multiple formats. In user mode, play control does not work. Therefore, the user mode is not recommended in the preview or playback scenario.

NOTE

The Hi3516A/Hi3518E V200/Hi3519 V100 supports only the user mode. If the low-power mode is enabled in VI-VPSS offline mode, the clock is disabled when the hardware of the VPSS module is not working, and reading or writing the VPSS registers may result in suspension. Therefore, it is recommended that the VPSS registers not be read or written in low-power mode.

5.2.3 Data Processing Workflow

[Figure 5-2](#) and [Figure 5-3](#) show the data processing workflow of the VPSS for the Hi3516A and Hi3518E V200/Hi3519 V100 respectively. [Table 5-1](#) and [Table 5-2](#) describe the channel specifications of the VPSS.

Figure 5-2 Data processing workflow of the VPSS for the Hi3516A

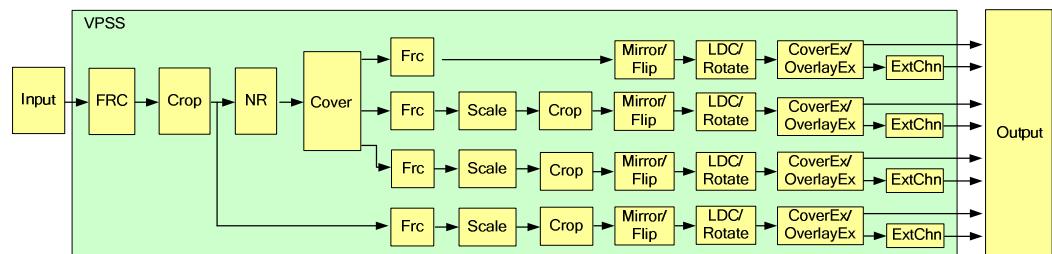




Figure 5-3 Data processing workflow of the VPSS for Hi3518E V200

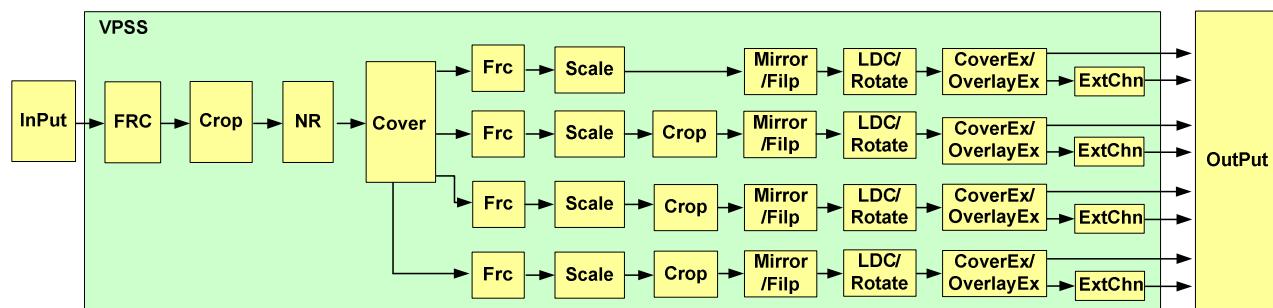


Figure 5-4 Data processing workflow of the VPSS for Hi3519 V100

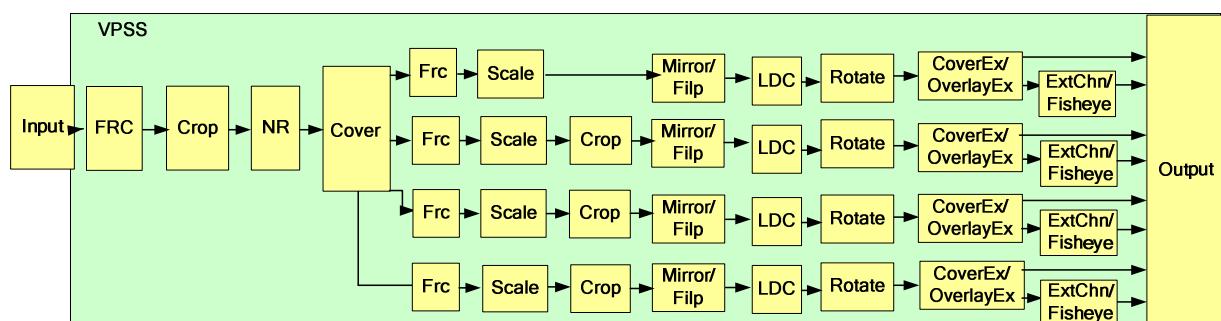


Table 5-1 VPSS channel specifications 1

Chip	Channel Name	Scaling	Sharpening	Field Capture	Recommended Application Scenario
Hi3516 A	Channel 0	Scaling and frame/field conversion not supported Output picture width (2-pixel-aligned): [VPSS_MIN_IMAGE_WIDTH, VPSS_MAX_IMAGE_WIDTH] Output picture height (2-pixel-aligned): [VPSS_MIN_IMAGE_HEIGHT, VPSS_MAX_IMAGE_HEIGHT]	N	N	Large stream encoding and JPEG snapshot
	Channel 1	At most VPSS_MAX_ZOOMOUT multiple horizontal or vertical zoom-out Output picture width (2-pixel-aligned):	N	N	Stream encoding



Chip	Channel Name	Scaling	Sharpening	Field Capture	Recommended Application Scenario
		<p>[VPSS_MIN_IMAGE_WIDTH, 1280]</p> <p>Output picture height (2-pixel-aligned):</p> <p>[VPSS_MIN_IMAGE_HEIGHT, VPSS_MAX_IMAGE_HEIGHT]</p> <p>Scaling is automatically not supported when the output picture width exceeds 1280.</p> <p>Zoom-in and frame/field conversion not supported</p>			
	Channel 2	<p>At most VPSS_MAX_ZOOMOUT multiple horizontal or vertical zoom-out</p> <p>Output picture width (2-pixel-aligned):</p> <p>[VPSS_MIN_IMAGE_WIDTH, 720]</p> <p>Output picture height (2-pixel-aligned):</p> <p>[VPSS_MIN_IMAGE_HEIGHT, VPSS_MAX_IMAGE_HEIGHT]</p> <p>Scaling is automatically not supported when the output picture width exceeds 720.</p> <p>Zoom-in and frame/field conversion not supported</p>	N	N	Stream encoding
	Channel 3	<p>At most VPSS_MAX_ZOOMOUT multiple horizontal or vertical zoom-out</p> <p>Output picture width (2-pixel-aligned):</p> <p>[VPSS_MIN_IMAGE_WIDTH, 1920]</p> <p>Output picture height (2-pixel-aligned):</p> <p>[VPSS_MIN_IMAGE_HEIGHT, VPSS_MAX_IMAGE_HEIGHT]</p> <p>Scaling is automatically not supported when the output picture width exceeds 1920.</p> <p>Zoom-in and frame/field conversion not supported</p>	N	N	Target picture required by the motion detection (MD) and intelligent video engine (IVE)



Chip	Channel Name	Scaling	Sharpening	Field Capture	Recommended Application Scenario
	Channels 4–11	<p>At most VPSS_EXT_CHN_MAX_ZOOM_OUT multiple horizontal or vertical zoom-out and VPSS_EXT_CHN_MAX_ZOOM_IN multiple horizontal or vertical zoom-in</p> <p>Output picture width (2-pixel-aligned): [VPSS_MIN_IMAGE_WIDTH, VPSS_MAX_IMAGE_WIDTH]</p> <p>Output picture height (2-pixel-aligned): [VPSS_MIN_IMAGE_HEIGHT, VPSS_MAX_IMAGE_HEIGHT]</p> <p>Frame/Field conversion not supported</p>	N	N	Secondary scaling
Hi3518E V200	Channel 0	<ul style="list-style-type: none">Offline mode: at most VPSS_MAX_ZOOMIN multiple zoom-in; zoom-out and frame/field conversion not supportedOnline mode: scaling and frame/field conversion not supported <p>In online and offline modes:</p> <p>Output picture width (4-pixel-aligned): [VPSS_MIN_IMAGE_WIDTH, VPSS_MAX_IMAGE_WIDTH]</p> <p>Output picture height (4-pixel-aligned): [VPSS_MIN_IMAGE_HEIGHT, VPSS_MAX_IMAGE_HEIGHT]</p>	N	N	Large stream encoding and JPEG snapshot
	Channel 1	<p>At most VPSS_MAX_ZOOMOUT multiple horizontal or vertical zoom-out</p> <p>Output picture width (4-pixel-aligned): [VPSS_MIN_IMAGE_WIDTH, VPSS_MAX_IMAGE_WIDTH]</p> <p>Output picture height (4-pixel-aligned): [VPSS_MIN_IMAGE_HEIGHT, VPSS_MAX_IMAGE_HEIGHT]</p>	N	N	Stream encoding



Chip	Channel Name	Scaling	Sharpening	Field Capture	Recommended Application Scenario
		Zoom-in and frame/field conversion not supported			
	Channel 2	At most VPSS_MAX_ZOOMOUT multiple horizontal or vertical zoom-out Output picture width (4-pixel-aligned): [VPSS_MIN_IMAGE_WIDTH , VPSS_MAX_IMAGE_WIDTH] Output picture height (4-pixel-aligned): [VPSS_MIN_IMAGE_HEIGHT , VPSS_MAX_IMAGE_HEIGHT] Zoom-in and frame/field conversion not supported	N	N	Stream encoding
	Channel 3	At most VPSS_MAX_ZOOMOUT multiple horizontal or vertical zoom-out Output picture width (4-pixel-aligned): [VPSS_MIN_IMAGE_WIDTH , VPSS_MAX_IMAGE_WIDTH] Output picture height (4-pixel-aligned): [VPSS_MIN_IMAGE_HEIGHT , VPSS_MAX_IMAGE_HEIGHT] Zoom-in and frame/field conversion not supported	N	N	Target picture required by the MD and IVE
	Channels 4-11	At most VPSS_EXT_CHN_MAX_ZOOM_OUT multiple horizontal or vertical zoom-out and VPSS_EXT_CHN_MAX_ZOOM_IN multiple horizontal or vertical zoom-in Output picture width (2-pixel-aligned): [VPSS_MIN_IMAGE_WIDTH , VPSS_EXTCHN_MAX_IMAGE_WIDTH] Output picture height (2-pixel-aligned): [VPSS_MIN_IMAGE_HEIGHT , VPSS_EXTCHN_MAX_IMAGE	N	N	Secondary scaling



Chip	Channel Name	Scaling	Sharpening	Field Capture	Recommended Application Scenario
		<u>_HEIGHT</u> Frame/Field conversion not supported			
Hi3519 V100	Channel 0	<ul style="list-style-type: none">Offline mode: at most <u>VPSS_MAX_ZOOMIN</u> multiple zoom-in; zoom-out and frame/field conversion not supported <p>Output picture width (4-pixel-aligned): <u>[VPSS_MIN_IMAGE_WIDTH, VPSS_OFFLINE_MAX_IMAGE_WIDTH]</u></p> <p>Output picture height (4-pixel-aligned): <u>[VPSS_MIN_IMAGE_HEIGHT, VPSS_MAX_IMAGE_HEIGHT]</u>.</p> <ul style="list-style-type: none">Online mode: scaling and frame/field conversion not supported <p>Output picture width (4-pixel-aligned): <u>[VPSS_MIN_IMAGE_WIDTH, VPSS_ONLINE_MAX_IMAGE_WIDTH]</u></p> <p>Output picture height (4-pixel-aligned): <u>[VPSS_MIN_IMAGE_HEIGHT, VPSS_MAX_IMAGE_HEIGHT]</u></p>	N	N	Large stream encoding and JPEG snapshot
	Channel 1	At most <u>VPSS_MAX_ZOOMOUT</u> multiple horizontal or vertical zoom-out <ul style="list-style-type: none">Offline mode: Output picture width (4-pixel-aligned): <u>[VPSS_MIN_IMAGE_WIDTH, VPSS_OFFLINE_MAX_IMAGE_WIDTH]</u> <p>Output picture height (4-pixel-aligned): <u>[VPSS_MIN_IMAGE_HEIGHT, VPSS_MAX_IMAGE_HEIGHT]</u></p> <ul style="list-style-type: none">Online mode: Output picture width (4-pixel-aligned):	N	N	Stream encoding



Chip	Channel Name	Scaling	Sharpening	Field Capture	Recommended Application Scenario
		<p>[VPSS_MIN_IMAGE_WIDTH, VPSS_ONLINE_MAX_IMAGE_WIDTH]</p> <p>Output picture height (4-pixel-aligned):</p> <p>[VPSS_MIN_IMAGE_HEIGHT, VPSS_MAX_IMAGE_HEIGHT]</p> <p>Zoom-in and frame/field conversion not supported</p>			
	Channel 2	<p>At most</p> <p>VPSS_MAX_ZOOMOUT</p> <p>multiple horizontal or vertical zoom-out</p> <ul style="list-style-type: none">• Offline mode: <p>Output picture width (4-pixel-aligned):</p> <p>[VPSS_MIN_IMAGE_WIDTH, VPSS_OFFLINE_MAX_IMAGE_WIDTH]</p> <p>Output picture height (4-pixel-aligned):</p> <p>[VPSS_MIN_IMAGE_HEIGHT, VPSS_MAX_IMAGE_HEIGHT]</p> <ul style="list-style-type: none">• Online mode: <p>Output picture width (4-pixel-aligned):</p> <p>[VPSS_MIN_IMAGE_WIDTH, VPSS_ONLINE_MAX_IMAGE_WIDTH]</p> <p>Output picture height (4-pixel-aligned):</p> <p>[VPSS_MIN_IMAGE_HEIGHT, VPSS_MAX_IMAGE_HEIGHT]</p> <p>Zoom-in and frame/field conversion not supported</p>	N	N	Stream encoding
	Channel 3	<p>At most</p> <p>VPSS_MAX_ZOOMOUT</p> <p>multiple horizontal or vertical zoom-out</p> <ul style="list-style-type: none">• Offline mode: <p>Output picture width (4-pixel-aligned):</p> <p>[VPSS_MIN_IMAGE_WIDTH, VPSS_OFFLINE_MAX_IMAGE_WIDTH]</p>	N	N	Target picture required by the MD and IVE



Chip	Channel Name	Scaling	Sharpening	Field Capture	Recommended Application Scenario
		<p>Output picture height (4-pixel-aligned): [VPSS_MIN_IMAGE_HEIGHT, VPSS_MAX_IMAGE_HEIGHT]</p> <ul style="list-style-type: none">• Online mode: Output picture width (4-pixel-aligned): [VPSS_MIN_IMAGE_WIDTH, VPSS_ONLINE_MAX_IMAGE_WIDTH] <p>Output picture height (4-pixel-aligned): [VPSS_MIN_IMAGE_HEIGHT, VPSS_MAX_IMAGE_HEIGHT]</p> <p>Zoom-in and frame/field conversion not supported</p>			
	Channels 4–11	<p>At most VPSS_EXT_CHN_MAX_ZOOM_OUT multiple horizontal or vertical zoom-out and VPSS_EXT_CHN_MAX_ZOOM_IN multiple horizontal or vertical zoom-in</p> <p>Output picture width (2-pixel-aligned): [VPSS_MIN_IMAGE_WIDTH, VPSS_EXTCHN_MAX_IMAGE_WIDTH]</p> <p>Output picture height (2-pixel-aligned): [VPSS_MIN_IMAGE_HEIGHT, VPSS_EXTCHN_MAX_IMAGE_HEIGHT]</p> <p>Frame/Field conversion not supported</p>	N	N	Secondary scaling Fisheye

NOTE

- The application scenarios in the **Recommended Application Scenario** column are the recommended schemes. You can use each channel as required.
- For Hi3519 V100, if the width of the input picture in offline mode (the actual picture after group cropping) exceeds 4096, neither compression nor mirroring is supported. If the output picture width exceeds 4096, neither compression nor rotation is supported.



Table 5-2 VPSS channel specifications 2

Chip	Channel	Cropping	Compression /Non-Compression	Pixel Format	Mirror/Flip	Online Low-Delay Mode
Hi3516 A	Channel 0	N	Y	Semi-planar 420 or semi-planar 422. Conversion between semi-planar 420 and semi-planar 422 is not supported.	Y	Y
	Channel 1	Y	Y	Semi-planar 420 or semi-planar 422. Conversion between semi-planar 420 and semi-planar 422 is supported when the output after scaling is less than or equal to 1280.	Y	Y
	Channel 2	Y	Y	Semi-planar 420 or semi-planar 422. Conversion between semi-planar 420 and semi-planar 422 is supported when the output after scaling is less than or equal to 720.	Y	Y
	Channel 3	Y	Y	Semi-planar 420 or semi-planar 422. Conversion between semi-planar 420 and semi-planar 422 is supported when the output after scaling is less than or equal to 1920.	Y	N
	Channels 4–11	Y	Y	Semi-planar 420 or semi-planar 422. Conversion between semi-planar 420 and semi-planar 422 is supported.	N	N
Hi3518 E V200	Channel 0	N	Y	Semi-planar 420, semi-planar 422, or YUV400. Conversion from semi-planar 422 to semi-planar 420 is supported. However, the conversion is not supported when 3DNR is enabled and the reference frames are large streams.	Y	Y
	Channel 1	Y	Y	Semi-planar 420, semi-planar 422, or YUV400. Conversion from semi-planar 422 to semi-planar 420 is supported.	Y	Y
	Channel 2	Y	Y	Semi-planar 420, semi-planar 422, or YUV400. Conversion from semi-planar 422 to semi-planar 420 is supported.	Y	Y
	Channel	Y	Only non-compression is	Semi-planar 420, semi-planar 422, or YUV400. Conversion from	Y	N



Chip	Channel	Cropping	Compression /Non-Compression	Pixel Format	Mirror/Flip	Online Low-Delay Mode
	3		supported.	semi-planar 422 to semi-planar 420 is supported.		
	Channels 4–11	Y	Y	Semi-planar 420, semi-planar 422, or YUV400. Conversion between semi-planar 420 and semi-planar 422, and conversion from semi-planar 420/semi-planar 422 to single component are supported.	N	N
Hi3519 V100	Channel 0	N	Y	Semi-planar 420, semi-planar 422, or YUV400. Conversion from semi-planar 422 to semi-planar 420 is supported. However, the conversion is not supported when 3DNR is enabled and the reference frames are large streams.	Y	Y
	Channel 1	Y	Y	Semi-planar 420, semi-planar 422, or YUV400. Conversion from semi-planar 422 to semi-planar 420 is supported.	Y	Y
	Channel 2	Y	Y	Semi-planar 420, semi-planar 422, or YUV400. Conversion from semi-planar 422 to semi-planar 420 is supported.	Y	Y
	Channel 3	Y	Y	Semi-planar 420, semi-planar 422, or YUV400. Conversion from semi-planar 422 to semi-planar 420 is supported.	Y	N
	Channels 4–11	Y	Y	Semi-planar 420, semi-planar 422, or YUV400. Conversion between semi-planar 420 and semi-planar 422, and conversion from semi-planar 420/semi-planar 422 to YUV400 are supported.	N	N

5.3 API Reference

The VPSS provides the following MPIs:

- [HI_MPI_VPSS_CreateGrp](#): Creates a VPSS group.
- [HI_MPI_VPSS_DestroyGrp](#): Destroys a VPSS group.
- [HI_MPI_VPSS_GetGrpAttr](#): Obtains VPSS group attributes.



- [HI_MPI_VPSS_SetGrpAttr](#): Sets VPSS group attributes.
- [HI_MPI_VPSS_StartGrp](#): Starts a VPSS group.
- [HI_MPI_VPSS_StopGrp](#): Disables a VPSS group.
- [HI_MPI_VPSS_ResetGrp](#): Resets a VPSS group.
- [HI_MPI_VPSS_GetChnAttr](#): Obtains VPSS channel attributes.
- [HI_MPI_VPSS_SetChnAttr](#): Sets VPSS channel attributes.
- [HI_MPI_VPSS_EnableChn](#): Enables a VPSS channel.
- [HI_MPI_VPSS_DisableChn](#): Disables a VPSS channel.
- [HI_MPI_VPSS_EnableBackupFrame](#): Enables backup frames.
- [HI_MPI_VPSS_DisableBackupFrame](#): Disables backup frames.
- [HI_MPI_VPSS_SetGrpParam](#): Sets the advanced attributes of a VPSS group.
- [HI_MPI_VPSS_GetGrpParam](#): Obtains the advanced attributes of a VPSS group.
- [HI_MPI_VPSS_SetGrpCrop](#): Sets the crop attributes of the VPSS.
- [HI_MPI_VPSS_GetGrpCrop](#): Obtains the crop attributes of the VPSS.
- [HI_MPI_VPSS_SendFrame](#): Sends data to a VPSS group.
- [HI_MPI_VPSS_SetChnMode](#): Sets the working mode of a VPSS channel.
- [HI_MPI_VPSS_GetChnMode](#): Obtains the working mode of a VPSS channel.
- [HI_MPI_VPSS_SetDepth](#): Sets the depth of the user picture queue.
- [HI_MPI_VPSS_GetDepth](#): Obtains the depth of the user picture queue.
- [HI_MPI_VPSS_GetChnFrame](#): Obtains a frame in user state.
- [HI_MPI_VPSS_ReleaseChnFrame](#): Releases the buffer for storing a frame.
- [HI_MPI_VPSS_GetGrpFrame](#): Obtains a source frame from a VPSS group.
- [HI_MPI_VPSS_ReleaseGrpFrame](#): Releases the buffer for storing a source frame.
- [HI_MPI_VPSS_SetChnParam](#): Sets channel advanced parameters.
- [HI_MPI_VPSS_GetChnParam](#): Obtains channel advanced parameters.
- [HI_MPI_VPSS_SetGrpFrameRate](#): Sets the frame rate control information for the VPSS.
- [HI_MPI_VPSS_GetGrpFrameRate](#): Obtains the frame rate control information for the VPSS.
- [HI_MPI_VPSS_SetGrpDelay](#): Sets the length of the delay queue for delaying the start of VPSS processing.
- [HI_MPI_VPSS_GetGrpDelay](#): Obtains the length of the delay queue for delaying the start of VPSS processing.
- [HI_MPI_VPSS_SetExtChnAttr](#): Sets the attributes of an extended VPSS channel.
- [HI_MPI_VPSS_GetExtChnAttr](#): Obtains the attributes of an extended VPSS channel.
- [HI_MPI_VPSS_SetChnOverlay](#): Enables or disables the video overlay region of a VPSS channel.
- [HI_MPI_VPSS_GetChnOverlay](#): Obtains the enable status of the video overlay region of a VPSS channel.
- [HI_MPI_VPSS_SetChnCover](#): Enables or disables the video cover region a VPSS channel.
- [HI_MPI_VPSS_GetChnCover](#): Obtains the enable status of the video cover region of a VPSS channel.
- [HI_MPI_VPSS_GetRegionLuma](#): Obtains the total luminance of the regions in a specified picture.



- [HI_MPI_VPSS_SetChnCrop](#): Sets the cropping attributes of a VPSS channel.
- [HI_MPI_VPSS_GetChnCrop](#): Obtains the cropping attributes of a VPSS channel.
- [HI_MPI_VPSS_SetLDCAttr](#): Sets the lens distortion correction (LDC) attributes of a VPSS channel.
- [HI_MPI_VPSS_GetLDCAttr](#): Obtains the LDC attributes of a VPSS channel.
- [HI_MPI_VPSS_SetRotate](#): Sets picture rotation attributes of a VPSS channel.
- [HI_MPI_VPSS_GetRotate](#): Obtains picture rotation attributes of a VPSS channel.
- [HI_MPI_VPSS_SetChnNR](#): Enables or disables NR of a VPSS channel.
- [HI_MPI_VPSS_GetChnNR](#): Obtains the NR enable status of a VPSS channel.
- [HI_MPI_VPSS_SetLowDelayAttr](#): Sets short delay attributes of a VPSS channel.
- [HI_MPI_VPSS_GetLowDelayAttr](#): Obtains short delay attributes of a VPSS channel.
- [HI_MPI_VPSS_SetRefSelect](#): Sets the NR reference frame source.
- [HI_MPI_VPSS_GetRefSelect](#): Obtains the NR reference frame source.
- [HI_MPI_VPSS_SetExtChnCrop](#): Sets the cropping attributes of an extended VPSS channel.
- [HI_MPI_VPSS_GetExtChnCrop](#): Obtains the cropping attributes of an extended VPSS channel.
- [HI_MPI_VPSS_SetGrpParamV2](#): Sets the VPSS 3DNR parameters (version 2). This MPI is recommended.
- [HI_MPI_VPSS_GetGrpParamV2](#): Obtains the VPSS 3DNR parameters (version 2). This MPI is recommended.
- [HI_MPI_VPSS_SetNRParam](#): Sets the VPSS 3DNR parameters.
- [HI_MPI_VPSS_GetNRParam](#): Obtains the VPSS 3DNR parameters.
- [HI_MPI_VPSS_SetFisheyeConfig](#): Sets the LMF parameters of the fisheye lens for the VPSS.
- [HI_MPI_VPSS_GetFisheyeConfig](#): Obtains the LMF parameters of the fisheye lens for the VPSS.
- [HI_MPI_VPSS_SetFisheyeAttr](#): Sets the fisheye attribute of an extended VPSS channel.
- [HI_MPI_VPSS_GetFisheyeAttr](#): Obtains the fisheye attribute of an extended VPSS channel.
- [HI_MPI_VPSS_SetModParam](#): Sets the VPSS module parameters.
- [HI_MPI_VPSS_GetModParam](#): Obtains the VPSS module parameters.
- [HI_MPI_VPSS_SetNRBParam](#): Sets the attributes of the VPSS 3DNR parameters.
- [HI_MPI_VPSS_GetNRBParam](#): Obtains the attributes of the VPSS 3DNR parameters.
- [HI_MPI_VPSS_SetNRV3Param](#): Sets the VPSS 3DNR parameters (V3 version).
- [HI_MPI_VPSS_GetNRV3Param](#): Obtains the VPSS 3DNR parameters (V3 version).
- [HI_MPI_VPSS_SetGrpNRSPParam](#): Sets the effect parameters of the standard 3DNR interface of the VPSS group.
- [HI_MPI_VPSS_GetGrpNRSPParam](#): Obtains the effect parameters of the standard 3DNR interface of the VPSS group.
- [HI_MPI_VPSS_SetGrpNRBParam](#): Sets the effect parameters of the advanced 3DNR interface of the VPSS group.
- [HI_MPI_VPSS_GetGrpNRBParam](#): Obtains the effect parameters of the advanced 3DNR interface of the VPSS group.



HI_MPI_VPSS_CreateGrp

[Description]

Creates a VPSS group.

[Syntax]

```
HI_S32 HI_MPI_VPSS_CreateGrp(VPSS_GRP VpssGrp, VPSS_GRP_ATTR_S  
*pstGrpAttr)
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
pstGrpAttr	Pointer to the VPSS group attributes	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

- In offline mode, multiple groups can be created, and the maximum group ID is **VPSS_MAX_GRP_NUM**. In online mode, only one group can be created, and the group ID must be 0.
- This MPI cannot be repeatedly called.
- In online mode, the timings must be synchronous for VI and VPSS logic processing. Therefore, group picture attributes must be the same as VI picture attributes when a group is created. Otherwise, a VPSS interrupt occurs. For details, see [VPSS_GRP_ATTR_S](#).
- For details about the conversion between the supported pixel formats and other pixel formats, see [Table 5-2](#).
- When the memory for the RefBuffer and MadBuffer is sufficient, the VPSS is capable of processing pictures with the total resolution (picture width x picture height) less than or equal to **u32MaxW** x **u32MaxH**.
 - The memory of the RefBuffer is calculated as follows:



RefBuffer for YUV420 pictures = (align(u32MaxW,16) x align(u32MaxH,16)/16) x 4/8 + align(u32MaxW,16) x align(u32MaxH,16) + align(u32MaxW,16) x align(u32MaxH,16)/2

RefBuffer for YUV422 pictures = (align(u32MaxW,16) x align(u32MaxH,16)/16) x 4/8 + align(u32MaxW,16) x align(u32MaxH,16) + align(u32MaxW,16) x align(u32MaxH,16)

- The memory of the MadBuffer is calculated as follows:

MadBuffer = Max((align(u32MaxW,16)/4) x (align(u32MaxH,16)/2),
(align(u32MaxH,16)/4) x (align(u32MaxW,16)/2))

[Example]

```
VPSS_GRP_ATTR_S stGrpVpssAttr;
VPSS_CHN_ATTR_S stChnAttr;
VPSS_CROP_INFO_S stCropInfo;
HI_S32 s32Ret = HI_SUCCESS;
VPSS_GRP VpssGrp;
VPSS_CHN VpssChn;
VpssGrp = 0;
VpssChn = 0;
stGrpVpssAttr.u32MaxW = 720;
stGrpVpssAttr.u32MaxH = 576;
stGrpVpssAttr.bIeEn = HI_FALSE;
stGrpVpssAttr.bDciEn = HI_FALSE;
stGrpVpssAttr.bNrEn = HI_FALSE;
stGrpVpssAttr.bHistEn = HI_FALSE;
stGrpVpssAttr.enDieMode = VPSS_DIE_MODE_NODIE;
stGrpVpssAttr.enPixFmt = PIXEL_FORMAT_YUV_SEMIPLANAR_422;
s32Ret = HI_MPI_VPSS_CreateGrp
(VpssGrp, &stGrpVpssAttr);
if(s32Ret != HI_SUCCESS)
{
    return s32Ret;
}

s32Ret = HI_MPI_VPSS_GetGrpAttr(VpssGrp, &stGrpVpssAttr);
if(s32Ret != HI_SUCCESS)
{
    return s32Ret;
}
stGrpVpssAttr.bIeEn = HI_FALSE;
stGrpVpssAttr.bNrEn = HI_TRUE;
s32Ret = HI_MPI_VPSS_SetGrpAttr(VpssGrp, &stGrpVpssAttr);
if(s32Ret != HI_SUCCESS)
{
    return s32Ret;
```



```
    }

    s32Ret = HI_MPI_VPSS_GetGrpCrop(VpssGrp, &stCropInfo);

    if(s32Ret != HI_SUCCESS)
    {
        return s32Ret;
    }

    stCropInfo.bEnable = HI_TRUE;
    stCropInfo.enCropCoordinate = VPSS_CROP_ABS_COOR;
    stCropInfo.stCropRect.s32X = 180;
    stCropInfo.stCropRect.s32Y = 252;
    stCropInfo.stCropRect.u32Width = 1920;
    stCropInfo.stCropRect.u32Height = 1080;

    s32Ret = HI_MPI_VPSS_SetGrpCrop (VpssGrp, &stCropInfo);
    if(s32Ret != HI_SUCCESS)
    {
        return s32Ret;
    }

    s32Ret = HI_MPI_VPSS_GetChnAttr(VpssGrp, VpssChn,&stChnAttr);
    if(s32Ret != HI_SUCCESS)
    {
        return s32Ret;
    }

    stChnAttr.bFrameEn = 0;
    stChnAttr.bSpEn = HI_FALSE;

    s32Ret = HI_MPI_VPSS_SetChnAttr(VpssGrp, VpssChn,&stChnAttr);
    if(s32Ret != HI_SUCCESS)
    {
        return s32Ret;
    }

    s32Ret = HI_MPI_VPSS_EnableChn(VpssGrp, VpssChn);
    if(s32Ret != HI_SUCCESS)
    {
        return s32Ret;
    }

    s32Ret = HI_MPI_VPSS_StartGrp (VpssGrp);
    if(s32Ret != HI_SUCCESS)
    {
        return s32Ret;
    }

    /*****call sys bind interface ****/
    /*      call sys bind interface      */
```



```
/*****************/
s32Ret = HI_MPI_VPSS_StopGrp (VpssGrp);
if(s32Ret != HI_SUCCESS)
{
    return s32Ret;
}

s32Ret = HI_MPI_VPSS_DisableChn(VpssGrp, VpssChn);
if(s32Ret != HI_SUCCESS)
{
    return s32Ret;
}
/*****************/
/*      call sys unbind interface */
/*****************/

s32Ret = HI_MPI_VPSS_DestroyGrp(VpssGrp);
if(s32Ret != HI_SUCCESS)
{
    return s32Ret;
}
```

[See Also]

- [HI_MPI_VPSS_DestroyGrp](#)
- [HI_MPI_VPSS_GetGrpAttr](#)
- [HI_MPI_VPSS_StartGrp](#)
- [HI_MPI_VPSS_StopGrp](#)
- [HI_MPI_VPSS_DestroyGrp](#)
- [HI_MPI_VPSS_SetChnAttr](#)
- [HI_MPI_VPSS_EnableChn](#)
- [HI_MPI_VPSS_DisableChn](#)
- [HI_MPI_VPSS_SetGrpCrop](#)
- [HI_MPI_VPSS_GetGrpCrop](#)

HI_MPI_VPSS_DestroyGrp

[Description]

Destroys a VPSS group.

[Syntax]

```
HI_S32 HI_MPI_VPSS_DestroyGrp(VPSS\_GRP VpssGrp)
```

[Parameter]



Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpmpi.a

[Note]

- A group must be created before this MPI is called.
- Before destroying a group by calling HI_MPI_VPSS_DestroyGrp, you must disable this group by calling [HI_MPI_VPSS_StopGrp](#).
- After this MPI is called, the VPSS group is destroyed only after the current task of this group is complete.

[Example]

See the example of [HI_MPI_VPSS_CreateGrp](#).

[See Also]

[HI_MPI_VPSS_CreateGrp](#)

HI_MPI_VPSS_GetGrpAttr

[Description]

Obtains VPSS group attributes.

[Syntax]

```
HI_S32 HI_MPI_VPSS_GetGrpAttr (VPSS_GRP VpssGrp, VPSS_GRP_ATTR_S
*pstVpssGrpAttr)
```

[Parameter]



Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
pstVpssGrpAttr	Pointer to VPSS group attributes	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

- A group must be created before this MPI is called.
- The group attributes must be valid, and some static attributes cannot be dynamically set. For details, see the description of [VPSS_GRP_ATTR_S](#).

[Example]

See the example of [HI_MPI_VPSS_CreateGrp](#).

[See Also]

[HI_MPI_VPSS_CreateGrp](#)

HI_MPI_VPSS_SetGrpAttr

[Description]

Sets VPSS group attributes.

[Syntax]

```
HI_S32 HI_MPI_VPSS_SetGrpAttr (VPSS_GRP VpssGrp, VPSS_GRP_ATTR_S  
*pstVpssGrpAttr)
```

[Parameter]



Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
pstVpssGrpAttr	Pointer to VPSS group attributes	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

- A group must be created before this MPI is called.
- The group attributes must be valid, and some static attributes cannot be dynamically set. For details, see the description of [VPSS_GRP_ATTR_S](#).

[Example]

See the example of [HI_MPI_VPSS_CreateGrp](#).

[See Also]

[HI_MPI_VPSS_CreateGrp](#)

HI_MPI_VPSS_StartGrp

[Description]

Starts a VPSS group.

[Syntax]

```
HI_S32 HI_MPI_VPSS_StartGrp(VPSS\_GRP VpssGrp)
```

[Parameter]



Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

- A group must be created before this MPI is called.
- A code indicating success is returned when this MPI is called repeatedly to set a VPSS group.

[Example]

See the example of [HI_MPI_VPSS_CreateGrp](#).

[See Also]

[HI_MPI_VPSS_CreateGrp](#)

HI_MPI_VPSS_StopGrp

[Description]

Disables a VPSS group.

[Syntax]

`HI_S32 HI_MPI_VPSS_StopGrp(VPSS_GRP VpssGrp)`

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input



[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

- A group must be created before this MPI is called.
- If a VPSS group is repeatedly disabled, the code indicating success is returned.

[Example]

See the example of [HI_MPI_VPSS_CreateGrp](#).

[See Also]

[HI_MPI_VPSS_CreateGrp](#)

HI_MPI_VPSS_ResetGrp

[Description]

Resets a VPSS group.

[Syntax]

`HI_S32 HI_MPI_VPSS_ResetGrp (VPSS_GRP VpssGrp)`

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input

[Return Value]

Return Value	Description
0	Success



Return Value	Description
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header file: mpi_vpss.h
- Library file: libmpi.a

[Note]

A group must be created before this MPI is called.

[Example]

See the example of [HI_MPI_VPSS_CreateGrp](#).

[See Also]

None

HI_MPI_VPSS_GetChnAttr

[Description]

Obtains VPSS channel attributes.

[Syntax]

```
HI_S32 HI_MPI_VPSS_GetChnAttr(VPSS_GRP VpssGrp, VPSS_CHN VpssChn,  
VPSS_CHN_ATTR_S *pstChnAttr)
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
VpssChn	VPSS channel ID Value range: [0, VPSS_MAX_PHY_CHN_NUM)	Input
pstChnAttr	VPSS channel attributes	Output

[Return Value]

Return Value	Description
0	Success



Return Value	Description
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

- A group must be created before this MPI is called.
- The extended channels do not support this MPI.

[Example]

See the example of [HI_MPI_VPSS_CreateGrp](#).

[See Also]

- [VPSS_CHN_ATTR_S](#)
- [HI_MPI_VPSS_CreateGrp](#)

HI_MPI_VPSS_SetChnAttr

[Description]

Sets VPSS channel attributes.

[Syntax]

```
HI_S32 HI_MPI_VPSS_SetChnAttr(VPSS_GRP VpssGrp, VPSS_CHN VpssChn,  
VPSS_CHN_ATTR_S *pstChnAttr)
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
VpssChn	VPSS channel ID Value range: [0, VPSS_MAX_PHY_CHN_NUM)	Input
pstChnAttr	VPSS channel attributes	Input

[Return Value]



Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

- A group must be created before this MPI is called.
- The extended channels do not support this MPI.

[Example]

See the example of [HI_MPI_VPSS_CreateGrp](#).

[See Also]

[HI_MPI_VPSS_GetChnAttr](#)

HI_MPI_VPSS_EnableChn

[Description]

Enables a bypass channel.

[Syntax]

```
HI_S32 HI_MPI_VPSS_EnableChn(VPSS_GRP VpssGrp, VPSS_CHN VpssChn)
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
VpssChn	VPSS channel ID Value range: [0, VPSS_MAX_CHN_NUM)	Input

[Return Value]

Return Value	Description
0	Success



Return Value	Description
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

- If a VPSS channel is repeatedly enabled, a code indicating success is returned.
- A group must be created before this MPI is called.
- If an extended channel is supported, ensure that the source physical channel bound to the extended channel is enabled. Otherwise, an error code indicating failure is returned.
- If a physical channel is used, [HI_MPI_VPSS_SetChnMode](#) must be called to set the working mode of the channel before [HI_MPI_VPSS_EnableChn](#) is called. Otherwise, an error code indicating failure is returned.

[Example]

See the example of [HI_MPI_VPSS_CreateGrp](#).

[See Also]

[HI_MPI_VPSS_DisableChn](#)

HI_MPI_VPSS_DisableChn

[Description]

Disables a VPSS channel.

[Syntax]

`HI_S32 HI_MPI_VPSS_DisableChn(VPSS_GRP VpssGrp, VPSS_CHN VpssChn)`

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
VpssChn	VPSS channel ID Value range: [0, VPSS_MAX_CHN_NUM)	Input

[Return Value]



Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

- If a VPSS channel is repeatedly disabled, a code indicating success is returned.
- A group must be created before this MPI is called.
- If an extended channel is supported, ensure that all the target physical channels bound to the extended channel are enabled. Otherwise, an error code indicating failure is returned.

[Example]

See the example of [HI_MPI_VPSS_CreateGrp](#).

[See Also]

[HI_MPI_VPSS_EnableChn](#)

HI_MPI_VPSS_EnableBackupFrame

[Description]

Enables backup frames.

[Syntax]

`HI_S32 HI_MPI_VPSS_EnableBackupFrame (VPSS_GRP VpssGrp)`

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM] Value in online mode: 0	Input

[Online/Offline Difference]

Mode	Description
Offline mode	This MPI is supported.
Online mode	This MPI is not supported.



[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

- A group must be created before this MPI is called.
- Backup frames are disabled by default.

[Example]

None

[See Also]

None

HI_MPI_VPSS_DisableBackupFrame

[Description]

Disables backup frames.

[Syntax]

```
HI_S32 HI_MPI_VPSS_DisableBackupFrame (VPSS\_GRP VpssGrp)
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM] Value in online mode: 0	Input

[Online/Offline Difference]

Mode	Description
Offline mode	This MPI is supported.
Online mode	This MPI is not supported.



[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

- A group must be created before this MPI is called.
- After backup frames are disabled, the memory required for storing a frame is saved for each VPSS group. However, calling [HI_MPI_VPSS_GetGrpFrame](#) returns failure.

[Example]

None

[See Also]

None

HI_MPI_VPSS_SetGrpParam

[Description]

Sets the advanced attributes of a VPSS group.

[Syntax]

```
HI_S32 HI_MPI_VPSS_SetGrpParam(VPSS_GRP VpssGrp, VPSS_GRP_PARAM_S
*pstVpssParam)
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
pstVpssParam	Advanced attributes	Input

[Difference]



Chip	Description
Hi3516A	This MPI is supported.
Hi3518E V200	This MPI is not supported.
Hi3519 V100	This MPI is not supported.

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

- A group must be created before this MPI is called.
- For details about the limitations on the **pstVpssParam** members, see the description of [VPSS_GRP_PARAM_S](#).

[Example]

None

[See Also]

[VPSS_GRP_PARAM_S](#)

HI_MPI_VPSS_GetGrpParam

[Description]

Obtains the advanced attributes of a VPSS group.

[Syntax]

```
HI_S32 HI_MPI_VPSS_GetGrpParam( VPSS_GRP *VpssGrp, VPSS_GRP_PARAM_S *pstVpssParam)
```

[Parameter]



Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
pstVpssParam	Advanced attributes	Output

[Difference]

Chip	Description
Hi3516A	This MPI is supported.
Hi3518E V200	This MPI is not supported.
Hi3519 V100	This MPI is not supported.

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

A group must be created before this MPI is called.

[Example]

None

[See Also]

[VPSS_GRP_PARAM_S](#)

HI_MPI_VPSS_SetGrpCrop

[Description]

Sets the crop attributes of the VPSS.

[Syntax]

`HI_S32 HI_MPI_VPSS_SetGrpCrop(VPSS_GRP VpssGrp, VPSS_CROP_INFO_S`



*pstCropInfo)

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
pstCropInfo	Crop function parameter	Input

[Online/Offline Difference]

Mode	Description
Offline mode	This MPI is supported.
Online mode	This MPI is not supported.

[Difference]

Chip	Description
Hi3516A	<ul style="list-style-type: none">When the coordinate type of the start point is VPSS_CROP_ABS_COOR (absolute coordinates), the valid values of s32X and s32Y must ensure that the maximum width and height of the cropped rectangle fall within the range supported by the VPSS.The workflow of processing pictures with different widths varies according to chips. Figure 5-2 shows the data processing workflow for the Hi3516A.The coordinates of the start point of the cropped region must be 2-pixel-aligned, and the width and height must be 2-pixel-aligned. If the coordinate type is VPSS_CROP_RATIO_COOR (relative coordinates), the coordinates of the start point of the cropped region are automatically rounded down to integral multiples of 2, and the width and height are automatically rounded down to integral multiples of 2.



Chip	Description
Hi3518E V200/Hi3519 V100	<ul style="list-style-type: none">When the coordinate type of the start point is VPSS_CROP_ABS_COOR (absolute coordinates), the valid values of s32X and s32Y must ensure that the maximum width and height of the cropped rectangle fall within the range supported by the VPSS.The workflow of processing pictures with different widths varies according to chips. Figure 5-3 shows the data processing workflow for Hi3518E V200/Hi3519 V100.The coordinates of the start point of the cropped region must be 2-pixel-aligned, and the width and height must be 4-pixel-aligned. If the coordinate type is VPSS_CROP_RATIO_COOR (relative coordinates), the coordinates of the start point of the cropped region are automatically rounded down to integral multiples of 2, and the width and height are automatically rounded down to integral multiples of 4.

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: `hi_comm_vpss.h`, `mpi_vpss.h`
- Library file: `libmpi.a`

[Note]

- A group must be created before this MPI is called.
- When the coordinate type of the start point is **VPSS_CROP_RATIO_COOR** (relative coordinates), the value ranges of **s32X** and **s32Y** are [0, 999].
- The size of the crop region can neither be less than the minimum size supported by the VPSS nor be greater than the maximum size supported by the VPSS.

[Example]

See the example of [HI_MPI_VPSS_CreateGrp](#).

[See Also]

- [HI_MPI_VPSS_CreateGrp](#)
- [VPSS_CROP_COORDINATE_E](#)
- [VPSS_CROP_INFO_S](#)



HI_MPI_VPSS_GetGrpCrop

[Description]

Obtains the crop attributes of the VPSS.

[Syntax]

```
HI_S32 HI_MPI_VPSS_GetGrpCrop(VPSS_GRP VpssGrp, VPSS_CROP_INFO_S *pstCropInfo)
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
pstCropInfo	Crop function parameter	Output

[Online/Offline Difference]

Mode	Description
Offline mode	This MPI is supported.
Online mode	This MPI is not supported.

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

None

[Example]

See the example of [HI_MPI_VPSS_CreateGrp](#).

[See Also]



[HI_MPI_VPSS_CreateGrp](#)

HI_MPI_VPSS_SendFrame

[Description]

Sends data to a VPSS group.

[Syntax]

```
HI_S32 HI_MPI_VPSS_SendFrame(VPSS\_GRP VpssGrp, VIDEO_FRAME_INFO_S  
*pstVideoFrame, HI_S32 s32MilliSec)
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
pstVideoFrame	Information about the picture to be sent	Input
u32MilliSec	Timeout period. If s32MilliSec is -1 , this MPI is a block MPI; if s32MilliSec is 0 , this MPI is a non-block MPI; If s32MilliSec is greater than 0, the value indicates the timeout period, and its unit is ms.	Input

[Chip Difference]

Chip	Description
Hi3516A	The picture width and height must be 2-pixel-aligned.
Hi3518E V200	The picture width and height must be 4-pixel-aligned.
Hi3519 V100	The picture width and height must be 4-pixel-aligned.

[Online/Offline Difference]

Mode	Description
Offline mode	This MPI is supported.
Online mode	This MPI is not supported.

[Return Value]

Return Value	Description
0	Success



Return Value	Description
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

- A group must be created before this MPI is called.
- You need to control the frame rate when calling this MPI.
- If the VPSS channel is set to **VPSS_CHN_MODE_AUTO** mode, you need to start the VPSS normally and bind it to the back-end receiver before calling this MPI. If the VPSS channel is set to **VPSS_CHN_MODE_USER** mode, you can obtain pictures by calling the related MPI.

[Example]

None

[See Also]

None

HI_MPI_VPSS_SetChnMode

[Description]

Sets the working mode of a VPSS channel. The user mode is used to bind a VPSS channel to multiple output sources. In user mode, the VPSS back-end does not need to be bound to any output source, and pictures are obtained by calling the related MPI. The automatic mode is used in common scenarios for back-end adaptation.

[Syntax]

```
HI_S32 HI_MPI_VPSS_SetChnMode(VPSS_GRP VpssGrp, VPSS_CHN VpssChn,  
VPSS_CHN_MODE_S *pstVpssMode)
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
VpssChn	VPSS channel ID Value range: [0, VPSS_MAX_PHY_CHN_NUM)	Input
pstVpssMode	Channel working mode. For details about limitations, see the description of VPSS_CHN_MODE_S.	Input



[Difference]

Chip	Description
Hi3516A/Hi3518E V200/Hi3519 V100	Only the user mode is supported.

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

- A group must be created before this MPI is called.
- The channel ID must fall within [0, VPSS_MAX_CHN_NUM).
- The automatic mode is used by default. The working mode can be dynamically switched.
- The working mode of extended channels cannot be set.
- For Hi3519 V100, if the width exceeds 4096, compression is not supported.

[Example]

```
VPSS_GRP_ATTR_S stGrpVpssAttr;
VPSS_CHN_ATTR_S stChnAttr;
VIDEO_FRAME_INFO_S stFrame;
HI_S32 s32Ret = HI_SUCCESS;
VPSS_CHN_MODE_S stVpssMode;
HI_U32 u32Depth = 8;
VPSS_GRP VpssGrp;
VPSS_CHN VpssChn
VpssGrp = 0;
VpssChn = 0;

stGrpVpssAttr.u32MaxW = 720;
stGrpVpssAttr.u32MaxH = 576;
stGrpVpssAttr.bIeEn = HI_FALSE;
stGrpVpssAttr.bDciEn = HI_FALSE;
stGrpVpssAttr.bNrEn = HI_FALSE;
```



```
    stGrpVpssAttr.bHistEn = HI_FALSE;
    stGrpVpssAttr.enDieMode = VPSS_DIE_MODE_NODIE;
    stGrpVpssAttr.enPixelFormat = PIXEL_FORMAT_YUV_SEMIPLANAR_422;
    s32Ret = HI_MPI_VPSS_CreateGrp
(VpssGrp, &stGrpVpssAttr);
    if(s32Ret != HI_SUCCESS)
    {
        return s32Ret;
    }

    s32Ret = HI_MPI_VPSS_GetGrpAttr(VpssGrp, &stGrpVpssAttr);
    if(s32Ret != HI_SUCCESS)
    {
        return s32Ret;
    }

    stGrpVpssAttr.bIeEn = HI_FALSE;
    stGrpVpssAttr.bNrEn = HI_TRUE;
    s32Ret = HI_MPI_VPSS_SetGrpAttr(VpssGrp, &stGrpVpssAttr);
    if(s32Ret != HI_SUCCESS)
    {
        return s32Ret;
    }
    s32Ret = HI_MPI_VPSS_GetChnMode(VpssGrp, VpssChn, &stVpssMode);
if(s32Ret != HI_SUCCESS)
{
    return s32Ret;
}
    stVpssMode.enChnMode = VPSS_CHN_MODE_USER;
    stVpssMode.enPixelFormat = PIXEL_FORMAT_YUV_SEMIPLANAR_420;
    stVpssMode.u32Width = 720;
    stVpssMode.u32Height = 576;
    s32Ret = HI_MPI_VPSS_SetChnMode(VpssGrp, VpssChn, &stVpssMode);
if(s32Ret != HI_SUCCESS)
{
    return s32Ret;
}
    HI_MPI_VPSS_SetDepth(VpssGrp, VpssChn, &u32Depth);
    HI_MPI_VPSS_EnableChn(VpssGrp, VpssChn);
    HI_MPI_VPSS_StartGrp(VpssGrp);
    s32Ret = HI_MPI_VPSS_GetChnFrame(VpssGrp, VpssChn, &stFrame);
    if(s32Ret == HI_SUCCESS)
    {
        HI_MPI_VPSS_ReleaseChnFrame(VpssGrp, VpssChn, &stFrame);
    }
```



}

[See Also]

None

HI_MPI_VPSS_GetChnMode

[Description]

Obtains the working mode of a VPSS channel.

[Syntax]

```
HI_S32 HI_MPI_VPSS_GetChnMode (VPSS_GRP VpssGrp, VPSS_CHN VpssChn,  
VPSS_CHN_MODE_S *pstVpssMode)
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
VpssChn	VPSS channel ID Value range: [0, VPSS_MAX_PHY_CHN_NUM)	Input
pstVpssMode	Channel working mode	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

- A group must be created before this MPI is called.
- The working mode of extended channels cannot be set.

[Example]

For details, see [HI_MPI_VPSS_SetChnMode](#).



[See Also]

[HI_MPI_VPSS_SetChnMode](#)

HI_MPI_VPSS_SetDepth

[Description]

Sets the depth of the user picture queue, that is, the length of the picture queue.

[Syntax]

```
HI_S32 HI_MPI_VPSS_SetDepth(VPSS_GRP VpssGrp, VPSS_CHN VpssChn, HI_U32 u32Depth)
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
VpssChn	VPSS channel ID Value range: [0, VPSS_MAX_CHN_NUM)	Input
u32Depth	Queue depth	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

- A group must be created before this MPI is called.
- The default user queue depth is 0, and the maximum user queue depth is 8 (the queue length is 8).
- The user queue buffers the pictures that are not obtained by users and do not contain the pictures that are obtained by users. However, the buffer would not be reclaimed as long as the users do not release the pictures.

[Example]



See the example of [HI_MPI_VPSS_SetChnMode](#).

[See Also]

None

HI_MPI_VPSS_GetDepth

[Description]

Obtains the depth of the user picture queue.

[Syntax]

```
HI_S32 HI_MPI_VPSS_GetDepth(VPSS_GRP VpssGrp, VPSS_CHN VpssChn, HI_U32
*pu32Depth)
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
VpssChn	VPSS channel ID Value range: [0, VPSS_MAX_CHN_NUM)	Input
pu32Depth	Queue depth	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpapi.a

[Note]

A group must be created before this MPI is called.

[Example]

None

[See Also]



None

HI_MPI_VPSS_GetChnFrame

[Description]

Obtains a processed frame from a channel.

[Syntax]

```
HI_S32 HI_MPI_VPSS_GetChnFrame(VPSS_GRP VpssGrp, VPSS_CHN VpssChn,  
VIDEO_FRAME_INFO_S *pstVideoFrame, HI_S32 s32MilliSec)
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
VpssChn	VPSS channel ID Value range: [0, VPSS_MAX_CHN_NUM)	Input
pstVideoFrame	Picture information	Output
s32MilliSec	Timeout period	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

- A group must be created before this MPI is called.
- This MPI applies to all VPSS channels, including physical channels and extended channels.
- Pictures can be obtained only when the channel working mode is user mode and the queue depth is not 0.



- Obtaining pictures by calling this MPI has no effect on the bound back-end modules. For example, if pictures are obtained when the back-end VOU is working, the VOU can normally display pictures.
- If **u32MilliSec** is set to **-1**, the block mode is used. In this case, the program keeps waiting, and a code is returned only after pictures are obtained. If **u32MilliSec** is set to **0**, the non-block mode is used. If **u32MilliSec** is greater than **0**, the timeout waiting mode is used. In this case, **u32MilliSec** indicates the timeout period and its unit is ms. If no picture is obtained within **u32MilliSec**, timeout occurs.
- When a low-delay channel is used, the frame may fail to be obtained.

[Example]

See the example of [HI_MPI_VPSS_SetChnMode](#).

[See Also]

None

HI_MPI_VPSS_ReleaseChnFrame

[Description]

Releases the buffer for storing a frame.

[Syntax]

```
HI_S32 HI_MPI_VPSS_ReleaseChnFrame (VPSS_GRP VpssGrp, VPSS_CHN VpssChn,  
VIDEO_FRAME_INFO_S *pstVideoFrame)
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
VpssChn	VPSS channel ID Value range: [0, VPSS_MAX_CHN_NUM)	Input
pstVideoFrame	Picture information	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]



- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

This MPI must work with [HI_MPI_VPSS_GetChnFrame](#).

[Example]

See the example of [HI_MPI_VPSS_SetChnMode](#).

[See Also]

None

HI_MPI_VPSS_GetGrpFrame

[Description]

Obtains a source frame from a group. The following is the typical application scenario: The same frame needs to be displayed in two channels at the PIP layer and common video layer when the pause and step operations are required during playback on an HD device. The preceding function is supported when [HI_MPI_VPSS_GetGrpFrame](#) works with [HI_MPI_VPSS_SendFrame](#).

[Syntax]

```
HI_S32 HI_MPI_VPSS_GetGrpFrame( VPSS_GRP VpssGrp, VIDEO_FRAME_INFO_S
*pstVideoFrame, HI_U32 u32FrameIndex)
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
pstVideoFrame	Picture information	Output
u32FrameIndex	Picture index ID. Currently, this parameter is invalid, and the default value 0. This reserved parameter will be used for later chips. It is used to specify whether the picture to be obtained is the picture in the current queue or backup queue.	Input

[Online/Offline Difference]

Mode	Description
Offline mode	This MPI is supported.
Online mode	This MPI is not supported.



[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

- Before calling this MPI, ensure that a group is created and enabled and a channel is enabled.
- The source picture of the backup node in a group can be obtained by calling this MPI. The VPSS places the picture of the head node in the cache queue in the backup node in the following situations:
 - If the picture of the head node in the cache queue requires to be processed by VPSS hardware, the VPSS places this picture in the backup node and replaces the source picture.
 - If the bound back-end RX module requires that the VPSS place the picture of the head node in the cache queue in the backup node, the VPSS replaces the picture in the backup node even if this picture is not processed by hardware.
- If the backup node is null, calling this MPI returns the HI_ERR_VPSS_NOTREADY error code.
- The buffer for storing the obtained pictures needs to be released in a timely manner. Otherwise, the VB is insufficient or playback stops. You are advised to call [HI_MPI_VPSS_ReleaseGrpFrame](#) after calling [HI_MPI_VPSS_GetGrpFrame](#).
- The backup node is disabled by default. Therefore, you must call [HI_MPI_VPSS_EnableBackupFrame](#) to enable the backup frame before calling [HI_MPI_VPSS_GetGrpFrame](#). After [HI_MPI_VPSS_GetGrpFrame](#) is used, you can call [HI_MPI_VPSS_DisableBackupFrame](#) to disable the backup frame.
- After [HI_MPI_VPSS_EnableBackupFrame](#) is called to enable backup frames, there is a delay in obtaining pictures because pictures are available in the next frame.

[Example]

For details, see the VDEC samples in the release package.

[See Also]

None

HI_MPI_VPSS_ReleaseGrpFrame

[Description]

Releases the buffer for storing a source frame.

[Syntax]



```
HI_S32 HI_MPI_VPSS_ReleaseGrpFrame (VPSS_GRP VpssGrp, VIDEO_FRAME_INFO_S  
*pstVideoFrame)
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
pstVideoFrame	Picture information	Input

[Online/Offline Difference]

Mode	Description
Offline mode	This MPI is supported.
Online mode	This MPI is not supported.

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

- The **VpssGrp** parameter for this MPI is meaningless. You can set it to any value within the value range.
- This MPI must work with [HI_MPI_VPSS_GetGrpFrame](#).

[Example]

None

[See Also]

None



HI_MPI_VPSS_SetChnParam

[Description]

Sets channel advanced attributes.

[Syntax]

```
HI_S32 HI_MPI_VPSS_SetChnParam(VPSS_GRP VpssGrp, VPSS_CHN VpssChn,  
VPSS_CHN_PARAM_S *pstChnSpParam)
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
VpssChn	VPSS channel ID Value range: [0, VPSS_MAX_PHY_CHN_NUM)	Input
pstChnSpParam	SP advanced attributes	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Difference]

Chip	Description
Hi3516A	This MPI is not supported.
Hi3518E V200	This MPI is not supported.
Hi3519 V100	This MPI is not supported.

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

- A group must be created before this MPI is called.



- For details about the limitations on the **pstChnSpParam** members, see the description of [VPSS_CHN_PARAM_S](#).

[Example]

None

[See Also]

[VPSS_CHN_PARAM_S](#)

HI_MPI_VPSS_GetChnParam

[Description]

Obtains channel advanced attributes.

[Syntax]

```
HI_S32 HI_MPI_VPSS_GetChnParam(VPSS\_GRP VpssGrp, VPSS\_CHN VpssChn,  
VPSS\_CHN\_PARAM\_S *pstChnSpParam)
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
VpssChn	VPSS channel ID Value range: [0, VPSS_MAX_PHY_CHN_NUM)	Input
pstChnSpParam	SP advanced attributes	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Difference]

Chip	Description
Hi3516A	This MPI is not supported.
Hi3518E V200	This MPI is not supported.
Hi3519 V100	This MPI is not supported.



[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

- A group must be created before this MPI is called.
- For details about the limitations on the **pstChnSpParam** members, see the description of [VPSS_CHN_PARAM_S](#).

[Example]

None

[See Also]

[VPSS_CHN_PARAM_S](#)

HI_MPI_VPSS_SetGrpFrameRate

[Description]

Sets the frame rate control information for the VPSS.

[Syntax]

```
HI_S32 HI_MPI_VPSS_SetGrpFrameRate (VPSS_GRP VpssGrp, VPSS_FRAME_RATE_S  
*pstVpssFrameRate)
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
pstVpssFrameRate	Frame rate control information	Input

[Online/Offline Difference]

Mode	Description
Offline mode	This MPI is supported.
Online mode	This MPI is not supported.

[Return Value]



Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

- A group must be created before this MPI is called.
- Frame rate is controlled at the very beginning, which affects subsequent processing.

[Example]

None

[See Also]

[VPSS_FRAME_RATE_S](#)

HI_MPI_VPSS_GetGrpFrameRate

[Description]

Obtains the frame rate control information for the VPSS.

[Syntax]

```
HI_S32 HI_MPI_VPSS_GetGrpFrameRate( VPSS_GRP VpssGrp, VPSS_FRAME_RATE_S  
*pstVpssFrameRate)
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
pstVpssFrameRate	Frame rate control information	Output

[Online/Offline Difference]

Mode	Description
Offline mode	This MPI is supported.
Online mode	This MPI is not supported.



[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

A group must be created before this MPI is called.

[Example]

None

[See Also]

[VPSS_FRAME_RATE_S](#)

HI_MPI_VPSS_SetGrpDelay

[Description]

Sets the length of the delay queue for delaying the start of VPSS processing. A delay may occur when the pictures from the front-end are transferred to the master chip over the PCI. To implement synchronization, the VPSS delays the master chip pictures by using the delay queue. When you obtain a picture from the VI channels and draw borders by using the TDE, the picture is transmitted to the VPSS at the same time. Because the processing speed of the VPSS is faster than that of the TDE, the VPSS needs to be delayed.

[Syntax]

```
HI_S32 HI_MPI_VPSS_SetGrpDelay (VPSS_GRP VpssGrp, HI_U32 u32Delay)
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
u32Delay	Delay queue length Value range: [0, 5]	Input



[Online/Offline Difference]

Mode	Description
Offline mode	This MPI is supported.
Online mode	This MPI is not supported.

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

A group must be created before this MPI is called.

[Example]

None

[See Also]

[HI_MPI_VPSS_GetGrpDelay](#)

HI_MPI_VPSS_GetGrpDelay

[Description]

Obtain the length of the delay queue for delaying the start of VPSS processing.

[Syntax]

```
HI_S32 HI_MPI_VPSS_GetGrpDelay (VPSS_GRP VpssGrp, HI_U32 *pu32Delay)
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
pu32Delay	Pointer to the obtained delay queue	Output



[Online/Offline Difference]

Mode	Description
Offline mode	This MPI is supported.
Online mode	This MPI is not supported.

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

A group must be created before this MPI is called.

[Example]

None

[See Also]

[HI_MPI_VPSS_SetGrpDelay](#)

HI_MPI_VPSS_SetExtChnAttr

[Description]

Sets the attributes of an extended VPSS channel. The extended channel is used to implement second scaling and control the frame rate.

[Syntax]

```
HI_S32 HI_MPI_VPSS_SetExtChnAttr(VPSS\_GRP VpssGrp, VPSS\_CHN VpssChn,  
VPSS\_EXT\_CHN\_ATTR\_S *pstExtChnAttr
```

[Parameter]



Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
VpssChn	VPSS extended channel ID Value range: [VPSS_MAX_PHY_CHN_NUM, VPSS_MAX_CHN_NUM)	Input
pExtChnAttr	Attributes of an extended VPSS channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

- The input sources to be bound to extended channels must be physical channels. The channel width and height must be 2-pixel-aligned. **SrcFrameRate** must be greater than or equal to **DstFrameRate**. If both **SrcFrameRate** and **DstFrameRate** are set to **-1**, the frame rate is not controlled.
- The extended channels connect to physical channels after they are bound. Each extended channel can be bound to only one physical channel.
- Multiple channels can be bound to the same physical channel. If the maximum number of extended channels to be bound is reached, the attributes fail to be set.
- A group must be created before this MPI is called.
- For Hi3519 V100, if the width exceeds 4096, compression is not supported.

[Example]

None

[See Also]

[VPSS_EXT_CHN_ATTR_S](#)

HI_MPI_VPSS_GetExtChnAttr

[Description]



Obtains the attributes of an extended VPSS channel.

[Syntax]

```
HI_S32 HI_MPI_VPSS_GetExtChnAttr(VPSS\_GRP VpssGrp, VPSS\_CHN VpssChn,  
VPSS\_EXT\_CHN\_ATTR\_S *pstExtChnAttr)
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
VpssChn	VPSS extended channel ID Value range: [VPSS_MAX_PHY_CHN_NUM , VPSS_MAX_CHN_NUM)	Input
pExtChnAttr	Attributes of an extended VPSS channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: `hi_comm_vpss.h`, `mpi_vpss.h`
- Library file: `libmpi.a`

[Note]

A group must be created before this MPI is called.

[Example]

None

[See Also]

[VPSS_EXT_CHN_ATTR_S](#)

HI_MPI_VPSS_SetChnOverlay

[Description]

Enables or disables the video overlay region of a VPSS channel. The REGION module centrally manages all video overlay regions. You can set overlay attributes by using the REGION module and then enable or disable an overlay region by calling this MPI.



[Syntax]

```
HI_S32 HI_MPI_VPSS_SetChnOverlay(VPSS_GRP VpssGrp, VPSS_CHN VpssChn,  
HI_U32 u32OverlayMask)
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
VpssChn	VPSS extended channel ID Value range: [0, VPSS_MAX_CHN_NUM)	Input
u32OverlayMask	Switch of the VPSS channel video overlay region The lower eight bits are valid, and each bit is used to enable or disable a region. The region with a lower priority is overlaid with the region with a higher priority by default.	Input

[Difference]

Chip	Description
Hi3516A/Hi3518E V200/Hi3519 V100	This MPI is not supported.

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

- A group must be created before this MPI is called.
- Bit 0 of **u32OverlayMask** corresponds to the overlay region with priority 0, bit 1 of **u32OverlayMask** corresponds to the overlay region with priority 1, and so on. If you



want to overlay multiple overlay regions by using the VPSS, the priorities of overlay regions must be unique.

- The enable status of the video overlay region cannot be configured for an extended channel. Calling this MPI has no effect for an extended channel, but no error is reported.

[Example]

None

[See Also]

[HI_MPI_VPSS_GetChnOverlay](#)

HI_MPI_VPSS_GetChnOverlay

[Description]

Obtains the enable status of the video overlay region of a VPSS channel.

[Syntax]

```
HI_S32 HI_MPI_VPSS_GetChnOverlay(VPSS_GRP VpssGrp, VPSS_CHN VpssChn,  
HI_U32 *pu32OverlayMask)
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
VpssChn	ID of an extended VPSS channel Value range: [0, VPSS_MAX_CHN_NUM)	Input
u32OverlayMask	Switch of the VPSS channel video overlay region The parameter value is an 8-bit binary number, and each bit is used to enable or disable a region.	Output

[Difference]

Chip	Description
Hi3516A/Hi3518E V200/Hi3519 V100	This MPI is not supported.

[Return Value]

Return Value	Description
0	Success



Return Value	Description
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

A group must be created before this MPI is called.

[Example]

None

[See Also]

[HI_MPI_VPSS_SetChnOverlay](#)

HI_MPI_VPSS_SetChnCover

[Description]

Enables or disables the video cover region a VPSS channel. The REGION module manages all video cover regions. You can set cover attributes by using the REGION module and then enable or disable a cover region by calling this MPI.

[Syntax]

```
HI_S32 HI_MPI_VPSS_SetChnCover( VPSS_GRP VpssGrp, VPSS_CHN VpssChn, HI_U32  
u32CoverMask );
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
VpssChn	VPSS channel ID Value range: [0, VPSS_MAX_PHY_CHN_NUM - 1)	Input
u32CoverMask	Switch of the VPSS channel video cover region The lower eight bits are valid, and each bit is used to enable or disable a region. The region with a higher priority covers the region with a lower priority by default.	Input



[Difference]

Chip	Description
Hi3516A	This MPI is supported.
Hi3518E V200	This MPI is not supported.
Hi3519 V100	This MPI is not supported.

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

- A group must be created before this MPI is called.
- Bit 0 of **u32CoverMask** corresponds to the cover region with priority 0, and bit 1 corresponds to the cover region with priority 1. If the cover regions have the same priority, their priorities are configured in the sequence that they are submitted. For example, if eight cover regions have priority 0, the bits of **u32CoverMask** enable and disable the cover regions in the sequence that they are submitted.
- The VPSS supports **COVER_MAX_NUM_VPSS** video Cover regions.

[Example]

None

[See Also]

[HI_MPI_VPSS_GetChnCover](#)

HI_MPI_VPSS_GetChnCover

[Description]

Obtains the enable status of the video cover region of a VPSS channel.

[Syntax]

```
HI_S32 HI_MPI_VPSS_GetChnCover(VPSS_GRP VpssGrp, VPSS_CHN VpssChn, HI_U32  
*pu32CoverMask);
```

[Parameter]



Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
VpssChn	VPSS channel ID Value range: [0, VPSS_MAX_PHY_CHN_NUM)	Input
pu32CoverMask	Switch of the VPSS channel video cover region The parameter value is an 8-bit binary number, and each bit is used to enable or disable a region.	Output

[Difference]

Chip	Description
Hi3516A	This MPI is supported.
Hi3518E V200	This MPI is not supported.
Hi3519 V100	This MPI is not supported.

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

A group must be created before this MPI is called.

[Example]

None

[See Also]

[HI_MPI_VPSS_SetChnCover](#)



HI_MPI_VPSS_GetRegionLuma

[Description]

Obtains the total luminance of the regions in a specified picture. This MPI is used to collect statistics on the luminance of cover regions. Then colors are inverted based on the luminance statistics, making the cover regions more obvious.

[Syntax]

```
HI_S32 HI_MPI_VPSS_GetRegionLuma(VPSS_GRP VpssGrp, VPSS_CHN VpssChn,  
VPSS_REGION_INFO_S *pstRegionInfo, HI_U32 *pu32LumaData, HI_S32  
s32MilliSec)
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
VpssChn	VPSS channel ID Value range: [0, VPSS_MAX_PHY_CHN_NUM)	Input
pstRegionInfo	Region information. pstRegionInfo->pstRegion indicates the attributes of statistics regions including the start position, width, and height. pstRegionInfo->u32RegionNum indicates the number of statistics regions.	Input
pu32LumaData	Pointer to memory for receiving region luminance statistics information. The memory size must be greater than or equal to sizeof(HI_U32) multiplied by pstRegionInfo->u32RegionNum .	Output
s32MilliSec	Timeout parameter. If s32MilliSec is -1 or 0 , this MPI is a block MPI. If s32MilliSec is greater than 0 , it indicates the timeout period, and its unit is ms.	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h



- Library file: libmpi.a

[Note]

- A group must be created before this MPI is called.
- A channel must be enabled before this MPI is called.
- The luminance statistics obtained by calling this MPI are the luminance after VPSS processing.
- A maximum of 64 statistics regions is supported.
- The start coordinates of the statistics regions are determined as follows:
 - If the crop function is disabled, the start coordinates are determined based on the coordinates of the upper left corner of the original picture.
 - If the crop function is enabled, the start coordinates are determined based on the coordinates of the upper left corner of the cropped picture.
- If the corresponding channel is an online low-delay channel, a code indicating "not supported" is directly returned by this MPI.

[Example]

None

[See Also]

[VPSS_REGION_INFO_S](#)

HI_MPI_VPSS_SetChnCrop

[Description]

Sets the cropping attributes of a VPSS channel.

[Syntax]

```
HI_S32 HI_MPI_VPSS_SetChnCrop(VPSS_GRP VpssGrp, VPSS_CHN VpssChn,  
VPSS_CROP_INFO_S *pstCropInfo);
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
VpssChn	VPSS channel ID Value range: (0, VPSS_MAX_PHY_CHN_NUM)	Input
pstCropInfo	Crop function parameter	Input

[Return Value]



Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

A group must be created before this MPI is called.

[Example]

None

[See Also]

[HI_MPI_VPSS_GetChnCrop](#)

HI_MPI_VPSS_GetChnCrop

[Description]

Obtains the cropping attributes of a VPSS channel.

[Syntax]

```
HI_S32 HI_MPI_VPSS_GetChnCrop(VPSS_GRP VpssGrp, VPSS_CHN VpssChn,  
VPSS_CROP_INFO_S *pstCropInfo);
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
VpssChn	VPSS channel ID Value range: (0, VPSS_MAX_PHY_CHN_NUM)	Input
pstCropInfo	Crop function parameter	Output

[Return Value]

Return Value	Description
0	Success



Return Value	Description
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

A group must be created before this MPI is called.

[Example]

None

[See Also]

[HI_MPI_VPSS_SetChnCrop](#)

HI_MPI_VPSS_SetLDCAttr

[Description]

Sets the LDC attributes of a VPSS channel.

[Syntax]

```
HI_S32 HI_MPI_VPSS_SetLDCAttr(VPSS_GRP VpssGrp, VPSS_CHN VpssChn, const VPSS_LDC_ATTR_S *pstLDCAttr);
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
VpssChn	VPSS channel ID Value range: [0, VPSS_MAX_PHY_CHN_NUM)	Input
pstLDCAttr	LDC attributes	Input

[Difference]

Chip	Mode	Description
Hi3516A	Offline	The LDC is supported. The LDC module supports only uncompressed semi-planar 420 and semi-planar 422 pictures.



Chip	Mode	Description
	Online	The LDC is supported. The LDC module supports only uncompressed semi-planar 420 and semi-planar 422 pictures.
Hi3518E V200	Offline	This MPI is supported.
	Online	This MPI is supported.
Hi3519 V100	Offline	This MPI is not supported.
	Online	This MPI is not supported.

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

- A group must be created before this MPI is called.
- LDC attributes are dynamic attributes.
- LDC is not supported for compressed pictures.
- LDC is performed after the VPSS processes pictures; however, the LDC effect is affected.
- The maximum LDC performance for the Hi3516A is 1080p@30 fps.

[Example]

None

[See Also]

- [VPSS_LDC_ATTR_S](#)
- [HI_MPI_VPSS_GetLDCAttr](#)

HI_MPI_VPSS_GetLDCAttr

[Description]

Obtains the LDC attributes of a VPSS channel.

[Syntax]



```
HI_S32 HI_MPI_VPSS_GetLDCAttr(VPSS_GRP VpssGrp, VPSS_CHN VpssChn,  
VPSS_LDC_ATTR_S *pstLDCAttr);
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
VpssChn	VPSS channel ID Value range: [0, VPSS_MAX_PHY_CHN_NUM)	Input
pstLDCAttr	LDC attributes	Output

[Difference]

Chip	Mode	Description
Hi3516A	Offline	This MPI is supported.
	Online	This MPI is supported.
Hi3518E V200	Offline	This MPI is supported.
	Online	This MPI is supported.
Hi3519 V100	Offline	This MPI is not supported.
	Online	This MPI is not supported.

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

A group must be created before this MPI is called.

[Example]



None

[See Also]

[HI_MPI_VPSS_SetLDCAttr](#)

HI_MPI_VPSS_SetRotate

[Description]

Sets picture rotation attributes of a VPSS channel.

[Syntax]

```
HI_S32 HI_MPI_VPSS_SetRotate(VPSS_GRP VpssGrp, VPSS_CHN VpssChn, ROTATE_E  
enRotate);
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
VpssChn	VPSS channel ID Value range: [0, VPSS_MAX_PHY_CHN_NUM)	Input
enRotate	Rotation attributes	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpia.a

[Note]

- A group must be created before this MPI is called.
- The compressed or uncompressed input and the uncompressed output are supported. The Hi3516A supports the semi-planar 420 pixel format, whereas Hi3518E V200 and Hi3519 V100 support the semi-planar 420 and YUV400 pixel formats.
- Pictures can be rotated only by 90°, 180°, or 270°.
- The picture rotation does not take effect when a low-delay channel is used.



- When rotation is enabled in channel 0 and **VPSS_REF_FROM_CHN0** is used as the source of the 3DNR reference frame, 3DNR takes no effect.
- For Hi3519 V100, if the channel width exceeds 4096, rotation is not supported.

[Example]

None

[See Also]

[HI_MPI_VPSS_GetChnCover](#)

HI_MPI_VPSS_GetRotate

[Description]

Obtains picture rotation attributes of a VPSS channel.

[Syntax]

```
HI_S32 HI_MPI_VPSS_GetRotate(VPSS_GRP VpssGrp, VPSS_CHN VpssChn, ROTATE_E  
*penRotate);
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
VpssChn	VPSS channel ID Value range: [0, VPSS_MAX_PHY_CHN_NUM)	Input
penRotate	Rotation attributes	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

A group must be created before this MPI is called.



[Example]

None

[See Also]

[HI_MPI_VPSS_SetChnCover](#)

HI_MPI_VPSS_SetChnNR

[Description]

Enables or disables NR of a VPSS channel.

[Syntax]

```
HI_S32 HI_MPI_VPSS_SetChnNR(VPSS_GRP VpssGrp, VPSS_CHN VpssChn, HI_BOOL bEnable);
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
VpssChn	VPSS channel ID Value: VPSS_CHN3	Input
bEnable	NR enable	Input

[Difference]

Chip	Description
Hi3516A	This MPI is supported.
Hi3518E V200	This MPI is not supported.
Hi3519 V100	This MPI is not supported.

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]



- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

- A group must be created before this MPI is called.
- If NR is disabled in the group attributes, the NR function cannot be enabled by calling this MPI.
- If NR is disabled in channel 3 of the Hi3516A, the pictures of channel 3 are not processed by the NR module.

[Example]

None

[See Also]

[HI_MPI_VPSS_GetChnNR](#)

HI_MPI_VPSS_GetChnNR

[Description]

Obtains the NR enable status of a VPSS channel.

[Syntax]

```
HI_S32 HI_MPI_VPSS_GetChnNR(VPSS_GRP VpssGrp, VPSS_CHN VpssChn, HI_BOOL *pbEnable);
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
VpssChn	VPSS channel ID Value: VPSS_CHN3	Input
pbEnable	NR enable status	Output

[Difference]

Chip	Description
Hi3516A	This MPI is supported.
Hi3518E V200	This MPI is not supported.
Hi3519 V100	This MPI is not supported.



[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

A group must be created before this MPI is called.

[Example]

None

[See Also]

[HI_MPI_VPSS_SetChnNR](#)

HI_MPI_VPSS_SetLowDelayAttr

[Description]

Sets short delay attributes of a VPSS channel.

[Syntax]

```
HI_S32 HI_MPI_VPSS_SetLowDelayAttr(VPSS\_GRP VpssGrp, VPSS\_CHN VpssChn,  
VPSS\_LOW\_DELAY\_INFO\_S *pstLowDelayInfo);
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
VpssChn	VPSS channel ID Value range: [0, 3)	Input
pstLowDelayInfo	Short delay attributes	Input

[Online/Offline Difference]



Chip	Description
Offline mode	This MPI is not supported.
Online mode	This MPI is supported.

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

- A group must be created before this MPI is called.
- **ChnMode** must be configured before this MPI is called.
- Channel 3 does not support the short delay configuration.
- Only one short delay channel can be enabled at a time. When the short delay of a channel is enabled, you can enable the short delay of another channel only after the short delay of the current channel is disabled.
- Short delay attributes are dynamic attributes.
- When the short delay of a channel is enabled, its LDC, rotation, CoverEx, OverlayEx, fisheye, and extended channels are invalid.
- The size of the required VB when the channel low-latency function is enabled is 48 bytes greater than that when the channel low-latency function is disabled (other attributes are the same).
- When the low delay function of the channel is enabled, the channel does not support flip.
- The frames in the low-latency channel can only be encoded. For details, see the **Note** field of `HI_MPI_VENC_CreateChn` in chapter 6 "VENC" or "VENC2" of the *HiMPP IPC V2.0 Media Processing Software Development Reference*.

[Example]

None

[See Also]

- [VPSS_LOW_DELAY_INFO_S](#)
- [HI_MPI_VPSS_GetLowDelayAttr](#)

HI_MPI_VPSS_GetLowDelayAttr

[Description]



Obtains short delay attributes of a VPSS channel.

[Syntax]

```
HI_S32 HI_MPI_VPSS_GetLowDelayAttr(VPSS_GRP VpssGrp, VPSS_CHN VpssChn,  
VPSS_LOW_DELAY_INFO_S *pstLowDelayInfo);
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
VpssChn	VPSS channel ID Value range: [0, 3)	Input
pstLowDelayInfo	Short delay attributes	Output

[Online/Offline Difference]

Mode	Description
Offline mode	This MPI is not supported.
Online mode	This MPI is supported.

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

A group must be created before this MPI is called.

[Example]

None

[See Also]



[HI_MPI_VPSS_SetLowDelayAttr](#)

HI_MPI_VPSS_SetRefSelect

[Description]

Sets the NR reference frame source.

[Syntax]

```
HI_S32 HI_MPI_VPSS_SetRefSelect(VPSS\_GRP VpssGrp, const  
VPSS\_REF\_SEL\_MODE\_E enRefSelMode);
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
enRefSelMode	NR reference frame source	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

A group must be created before this MPI is called.

[Example]

None

[See Also]

[HI_MPI_VPSS_GetRefSelect](#)

HI_MPI_VPSS_GetRefSelect

[Description]

Obtains the NR reference frame source.



[Syntax]

```
HI_S32 HI_MPI_VPSS_SetRefSelect( VPSS_GRP VpssGrp, VPSS_REF_SEL_MODE_E  
*enRefSelMode );
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
enRefSelMode	NR reference frame source	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

A group must be created before this MPI is called.

[Example]

None

[See Also]

[HI_MPI_VPSS_SetRefSelect](#)

HI_MPI_VPSS_SetExtChnCrop

[Description]

Sets the cropping attributes of an extended VPSS channel.

[Syntax]

```
HI_S32 HI_MPI_VPSS_SetExtChnCrop( VPSS_GRP VpssGrp, VPSS_CHN VpssChn,  
const VPSS_CROP_INFO_S *pstCropInfo );
```

[Parameter]



Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
VpssChn	VPSS channel ID Value range: [VPSS_MAX_PHY_CHN_NUM, VPSS_MAX_CHN_NUM)	Input
pstCropInfo	Crop function parameter	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

- A group must be created before this MPI is called.
- In an extended VPSS channel, a picture must be cropped before it is scaled.

[Example]

None

[See Also]

[HI_MPI_VPSS_GetChnCrop](#)

HI_MPI_VPSS_GetExtChnCrop

[Description]

Obtains the cropping attributes of an extended VPSS channel.

[Syntax]

```
HI_S32 HI_MPI_VPSS_GetExtChnCrop(VPSS\_GRP VpssGrp, VPSS\_CHN VpssChn,  
VPSS\_CROP\_INFO\_S *pstCropInfo);
```

[Parameter]



Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
VpssChn	VPSS channel ID Value range: [VPSS_MAX_PHY_CHN_NUM, VPSS_MAX_CHN_NUM)	Input
pstCropInfo	Crop function parameter	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

A group must be created before this MPI is called.

[Example]

None

[See Also]

[HI_MPI_VPSS_SetChnCrop](#)

HI_MPI_VPSS_SetGrpParamV2

[Description]

Sets the VPSS 3DNR parameters (version 2).

[Syntax]

```
HI_S32 HI_MPI_VPSS_SetGrpParamV2(VPSS_GRP_VpssGrp, VPSS_GRP_PARAM_V2_S  
*pstVpssParamV2);
```

[Parameter]



Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
pstVpssParamV2	Pointer to the VPSS 3DNR parameters (version 2)	Input

[Difference]

Chip	Description
Hi3516A	This MPI is supported and recommended.
Hi3518E V200	This MPI is not supported.
Hi3519 V100	This MPI is not supported.

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

A group must be created before this MPI is called.

[Example]

None

[See Also]

[HI_MPI_VPSS_GetGrpParamV2](#)

HI_MPI_VPSS_GetGrpParamV2

[Description]

Obtains the VPSS 3DNR parameters (version 2).

[Syntax]



```
HI_S32 HI_MPI_VPSS_GetGrpParamV2(VPSS_GRP_VpssGrp, VPSS_GRP_PARAM_V2_S
*pstVpssParamV2);
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
pstVpssParamV2	Pointer to the VPSS 3DNR parameters (version 2)	Output

[Difference]

Chip	Description
Hi3516A	This MPI is supported and recommended.
Hi3518E V200	This MPI is not supported.
Hi3519 V100	This MPI is not supported.

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

A group must be created before this MPI is called.

[Example]

None

[See Also]

[HI_MPI_VPSS_SetGrpParamV2](#)

HI_MPI_VPSS_SetNRParam

[Description]



Sets the VPSS 3DNR parameters.

[Syntax]

```
HI_S32 HI_MPI_VPSS_SetNRParam(VPSS_GRP VpssGrp, VPSS_NR_PARAM_U  
*punNrParam);
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
punNrParam	Pointer to the VPSS 3DNR parameter union	Input

[Difference]

Chip	Description
Hi3516A	This MPI is not supported.
Hi3518E V200	This MPI is supported.
Hi3519 V100	This MPI is not supported currently, and will be supported later.

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

A group must be created before this MPI is called.

[Example]

None

[See Also]

[HI_MPI_VPSS_GetNRParam](#)



HI_MPI_VPSS_GetNRParam

[Description]

Obtains the VPSS 3DNR parameters.

[Syntax]

```
HI_S32 HI_MPI_VPSS_GetNRParam(VPSS_GRP VpssGrp, VPSS_NR_PARAM_U  
*punNrParam);
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
punNrParam	Pointer to the VPSS 3DNR parameter union	Output

[Difference]

Chip	Description
Hi3516A	This MPI is not supported.
Hi3518E V200	This MPI is supported.
Hi3519 V100	This MPI is not supported currently, and will be supported later.

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

A group must be created before this MPI is called.

[Example]

None



[See Also]

[HI_MPI_VPSS_SetNRParam](#)

HI_MPI_VPSS_SetFisheyeConfig

[Description]

Sets the LMF parameters of the fisheye lens for the VPSS.

[Syntax]

```
HI_S32 HI_MPI_VPSS_SetFisheyeConfig(VPSS_GRP VpssGrp, const  
FISHEYE_CONFIG_S *pstFisheyeConfig);
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
pstFisheyeConfig	Pointer to the LMF parameters of the fisheye lens	Input

[Difference]

Chip	Description
Hi3516A	This MPI is not supported.
Hi3518E V200	This MPI is not supported.
Hi3519 V100	This MPI is supported.

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

A group must be created before this MPI is called.



[Example]

None

[See Also]

None

HI_MPI_VPSS_GetFisheyeConfig

[Description]

Obtains the LMF parameters of the fisheye lens for the VPSS.

[Syntax]

```
HI_S32 HI_MPI_VPSS_GetFisheyeConfig(VPSS_GRP VpssGrp, FISHEYE_CONFIG_S *pstFisheyeConfig);
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
pstFisheyeConfig	Pointer to the LMF parameters of the fisheye lens	Output

[Difference]

Chip	Description
Hi3516A	This MPI is not supported.
Hi3518E V200	This MPI is not supported.
Hi3519 V100	This MPI is supported.

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a



[Note]

- A group must be created before this MPI is called.
- The parameters can be obtained only after they are configured. If this MPI is called to obtain the parameters before they are configured, an error code indicating failure is returned.

[Example]

None

[See Also]

None

HI_MPI_VPSS_SetFisheyeAttr

[Description]

Sets the fisheye attribute of an extended VPSS channel.

[Syntax]

```
HI_S32 HI_MPI_VPSS_SetFisheyeAttr(VPSS_GRP VpssGrp, VPSS_CHN VpssChn,  
const FISHEYE_ATTR_S *pstFisheyeAttr);
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
VpssChn	Extended channel where the fisheye is located Value range: [VPSS_MAX_PHY_CHN_NUM, VPSS_MAX_CHN_NUM)	Input
pstFisheyeAttr	Pointer to the fisheye attribute	Input

[Difference]

Chip	Description
Hi3516A	This MPI is not supported.
Hi3518E V200	This MPI is not supported.
Hi3519 V100	This MPI is supported.

[Return Value]



Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

- A group must be created before this MPI is called.
- The channel must be an extended channel, and the attributes of the extended channel must be configured before this MPI is called.
- For details about the restrictions on the fisheye, see chapter 11 "Fisheye Subsystem" in the *HiMPP IPC V2.0 Media Processing Software Development Reference*.

[Example]

None

[See Also]

None

HI_MPI_VPSS_GetFisheyeAttr

[Description]

Obtains the fisheye attribute of an extended VPSS channel.

[Syntax]

```
HI_S32 HI_MPI_VPSS_GetFisheyeAttr(VPSS_GRP VpssGrp, VPSS_CHN VpssChn,  
FISHEYE_ATTR_S *pstFisheyeAttr);
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range in offline mode: [0, VPSS_MAX_GRP_NUM) Value in online mode: 0	Input
VpssChn	Extended channel where the fisheye is located Value range: [VPSS_MAX_PHY_CHN_NUM, VPSS_MAX_CHN_NUM)	Input
pstFisheyeAttr	Pointer to the fisheye attribute	Output



[Difference]

Chip	Description
Hi3516A	This MPI is not supported.
Hi3518E V200	This MPI is not supported.
Hi3519 V100	This MPI is supported.

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

- A group must be created before this MPI is called.
- The attributes can be obtained only after they are configured. If this MPI is called to obtain the attributes before they are configured, an error code indicating failure is returned.

[Example]

None

[See Also]

None

HI_MPI_VPSS_SetModParam

[Description]

Sets the VPSS module parameters.

[Syntax]

```
HI_S32 HI_MPI_VPSS_SetModParam(VPSS_MOD_PARAM_S *pstModParam);
```

[Parameter]

Parameter	Description	Input/Output
pstModParam	Pointer to the VPSS module parameter structure	Input



[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

The VPSS module must be loaded and no group is created before this MPI is called.

[Example]

None

[See Also]

None

HI_MPI_VPSS_GetModParam

[Description]

Obtains the VPSS module parameters.

[Syntax]

```
HI_S32 HI_MPI_VPSS_GetModParam(VPSS\_MOD\_PARAM\_S *pstModParam);
```

[Requirement]

Parameter	Description	Input/Output
pstModParam	Pointer to the VPSS module parameter structure	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."



[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

The VPSS module must be loaded before this MPI is called.

[Example]

None

[See Also]

None

HI_MPI_VPSS_SetNRBParam

[Description]

Sets the attributes of the VPSS 3DNR parameters. This MPI is used to set the 3DNR parameters, such as the 3DNR enable, side effect correction for the spatial filtering, temporal filtering/spatial filtering strength for the color difference signal, filtering strength for NR in the bright/dark region, absolute strength/relative strength of the temporal filtering, and motion threshold.

[Syntax]

```
HI_S32 HI_MPI_VPSS_SetNRBParam(VPSS_GRP VpssGrp, VPSS_GRP_VPPNRB_S  
*pstVpssNrBParam)
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range: [0, VPSS_MAX_GRP_NUM)	Input
pstVpssNrBParam	Configuration of the 3DNR parameter attributes	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Difference]



Chip	Description
Hi3516A	This MPI is not supported.
Hi3518E V200	This MPI is not supported.
Hi3519 V100	This MPI is supported.

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

A group must be created before this MPI is called.

[Example]

None

[See Also]

None

HI_MPI_VPSS_GetNRBParam

[Description]

Obtains the attributes of the VPSS 3DNR parameters.

[Syntax]

```
HI_S32 HI_MPI_VPSS_GetNRBParam(VPSS_GRP VpssGrp, VPSS_GRP_VPPNRB_S  
*pstVpssNrBParam)
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range: [0, VPSS_MAX_GRP_NUM)	Input
pstVpssNrBParam	Configuration of the 3DNR parameter attributes	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."



[Difference]

Chip	Description
Hi3516A	This MPI is not supported.
Hi3518E V200	This MPI is not supported.
Hi3519 V100	This MPI is supported.

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

A group must be created before this MPI is called.

[Example]

None

[See Also]

None

HI_MPI_VPSS_SetNRV3Param

[Description]

Sets the VPSS 3DNR parameters (V3 version).

[Syntax]

```
HI_S32 HI_MPI_VPSS_SetNRV3Param(VPSS_GRP VpssGrp, VPSS_GRP_VPPNRBEX_S  
*pstVpssNrParam)
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range: [0, VPSS_MAX_GRP_NUM)	Input
pstVpssNrParam	Configuration of the 3DNR parameter attributes	Input

[Return Value]

Return Value	Description
0	Success



Return Value	Description
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Difference]

Chip	Description
Hi3516A	This MPI is supported.
Hi3518E V200	This MPI is not supported.
Hi3519 V100	This MPI is not supported.

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

A group must be created before this MPI is called.

[Example]

None

[See Also]

None

HI_MPI_VPSS_GetNRV3Param

[Description]

Obtains the VPSS 3DNR parameters (V3 version).

[Syntax]

```
HI_S32 HI_MPI_VPSS_GetNRV3Param(VPSS_GRP VpssGrp, VPSS_GRP_VPPNRBEX_S
*pstVpssNrParam)
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range: [0, VPSS_MAX_GRP_NUM)	Input
pstVpssNrParam	Configuration of the 3DNR parameter attributes	Output



[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Difference]

Chip	Description
Hi3516A	This MPI is supported.
Hi3518E V200	This MPI is not supported.
Hi3519 V100	This MPI is not supported.

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

A group must be created before this MPI is called.

[Example]

None

[See Also]

None

HI_MPI_VPSS_SetGrpNRSPParam

[Description]

Sets the effect parameters of the standard 3DNR interface of the VPSS group. It is recommended that Hi3518E V200 and later chips use this MPI to set the effect parameters of the standard interface.

[Syntax]

```
HI_S32 HI_MPI_VPSS_SetGrpNRSPParam(VPSS_GRP VpssGrp, VPSS_GRP_NRS_PARAM_S *pstNRSPParam);
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range: [0, VPSS_MAX_GRP_NUM)	Input



Parameter	Description	Input/Output
pstNRSPParam	Effect parameters of the standard 3DNR interface	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Difference]

Chip	Description
Hi3516A	This MPI is not supported.
Hi3518E V200	This MPI is supported.
Hi3519 V100	This MPI is not supported.

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

- A group must be created before this MPI is called.
- The enumerated member **enNRVer** in **pstNRSPParam** indicates the version of the standard interface. This member is designed for the sake of extensibility, and its initial value is **VPSS_NR_V1**. The corresponding parameter structures of different standard interface versions are stored in the union of **pstNRSPParam**. Different chips may support different interface versions or the same interface version. One chip may support multiple interface versions. For details, see [VPSS_GRP_NRS_PARAM_S](#).
- **HI_MPI_VPSS_SetGrpNRSPParam** is supported by multiple chip versions. For Hi3518E V200, **HI_MPI_VPSS_SetNRParam** is implemented by the VPSS_NR_V1 version of **HI_MPI_VPSS_SetGrpNRSPParam**. You are advised to replace **HI_MPI_VPSS_SetNRParam** and **HI_MPI_VPSS_GetNRParam** with **HI_MPI_VPSS_SetGrpNRSPParam** and **HI_MPI_VPSS_GetGrpNRSPParam**, respectively.

[Example]

None

[See Also]

None



HI_MPI_VPSS_GetGrpNRSPParam

[Description]

Obtains the effect parameters of the standard 3DNR interface of the VPSS group. It is recommended that Hi3518E V200 and later chips use this MPI to obtain the effect parameters of the standard interface.

[Syntax]

```
HI_S32 HI_MPI_VPSS_GetGrpNRSPParam(VPSS_GRP VpssGrp, VPSS_GRP_NRS_PARAM_S  
*pstNRSPParam);
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range: [0,VPSS_MAX_GRP_NUM)	Input
pstNRSPParam	Effect parameters of the standard 3DNR interface	Input/Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Difference]

Chip	Description
Hi3516A	This MPI is not supported.
Hi3518E V200	This MPI is supported.
Hi3519 V100	This MPI is not supported.

[Requirement]

- Header files:hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

A group must be created before this MPI is called.

[Example]

None



[See Also]

None

HI_MPI_VPSS_SetGrpNRBParam

[Description]

Sets the effect parameters of the advanced 3DNR interface of the VPSS group. It is recommended that Hi3518E V200 and later chips use this MPI to set the effect parameters of the advanced interface.

[Syntax]

```
HI_S32 HI_MPI_VPSS_SetGrpNRBParam(VPSS_GRP_VpssGrp, VPSS_GRP_NRB_PARAM_S
*pstNRBParam);
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range: [0, VPSS_MAX_GRP_NUM)	Input
pstNRBParam	Effect parameters of the advanced 3DNR interface	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Difference]

Chip	Description
Hi3516A	This MPI is not supported.
Hi3518E V200	This MPI is supported.
Hi3519 V100	This MPI is supported.

[Requirement]

- Header files: hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpia.a

[Note]



- A group must be created before this MPI is called.
- The enumerated member **enNRVer** in **pstNRBParam** indicates the version of the advanced interface. This member is designed for the sake of extensibility, and its initial value is **VPSS_NR_V1**. The corresponding parameter structures of different advanced interface versions are stored in the union of **pstNRBParam**. Different chips may support different interface versions or the same interface version. One chip may support multiple interface versions. For details, see [VPSS_GRP_NRB_PARAM_S](#).
- **HI_MPI_VPSS_SetGrpNRBParam** is supported by multiple chip versions. For Hi3519 V100, **HI_MPI_VPSS_SetNRBParam** is implemented by the VPSS_NR_V2 version of **HI_MPI_VPSS_SetGrpNRBParam**. You are advised to replace **HI_MPI_VPSS_SetNRBParam** and **HI_MPI_VPSS_GetNRBParam** with **HI_MPI_VPSS_SetGrpNRBParam** and **HI_MPI_VPSS_GetGrpNRBParam**, respectively.

[Example]

None

[See Also]

None

HI_MPI_VPSS_GetGrpNRBParam

[Description]

Obtains the effect parameters of the advanced 3DNR interface of the VPSS group. It is recommended that Hi3518E V200 and later chips use this MPI to obtain the effect parameters of the advanced interface.

[Syntax]

```
HI_S32 HI_MPI_VPSS_GetGrpNRBParam(VPSS_GRP VpssGrp, VPSS_GRP_NRB_PARAM_S  
*pstNRBParam);
```

[Parameter]

Parameter	Description	Input/Output
VpssGrp	VPSS group ID Value range: [0,VPSS_MAX_GRP_NUM)	Input
pstNRBParam	Effect parameters of the advanced 3DNR interface	Input/Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 5.5 "Error Codes."

[Difference]



Chip	Description
Hi3516A	This MPI is not supported.
Hi3518E V200	This MPI is supported.
Hi3519 V100	This MPI is supported.

[Requirement]

- Header files:hi_comm_vpss.h, mpi_vpss.h
- Library file: libmpi.a

[Note]

A group must be created before this MPI is called.

[Example]

None

[See Also]

None

5.4 Data Structures

The VPSS data structures are as follows:

- [VPSS_MAX_GRP_NUM](#): Defines the maximum number of VPSS groups.
- [VPSS_MAX_CHN_NUM](#): Defines the maximum number of VPSS channels.
- [VPSS_MAX_PHY_CHN_NUM](#): Defines the maximum number of physical VPSS channels.
- [VPSS_MAX_EXT_CHN_NUM](#): Defines the maximum number of extended VPSS channels.
- [VPSS_MIN_IMAGE_WIDTH](#): Defines the minimum width of the VPSS picture.
- [VPSS_MIN_IMAGE_HEIGHT](#): Defines the minimum height of the VPSS picture.
- [VPSS_MAX_IMAGE_WIDTH](#): Defines the maximum width of the VPSS picture.
- [VPSS_MAX_IMAGE_HEIGHT](#): Defines the maximum height of the VPSS picture.
- [VPSS_OFFLINE_MAX_IMAGE_WIDTH](#): Defines the maximum width of the offline VPSS picture.
- [VPSS_ONLINE_MAX_IMAGE_WIDTH](#): Defines the maximum width of the online VPSS picture.
- [VPSS_EXTCNH_MAX_IMAGE_WIDTH](#): Defines the maximum width of the extended VPSS channel.
- [VPSS_EXTCNH_MAX_IMAGE_HEIGHT](#): Defines the maximum height of the extended VPSS channel.
- [VPSS_MAX_ZOOMIN](#): Defines the maximum amplification multiple of the physical VPSS channel.



- **VPSS_MAX_ZOOMOUT**: Defines the maximum minification multiple of the physical VPSS channel.
- **VPSS_EXT_CHN_MAX_ZOOMIN**: Defines the maximum amplification multiple of the extended VPSS channel.
- **VPSS_EXT_CHN_MAX_ZOOMOUT**: Defines the maximum minification multiple of the extended VPSS channel.
- **VPSS_GRP**: Defines the VPSS group type.
- **VPSS_CHN**: Defines the VPSS channel type.
- **VPSS_DIE_MODE_E**: Defines the DEI mode.
- **VPSS_CROP_COORDINATE_E**: Defines the type of the coordinates of the crop start point.
- **LDC_VIEW_TYPE_E**: Defines the LDC mode.
- **ROTATE_E**: Defines the rotation enumeration.
- **VPSS_REF_SEL_MODE_E**: Defines the NR reference frame source enumeration.
- **RECT_S**: Defines a rectangular area.
- **VPSS_CROP_INFO_S**: Defines the information required by the crop function.
- **VPSS_GRP_ATTR_S**: Defines the static attributes of a VPSS groups.
- **BORDER_S**: Defines the attributes of video borders.
- **VPSS_CHN_ATTR_S**: Defines the static attributes of a VPSS physical channel.
- **VPSS_GRP_PARAM_S**: Defines the advanced VPSS attributes.
- **VPSS_CHN_MODE_E**: Defines the working mode of a VPSS channel.
- **VPSS_CHN_MODE_S**: Defines the structure of the working mode of a VPSS channel.
- **VPSS_CHN_PARAM_S**: Sets the advanced channel attributes.
- **VPSS_FRAME_RATE_S**: Defines the VPSS frame rate control attributes.
- **VPSS_EXT_CHN_ATTR_S**: Defines the attributes of an extended VPSS channel.
- **VPSS_REGION_INFO_S**: Defines the VPSS region information for obtaining the total region luminance.
- **VPSS_LDC_ATTR_S**: Defines LDC attributes of VPSS channels for controlling channel picture correction.
- **VPSS_LOW_DELAY_INFO_S**: Defines short delay attributes of VPSS channels.
- **VPSS_GRP_PARAM_V2_S**: Defines the VPSS 3DNR parameters (version 2).
- **VPSS_NR_PARAM_U**: Defines the VPSS 3DNR parameter union.
- **VPSS_NR_PARAM_V1_S**: Defines the VPSS 3DNR parameters (version 1).
- **FISHEYE_CONFIG_S**: Defines the configuration of the LMF parameters for the fisheye lens.
- **FISHEYE_MOUNT_MODE_E**: Defines the installation mode in the fisheye attribute.
- **FISHEYE_VIEW_MODE_E**: Defines the correction mode in the fisheye attribute.
- **FISHEYE_REGION_ATTR_S**: Defines the attribute configuration in each fisheye correction region.
- **FISHEYE_ATTR_S**: Defines the configuration related to the fisheye attribute.
- **VPSS_MOD_PARAM_S**: Defines the structure of the VPSS module parameters.
- **VPSS_GRP_VPPNRB19CORE_S**: Defines the information of some 3DNR parameters.
- **VPSS_GRP_VPPNRB_S**: Defines the information of the 3DNR parameters so that the information can be set and obtained.



- [VPSS_GRP_VPPNRBCORE_S](#): Defines the information of some 3DNR parameters.
- [VPSS_GRP_VPPNRBEX_S](#): Defines the information of the 3DNR parameters so that the information can be set and obtained.
- [NRS_PARAM_V1_S](#): Defines the parameters of the standard 3DNR interface for Hi3518E V200.
- [NRB_SF_PARAM_V1_S](#): Defines spatial filtering parameters of the advanced 3DNR interface for Hi3518E V200.
- [NRB_TF_PARAM_V1_S](#): Defines temporal filtering parameters of the advanced 3DNR interface for Hi3518E V200.
- [NRB_PARAM_V1_S](#): Defines the parameters of the advanced 3DNR interface for Hi3518E V200.
- [NRB_PARAM_V2_S](#): Defines the parameters of the advanced 3DNR interface for Hi3519 V100.
- [VPSS_NR_VER_E](#): Defines the version of the 3DNR interface.
- [VPSS_GRP_NRS_PARAM_S](#): Defines the parameters of the standard 3DNR interface.
- [VPSS_GRP_NRB_PARAM_S](#): Defines the parameters of the advanced 3DNR interface.

VPSS_MAX_GRP_NUM

[Description]

Defines the maximum number of VPSS groups.

[Syntax]

```
#define VPSS_MAX_GRP_NUM    128      //Hi3516A
#define VPSS_MAX_GRP_NUM    32       //Hi3518EV200
#define VPSS_MAX_GRP_NUM    32       //Hi3519V100
```

[Note]

There is only one VPSS group (ID: 0) in online mode.

[See Also]

None

VPSS_MAX_CHN_NUM

[Description]

Defines the maximum number of VPSS channels.

[Syntax]

```
#define VPSS_MAX_CHN_NUM    (VPSS_MAX_PHY_CHN_NUM + VPSS_MAX_EXT_CHN_NUM)
```

[Note]

The total number of channels include the numbers of physical channels and extended channels.

[See Also]

None



VPSS_MAX_PHY_CHN_NUM

[Description]

Defines the maximum number of physical VPSS channels.

[Syntax]

```
#define VPSS_MAX_PHY_CHN_NUM    4
```

[Note]

The maximum number is related to hardware resources.

[See Also]

None

VPSS_MAX_EXT_CHN_NUM

[Description]

Defines the maximum number of extended VPSS channels.

[Syntax]

```
#define VPSS_MAX_EXT_CHN_NUM 3
```

[Note]

The maximum number depends on solution design and system performance.

[See Also]

None

VPSS_MIN_IMAGE_WIDTH

[Description]

Defines the minimum width of the VPSS picture.

[Syntax]

```
#define VPSS_MIN_IMAGE_WIDTH    64
```

[Note]

None

[See Also]

None

VPSS_MIN_IMAGE_HEIGHT

[Description]

Defines the minimum height of the VPSS picture.

[Syntax]



```
#define VPSS_MIN_IMAGE_HEIGHT 64
```

[Note]

None

[See Also]

None

VPSS_MAX_IMAGE_WIDTH

[Description]

Defines the maximum width of the VPSS picture.

[Syntax]

Hi3516A:

```
#define VPSS_MAX_IMAGE_WIDTH 2592
```

Hi3518E V200:

```
#define VPSS_MAX_IMAGE_WIDTH 2048
```

[Note]

For Hi3519 V100, the maximum width of the VPSS picture differs in online mode and offline mode. This macro is not used. VPSS_OFFLINE_MAX_IMAGE_WIDTH and VPSS_ONLINE_MAX_IMAGE_WIDTH are used to indicate the maximum width in online mode and offline mode respectively.

[See Also]

None

VPSS_MAX_IMAGE_HEIGHT

[Description]

Defines the maximum height of the VPSS picture.

[Syntax]

Hi3516A:

```
#define VPSS_MAX_IMAGE_HEIGHT 2592
```

Hi3518E V200:

```
#define VPSS_MAX_IMAGE_HEIGHT 2048
```

Hi3519 V100:

```
#define VPSS_MAX_IMAGE_HEIGHT 4096
```

[Note]

None

[See Also]



None

VPSS_OFFLINE_MAX_IMAGE_WIDTH

[Description]

Defines the maximum width of the VPSS offline picture.

[Syntax]

```
#define VPSS_OFFLINE_MAX_IMAGE_WIDTH    4608
```

[Note]

For Hi3516A/Hi3518E V200, the maximum widths of pictures in offline mode and online mode are the same. Therefore, this macro is not used.

[See Also]

None

VPSS_ONLINE_MAX_IMAGE_WIDTH

[Description]

Defines the maximum width of the online VPSS picture.

[Syntax]

Hi3519 V100:

```
#define VPSS_ONLINE_MAX_IMAGE_WIDTH    4096
```

[Note]

For Hi3516A/Hi3518E V200, the maximum widths of pictures in offline mode and online mode are the same. Therefore, this macro is not used.

[See Also]

None

VPSS_EXTCHN_MAX_IMAGE_WIDTH

[Description]

Defines the maximum width of the extended VPSS channel.

[Syntax]

Hi3516A:

```
#define VPSS_EXTCHN_MAX_IMAGE_WIDTH    4096
```

Hi3518E V200:

```
#define VPSS_EXTCHN_MAX_IMAGE_WIDTH    VPSS_MAX_IMAGE_WIDTH
```

Hi3519 V100:

```
#define VPSS_EXTCHN_MAX_IMAGE_WIDTH    VPSS_OFFLINE_MAX_IMAGE_WIDTH
```



[Note]

None

[See Also]

None

VPSS_EXTCHN_MAX_IMAGE_HEIGHT

[Description]

Defines the maximum height of the extended VPSS channel.

[Syntax]

Hi3516A:

```
#define VPSS_EXTCHN_MAX_IMAGE_HEIGHT 4096
```

Hi3518EV200:

```
#define VPSS_EXTCHN_MAX_IMAGE_HEIGHT VPSS_MAX_IMAGE_HEIGHT
```

Hi3519V100:

```
#define VPSS_EXTCHN_MAX_IMAGE_HEIGHT VPSS_MAX_IMAGE_HEIGHT
```

[Note]

None

[See Also]

None

VPSS_MAX_ZOOMIN

[Description]

Defines the maximum amplification multiple of the physical VPSS channel.

[Syntax]

Hi3516A:

```
#define VPSS_MAX_ZOOMIN 1
```

Hi3518E V200:

```
#define VPSS_MAX_ZOOMIN 2
```

Hi3519 V100:

```
#define VPSS_MAX_ZOOMIN 16
```

[Note]

None

[See Also]



None

VPSS_MAX_ZOOMOUT

[Description]

Defines the maximum minification multiple of the physical VPSS channel.

[Syntax]

```
#define VPSS_MAX_ZOOMOUT 15
```

[Note]

None

[See Also]

None

VPSS_EXT_CHN_MAX_ZOOMIN

[Description]

Defines the maximum amplification multiple of the extended VPSS channel.

[Syntax]

```
#define VPSS_EXT_CHN_MAX_ZOOMIN 16
```

[Note]

None

[See Also]

None

VPSS_EXT_CHN_MAX_ZOOMOUT

[Description]

Defines the maximum minification multiple of the extended VPSS channel.

[Syntax]

```
#define VPSS_EXT_CHN_MAX_ZOOMOUT 15
```

[Note]

None

[See Also]

None

VPSS_GRP

[Description]

Defines the VPSS group type.



[Syntax]

```
typedef HI_S32 VPSS_GRP;
```

[Note]

None

[See Also]

None

VPSS_CHN

[Description]

Defines the VPSS channel type.

[Syntax]

```
typedef HI_S32 VPSS_CHN;
```

[Note]

None

[See Also]

None

VPSS_DIE_MODE_E

[Description]

Defines the DEI mode.

[Syntax]

```
typedef enum hiVPSS_DIE_MODE_E
{
    VPSS_DIE_MODE_AUTO      = 0,
    VPSS_DIE_MODE_NODIE     = 1,
    VPSS_DIE_MODE_DIE       = 2,
    VPSS_DIE_MODE_BUTT
} VPSS_DIE_MODE_E;
```

[Member]

Member	Description
VPSS_DIE_MODE_AUTO	DEI is performed automatically based on the current picture format.
VPSS_DIE_MODE_NODIE	DEI is forcibly forbidden.
VPSS_DIE_MODE_DIE	DEI is forcibly performed.



[Note]

The Hi3516A/Hi3518E V200/Hi3519 V100 does not support the DEI function. Therefore, the DEI mode needs to be set to **VPSS_DIE_MODE_NODIE**.

[See Also]

[VPSS_GRP_ATTR_S](#)

VPSS_CROP_COORDINATE_E

[Description]

Defines the type of the coordinates of the crop start point.

[Syntax]

```
typedef enum hiVPSS_CROP_COORDINATE_E
{
    VPSS_CROP_RATIO_COOR = 0,
    VPSS_CROP_ABS_COOR
} VPSS_CROP_COORDINATE_E;
```

[Member]

Member	Description
VPSS_CROP_RATIO_COOR	Relative coordinate
VPSS_CROP_ABS_COOR	Absolute coordinate

[Note]

The relative coordinates indicate that the coordinates of the start point are represented by the aspect ratio of the current picture. The relative coordinates need to be converted in the actual application scenario. For details, see the description of [VPSS_CROP_INFO_S](#).

[See Also]

[VPSS_CROP_INFO_S](#)

LDC_VIEW_TYPE_E

[Description]

Defines the LDC mode.

[Syntax]

```
typedef enum hiLDC_VIEW_TYPE_E
{
    LDC_VIEW_TYPE_ALL = 0,
    LDC_VIEW_TYPE_CROP,
    LDC_VIEW_TYPE_BUTT,
} LDC_VIEW_TYPE_E;
```



[Member]

Member	Description
LDC_VIEW_TYPE_ALL	Full mode. The maximum rectangle is obtained based on the edges of the corrected picture, and then the rectangle is zoomed out to the source picture size. Picture edges may be retained.
LDC_VIEW_TYPE_CROP	Cropping mode. The corrected picture is cropped based on the source picture size, and picture edges may be lost.

[Difference]

Chip	LDC_VIEW_TYPE_CROP
Hi3516A/Hi3518E V200	The corrected picture is cropped and the size of the cropped region is equal to the size of the source picture. However, picture edges may be lost.
Hi3519 V100	LDC is not supported.

[Note]

None

[See Also]

[VPSS_LDC_ATTR_S](#)

ROTATE_E

[Description]

Defines the rotation enumeration.

[Syntax]

```
typedef enum hiROTATE_E
{
    ROTATE_NONE    = 0,
    ROTATE_90      = 1,
    ROTATE_180     = 2,
    ROTATE_270     = 3,
    ROTATE_BUTT
} ROTATE_E;
```

[Member]

Member	Description
ROTATE_NONE	No rotation



Member	Description
ROTATE_90	Clockwise rotation by 90°
ROTATE_180	Clockwise rotation by 180°
ROTATE_270	Clockwise rotation by 270°

[Note]

None

[See Also]

- [HI_MPI_VPSS_SetRotate](#)
- [HI_MPI_VPSS_GetRotate](#)

VPSS_REF_SEL_MODE_E

[Description]

Defines the NR reference frame source enumeration.

[Syntax]

```
typedef enum hiVPSS_REF_SEL_MODE_E
{
    VPSS_REF_FROM_RFR = 0,
    VPSS_REF_FROM_CHN0 =1,
    VPSS_REF_FROM_BUTT
}VPSS_REF_SEL_MODE_E;
```

[Member]

Member	Description
VPSS_REF_FROM_RFR	Reconstruction frame as the reference frame
VPSS_REF_FROM_CHN0	Channel 0 output as the reference frame

[Note]

None

[See Also]

- [HI_MPI_VPSS_SetRefSelect](#)
- [HI_MPI_VPSS_GetRefSelect](#)

RECT_S

[Description]

Defines a rectangular area.



[Syntax]

```
typedef struct hiRECT_S
{
    HI_S32 s32X;
    HI_S32 s32Y;
    HI_U32 u32Width;
    HI_U32 u32Height;
}RECT_S;
```

[Member]

Member	Description
s32X	Horizontal coordinate of the start point
s32Y	Vertical coordinate of the start point
u32Width	Width of the rectangle
u32Height	Height of the rectangle

[Note]

None

[See Also]

- [VPSS_CROP_INFO_S](#)
- [VPSS_REGION_INFO_S](#)

VPSS_CROP_INFO_S

[Description]

Defines the information required by the crop function.

[Syntax]

```
typedef struct hivPSS_CROP_INFO_S
{
    HI_BOOL bEnable;
    VPSS_CROP_COORDINATE_E enCropCoordinate;
    RECT_S stCropRect;
}VPSS_CROP_INFO_S;
```

[Member]

Member	Description
bEnable	Crop enable
enCropCoordinate	Coordinate type of the crop start point



Member	Description
stCropRect	Cropped rectangular area

[Note]

- If **enCropCoordinate** is **VPSS_CROP_RATIO_COOR** (relative coordinates), the **stCropRect** members need to be converted as follows: **s32X** = Horizontal coordinate of the start point x Original picture width/1000. The value range is [0, 999]. After conversion, the rounding operation and 2-byte-alignment operation are performed. The formula also applies to the vertical coordinate.
u32Width = Region width x Actual picture width/1000. The value range of the region width is [0, 1000]. The formula also applies to the region height.
- If **enCropCoordinate** is set to **VPSS_CROP_ABS_COOR** (absolute coordinate mode), the width, height, and coordinates must be 2-pixel-aligned for the Hi3516A. For Hi3518E V200/Hi3519 V100, the width and height must be 4-pixel-aligned and the coordinate must be 2-pixel-aligned. For the extend channel, the width, height, and coordinates must be 2-pixel-aligned.

[See Also]

[VPSS_CROP_COORDINATE_E](#)

VPSS_GRP_ATTR_S

[Description]

Defines the static attributes of a VPSS group.

[Syntax]

```
typedef struct hiVPSS_GRP_ATTR_S
{
    HI_U32    u32MaxW;
    HI_U32    u32MaxH;
    PIXEL_FORMAT_E enPixFmt;
    HI_BOOL   bIeEn;
    HI_BOOL   bDciEn;
    HI_BOOL   bNrEn;
    HI_BOOL   bHistEn;
    VPSS_DIE_MODE_E enDieMode;
}VPSS_GRP_ATTR_S;
```

[Member]



Member	Description	
	Hi3516A	Hi3518E V200/Hi3519 V100
u32MaxW	Maximum picture width Value range: [VPSS_MIN_IMAGE_WIDTH, VPSS_MAX_IMAGE_WIDTH] It is a static attribute. That is, it cannot be changed after it is set when a group is created.	Maximum picture width • Hi3518E V200: [VPSS_MIN_IMAGE_WIDTH, VPSS_MAX_IMAGE_WIDTH] • Hi3519 V100: [VPSS_MIN_IMAGE_WIDTH, VPSS_ONLINE_MAX_IMAGE_WIDTH] (in online mode) or [VPSS_MIN_IMAGE_WIDTH, VPSS_OFFLINE_MAX_IMAGE_WIDTH] (in offline mode) It is a static attribute. That is, it cannot be changed after it is set when a group is created.
u32MaxH	Maximum picture height Value range: [VPSS_MIN_IMAGE_HEIGHT, VPSS_MAX_IMAGE_HEIGHT] It is a static attribute. That is, it cannot be changed after it is set when a group is created.	Maximum picture height Value range: [VPSS_MIN_IMAGE_HEIGHT, VPSS_MAX_IMAGE_HEIGHT]
enPixFmt	Pixel format. Only the semi-planar422 and semi-planar420 formats are supported. It is a static attribute. That is, it cannot be changed after it is set when a group is created.	Pixel format of the input picture. Only the semi-planar422, semi-planar420, and YUV400 formats are supported. It is a static attribute. That is, it cannot be changed after it is set when a group is created.
bIeEn	Reserved. This member must be set to HI_FALSE .	Reserved. This member must be set to HI_FALSE .
bDciEn	Reserved. This member must be set to HI_FALSE .	Reserved. This member must be set to HI_FALSE .
bNrEn	NR enable	NR enable
bHistEn	Reserved. This member must be set to HI_FALSE .	Reserved. This member must be set to HI_FALSE .
enDieMode	Reserved. This member must be set to VPSS_DIE_MODE_NODIE .	Reserved. This member must be set to VPSS_DIE_MODE_NODIE .

[Note]



- **u32MaxW**, **u32MaxH**, and **enPixFmt** define the maximum size (width x height) of the picture received by the VPSS group in offline mode or the size of the picture received by the current VPSS group in online mode. The three parameters are configured when a group is created and cannot be dynamically changed.
- In online mode, the size of picture received by the current group must be the same as the size of the output picture of the VIU (specified in the VI channel attributes).

[See Also]

- [PIXEL_FORMAT_E](#)
- [VPSS_DIE_MODE_E](#)
- [HI_MPI_VPSS_CreateGrp](#)
- [HI_MPI_VPSS_SetGrpAttr](#)
- [HI_MPI_VPSS_GetGrpAttr](#)

BORDER_S

[Description]

Defines the attributes of video borders.

[Syntax]

```
typedef struct hiBORDER_S
{
    HI_U32 u32TopWidth;
    HI_U32 u32BottomWidth;
    HI_U32 u32LeftWidth;
    HI_U32 u32RightWidth;
    HI_U32 u32Color;
} BORDER_S;
```

[Member]

Member	Description
u32TopWidth	Top border width
u32BottomWidth	Bottom border width
u32LeftWidth	Left border width
u32RightWidth	Right border width
u32Color	Border color

[Note]

- The border width must be an even number ranging from 0 to 14.
- The border color is in RGB format. The R, G, and B components correspond to the following bits as follows:

31 23 15 7 0



|-----|-----R-----|-----G-----|-----B-----|

- The border color applies to four borders, that is, the four borders are in the same color.

[See Also]

[VPSS_CHN_ATTR_S](#)

VPSS_CHN_ATTR_S

[Description]

Defines the static attributes of a VPSS physical channel.

[Syntax]

```
typedef struct hiVPSS_CHN_ATTR_S
{
    HI_BOOL bSpEn;
    HI_BOOL bBorderEn;
    HI_BOOL bMirror;
    HI_BOOL bFlip;
    HI_S32 s32SrcFrameRate;
    HI_S32 s32DstFrameRate;
    BORDER_S stBorder;
}VPSS_CHN_ATTR_S;
```

[Member]

Member	Description
bSpEn	SP enable. It must be set to HI_FALSE .
bBorderEn	Border enable. It must be set to HI_FALSE .
bMirror	Mirror enable
bFlip	Flip enable
s32SrcFrameRate	Channel source frame rate control Value range: (0, 240] and -1
s32DstFrameRate	Channel target frame rate control Value range: [-1, 240]
stBorder	Border attributes

[Note]

- The channel frame rate control function is valid in VI-VPSS online and offline modes.
- If both the source frame rate and target frame rate are -1, the frame rate is not controlled.
- The target frame rate must be lower than or equal to the source frame rate.
- When the 3DNR reference frame source is **VPSS_REF_FROM_CHN0**, if the mirror/flip, rotation, scaling, and LDC functions are configured for channel 0, 3DNR



does not take effect. If the pixel format of the input picture is different from that of the output picture of the channel 0 (assuming that pixel format conversion is supported), the logmpp error information is displayed, and channel 0 does not output pictures.

- The online low-delay channel does not support flip.

[See Also]

[HI_MPI_VPSS_SetChnAttr](#)

VPSS_GRP_PARAM_S

[Description]

Defines the advanced VPSS attributes.

[Syntax]

```
typedef struct hivpss_grp_param_s
{
    HI_U32 u32Contrast;
    HI_S32 s32GlobalStrength;
    HI_S32 s32IeStrength;
    HI_S32 s32YSFStrength;
    HI_S32 s32YTFStrength;
    HI_S32 s32CSFStrength;
    HI_S32 s32CTFStrength;
    HI_S32 s32MotionLimen;
}VPSS_GRP_PARAM_S;
```

[Member]

Member	Description
u32Contrast	Reserved
s32GlobalStrength	3DNR strength. This parameter is the total NR strength. The value range is [0, 1408] and the default value is 128. When the value is 0, there is no noise reduction effect.
s32IeStrength	Picture texture enhancement. This parameter is the NR strength. The value range is [-1, +100] and the default value is -1. When the value is -1, the internal algorithm generates a parameter value by default.
s32YSFStrength	Strength of luminance spatial-domain NR. This parameter is the NR strength. The value range is [-1, +9999] and the default value is -1. When the value is -1, the internal algorithm generates a parameter value by default. This value affects the strength of luminance spatial filtering. A larger value indicates higher filtering strength.



Member	Description
s32YTFStrength	Strength of luminance time domain NR. This parameter is the NR strength. The value range is $[-1, +15]$ and the default value is -1 . When the value is -1 , the internal algorithm generates a parameter value by default. This value affects the strength of luminance time domain filtering. A larger value indicates higher filtering strength.
s32CSFStrength	Strength of chrominance spatial-domain NR. This parameter is the NR strength. The value range is $[-1, +255]$ and the default value is -1 . When the value is -1 , the internal algorithm generates a parameter value by default. This value affects the strength of chrominance spatial filtering. A larger value indicates higher filtering strength.
s32CTFStrength	Strength of chrominance time domain NR. This parameter is the NR strength. The value range is $[-1, +32]$ and the default value is -1 . When the value is -1 , the internal algorithm generates a parameter value by default. This value affects the strength of chrominance time domain filtering. A larger value indicates higher filtering strength.
s32MotionLimen	Motion threshold. This parameter is the NR strength. The value range is $[-1, +511]$ and the default value is -1 . When the value is -1 , the internal algorithm generates a parameter value by default. This value is used to determine whether a pixel is a motion pixel.

[Note]

None

[See Also]

- [HI_MPI_VPSS_SetGrpParam](#)
- [HI_MPI_VPSS_GetGrpParam](#)

VPSS_CHN_MODE_E

[Description]

Defines the working mode of a VPSS channel.

[Syntax]

```
typedef enum hiVPSS_CHN_MODE_E
```



```
{  
    VPSS_CHN_MODE_AUTO = 0,  
    VPSS_CHN_MODE_USER = 1  
}VPSS_CHN_MODE_E;
```

[Member]

Member	Description
VPSS_CHN_MODE_AUTO	Automatic mode. This mode is the default channel working mode.
VPSS_CHN_MODE_USER	User-defined mode

[Note]

None

[See Also]

[VPSS_CHN_MODE_S](#)

VPSS_CHN_MODE_S

[Description]

Defines the structure of the working mode of a VPSS channel.

[Syntax]

```
typedef struct hiVPSS_CHN_MODE_S  
{  
    VPSS_CHN_MODE_E enChnMode;  
    HI_U32 u32Width;  
    HI_U32 u32Height;  
    HI_BOOL bDouble;  
    PIXEL_FORMAT_E enPixelFormat;  
    COMPRESS_MODE_E enCompressMode;  
}VPSS_CHN_MODE_S;
```

[Member]

Member	Description
enChnMode	Working mode of a VPSS channel
u32Width	Target picture width
u32Height	Target picture height
bDouble	Reserved
enPixelFormat	Pixel format of the target picture



Member	Description
enCompressMode	Compression mode of the target picture

[Difference]

Chip	Description	
Hi3516A	Online mode	The value ranges of u32Width and u32Height vary according to channels, and u32Width and u32Height must be set to even numbers.
	Offline mode	The values of u32Width and u32Height must fall within the ranges supported by the VPSS and must be even numbers. For details, see the description of VPSS_GRP_ATTR_S .
	The output compression mode can be COMPRESS_MODE_NONE or COMPRESS_MODE_SEG .	
Hi3518E V200	The width and height must be 4-pixel-aligned. The output compression mode of channel 3 must be COMPRESS_MODE_NONE .	
Hi3519 V100	The width and height must be 4-pixel-aligned. The output compression mode can be COMPRESS_MODE_NONE or COMPRESS_MODE_SEG . If the width exceeds 4096, compression is not supported.	

[Note]

When the value of **u32Width** or **u32Height** is greater than the width or height of the scaled picture supported by the channel, scaling fails. In this case, you need to set **u32Width** and **u32Height** to the same values of the input picture. Then the picture is not scaled by default. This ensures that the picture size is the same as the size of the captured source picture after channel cropping.

[See Also]

- [HI_MPI_VPSS_SetChnMode](#)
- [HI_MPI_VPSS_GetChnMode](#)

VPSS_CHN_PARAM_S

[Description]

Defines VPSS channel advanced attributes.

[Syntax]

```
typedef struct hiVPSS_CHN_PARAM_S
{
    HI_U32 u32SpStrength;
} VPSS_CHN_PARAM_S;
```

[Member]



Member	Description
u32SpStrength	Channel SP strength Value range: [0, 100]

[Difference]

Chip	Description
Hi3516A/Hi3518E V200/Hi3519 V100	The u32SpStrength parameter is invalid because the Hi3516A/Hi3518E V200/Hi3519 V100 does not support SP.

[Note]

None

[See Also]

- [HI_MPI_VPSS_SetChnParam](#)
- [HI_MPI_VPSS_GetChnParam](#)

VPSS_FRAME_RATE_S

[Description]

Defines the VPSS frame rate control attributes.

[Syntax]

```
typedef struct hiVPSS_FRAME_RATE_S
{
    HI_S32 s32SrcFrmRate;
    HI_S32 s32DstFrmRate;
} VPSS_FRAME_RATE_S;
```

[Member]

Member	Description
s32SrcFrmRate	Source frame rate Value range: (0, 240] and -1
s32DstFrmRate	Target frame rate Value range: [-1, +240]

[Note]

- If both the source frame rate and target frame rate are -1, the frame rate is not controlled.
- The target frame rate must be lower than or equal to the source frame rate.



[See Also]

- [HI_MPI_VPSS_SetGrpFrameRate](#)
- [HI_MPI_VPSS_GetGrpFrameRate](#)

VPSS_EXT_CHN_ATTR_S

[Description]

Defines the attributes of an extended VPSS channel.

[Syntax]

```
typedef struct hiVPSS_EXT_CHN_ATTR_S
{
    VPSS_CHN    s32BindChn;
    HI_U32     u32Width;
    HI_U32     u32Height;
    HI_S32     s32SrcFrameRate;
    HI_S32     s32DstFrameRate;
    PIXEL_FORMAT_E enPixelFormat;
    COMPRESS_MODE_E enCompressMode;
} VPSS_EXT_CHN_ATTR_S;
```

[Member]

Member	Description
s32BindChn	Source physical channel to be bound
u32Width	Target output width of an extended channel
u32Height	Target output height of an extended channel
s32SrcFrameRate	Source input frame rate of an extended channel
s32DstFrameRate	Target output frame rate of an extended channel
enPixelFormat	Output picture format of an extended channel
enCompressMode	Output Compression format of an extended channel

[Difference]

Chip	Description
Hi3516A	The output data format can be YUV420 semi-planar or YUV422 semi-planar. The width and height must be 2-pixel-aligned.
Hi3518E V200/Hi3519 V100	The output data format can be YUV420 semi-planar, YUV422 semi-planar, or YUV400. The width and height must be 2-pixel-aligned.



[Note]

- The ID of the physical channel to be bound must fall within [0, [VPSS_MAX_PHY_CHN_NUM](#)).
- The channel width and height must be 2-pixel-aligned.
- The target frame rate must be lower than or equal to the source frame rate. If both the target frame rate and source frame rate are -1, the frame rate is not controlled.
- When the source physical channel of the extended channel is a low-delay channel, the extended channel receives no data.
- The extended channel of Hi3518E V200/Hi3519 V100 supports the YUV400 and conversion from semi-planar 420/semi-planar 422 to the YUV400 format.
- The Hi3516A does not support the YUV400.

[See Also]

[VPSS_REGION_INFO_S](#)

VPSS_REGION_INFO_S

[Description]

Defines the VPSS region information for obtaining the total region luminance.

[Syntax]

```
typedef struct hiVPSS_REGION_INFO_S
{
    RECT\_S *pstRegion;
    HI_U32 u32RegionNum;
} VPSS_REGION_INFO_S;
```

[Member]

Member	Description
pstRegion	Region attributes, including the start position, width, and height
u32RegionNum	Number of regions

[Note]

The start position, width, and height of a region must be 2-pixel-aligned.

[See Also]

- [RECT_S](#)
- [HI_MPI_VPSS_GetRegionLuma](#)

VPSS_LDC_ATTR_S

[Description]

Defines LDC attributes of VPSS channels for controlling channel picture correction.



[Syntax]

```
typedef struct hiLDC_ATTR_S
{
    LDC_VIEW_TYPE_E enViewType;
    HI_S32 s32CenterXOffset;
    HI_S32 s32CenterYOffset;
    HI_S32 s32Ratio;
} LDC_ATTR_S;

typedef struct hiVPSS_LDC_ATTR_S
{
    HI_BOOL bEnable;
    LDC_ATTR_S stAttr;
} VPSS_LDC_ATTR_S;
```

[Member]

Member		Description
bEnable		LDC enable
stAttr	enViewType	LDC mode (cropping mode or full mode)
	s32CenterXOffset	Horizontal offset of the distortion center relative to the picture center Value range: [-75, +75]
	s32CenterYOffset	Vertical offset of the distortion center relative to the picture center Value range: [-75, +75]
	s32Ratio	Distortion ratio Value range: [0, 511]

[Note]

The distortion ratio parameter **s32Ratio** varies according to the resolution of the input picture.

Hi3516A:

- When the resolution of the input picture is not greater than D1, the value range of **s32Ratio** is [0, 480].
- When the resolution of the input picture is greater than D1 but not greater than 720p, the value range of **s32Ratio** is [0, 433].
- When the resolution of the input picture is greater than 720p but not greater than 1080p, the value range of **s32Ratio** is [0, 400].
- When the resolution of the input picture is greater than 1080p but not greater than 2304 x 1536, the value range of **s32Ratio** is [0, 300].
- When the resolution of the input picture is greater than 2304 x 1536 but not greater than 5 megapixels, the value range of **s32Ratio** is [0, 168].

Hi3519 V100:



Hi3519 V100 does not support LDC.

[See Also]

- [HI_MPI_VPSS_SetLDCAttr](#)
- [HI_MPI_VPSS_GetLDCAttr](#)

VPSS_LOW_DELAY_INFO_S

[Description]

Defines short delay attributes of VPSS channels.

[Syntax]

```
typedef struct hiVPSS_LOW_DELAY_INFO_S
{
    HI_BOOL bEnable;
    HI_U32 u32LineCnt;
} VPSS_LOW_DELAY_INFO_S;
```

[Member]

Member	Description
bEnable	Short delay enable Default value: HI_FALSE
u32LineCnt	Number of lead lines for short delay The value must be greater than or equal to 16 and less than or equal to the output picture height specified in chnMode . Default value: 16

[Note]

None

[See Also]

- [HI_MPI_VPSS_SetLowDelayAttr](#)
- [HI_MPI_VPSS_GetLowDelayAttr](#)

VPSS_GRP_PARAM_V2_S

[Description]

Defines the VPSS 3DNR parameters (version 2).

[Syntax]

```
typedef struct hiVPSS_GRP_PARAM_V2_S
{
    HI_U8 Chroma_SF_Strength;
    HI_U8 Chroma_TF_Strength;
```



```
    HI_U16  IE_PostFlag;
    HI_U16  IE_Strength;
    HI_U16  Luma_MotionThresh;
    HI_U8   Luma_SF_MoveArea;
    HI_U8   Luma_SF_StillArea;
    HI_U8   Luma_TF_Strength;
    HI_U8   DeSand_Strength;
} VPSS_GRP_PARAM_V2_S;
```

[Member]

Member	Description
Chroma_SF_Strength	Strength of chrominance spatial filtering Value range: [0, 255] Default value: 8
Chroma_TF_Strength	Strength of chrominance temporal filtering Value range: [0, 32] Default value: 0
IE_PostFlag	Flag indicating whether texture/edge enhancement is performed before or after other processing Value range: [0, 1] Default value: 0
IE_Strength	Texture/Edge enhancement strength Value range: [0, 63] Default value: 21
Luma_MotionThresh	Threshold of luminance motion detection Value range: [0, 511] Default value: 64
Luma_SF_MoveArea	Strength of luminance spatial filtering for the motion area Value range: [0, 255] Default value: 32
Luma_SF_StillArea	Strength of luminance spatial filtering for the static area Value range: [0, 64] Default value: 32
Luma_TF_Strength	Strength of luminance temporal filtering Value range: [0, 15] Default value: 12
DeSand_Strength	De-granulation strength Value range: [0, 8] Default value: 0



[Note]

None

[See Also]

- [HI_MPI_VPSS_SetGrpParamV2](#)
- [HI_MPI_VPSS_GetGrpParamV2](#)

VPSS_NR_PARAM_U

[Description]

Defines the VPSS 3DNR parameter union.

[Syntax]

```
typedef union hiVPSS_NR_PARAM_U
{
    VPSS_NR_PARAM_V1_S stNRParam_V1;
} VPSS_NR_PARAM_U;
```

[Member]

Member	Description
stNRParam_V1	First version of VPSS 3DNR parameters

[Note]

None

[See Also]

- [HI_MPI_VPSS_SetNRParam](#)
- [HI_MPI_VPSS_GetNRParam](#)

VPSS_NR_PARAM_V1_S

[Description]

Defines the VPSS 3DNR parameters (version 1).

[Syntax]

```
typedef struct hiVPSS_NR_PARAM_V1_S
{
    HI_S32 s32YPKStr;
    HI_S32 s32YSFStr;
    HI_S32 s32YTFStr;
    HI_S32 s32TFStrMax;
    HI_S32 s32TFStrMov;
```



```
HI_S32 s32YSmthStr;
HI_S32 s32YSmthRat;
HI_S32 s32YSFStrDlt;
HI_S32 s32YSFStrDl;
HI_S32 s32YTFStrDlt;
HI_S32 s32YTFStrDl;
HI_S32 s32YSFBriRat;
HI_S32 s32CSFStr;
HI_S32 s32CTFstr;
HI_S32 s32YTFMdWin;
} VPSS_NR_PARAM_V1_S;
```

[Member]

Member	Description
s32YPKStr	Texture enhancement Value range: [0, 63] Default value: 0
s32YSFStr	Strength of main spatial filtering Value range: [0, 200] Default value: 100
s32YTFStr	Strength of main temporal filtering Value range: [0, 128] Default value: 64
s32TFStrMax	Upper limit for the strength of main temporal filtering Value range: [0, 15] Default value: 12
s32TFStrMov	Strength of temporal filtering in the motion region Value range: [0, 31] Default value: 0
s32YSmthStr	Strength of smooth filtering Value range: [0, 200] Default value: 0
s32YSmthRat	Relative strength of smooth filtering Value range: [0, 32] Default value: 16
s32YSFStrDlt	Strength 1 of auxiliary spatial filtering Value range: [-128, +127] Default value: 0



Member	Description
s32YSFStrDl	Strength 2 of auxiliary spatial filtering Value range: [0, 255] Default value: 0
s32YTFStrDlt	Strength 1 of auxiliary temporal filtering Value range: [-64, +63] Default value: 0
s32YTFStrDl	Strength 2 of auxiliary temporal filtering Value range: [0, 31] Default value: 0
s32YSFBriRat	Relative strength of spatial filtering for bright regions Value range: [0, 64] Default value: 24
s32CSFStr	Strength of color difference spatial filtering Value range: [0, 80] Default value: 32
s32CTFstr	Strength of color difference temporal filtering Value range: [0, 32] Default value: 0
s32YTFMdWin	MD window for luminance temporal filtering Value range: [0, 1] Default value: 1 The value 0 indicates a small window and the value 1 indicates a large window.

[Note]

None

[See Also]

- [HI_MPI_VPSS_SetNRParam](#)
- [HI_MPI_VPSS_GetNRParam](#)

FISHEYE_CONFIG_S

For details, see chapter 11 "Fisheye Subsystem" in the *HiMPP IPC V2.0 Media Processing Software Development Reference*.

FISHEYE_MOUNT_MODE_E

For details, see chapter 11 "Fisheye Subsystem" in the *HiMPP IPC V2.0 Media Processing Software Development Reference*.



FISHEYE_VIEW_MODE_E

For details, see chapter 11 "Fisheye Subsystem" in the *HiMPP IPC V2.0 Media Processing Software Development Reference*.

FISHEYE_REGION_ATTR_S

For details, see chapter 11 "Fisheye Subsystem" in the *HiMPP IPC V2.0 Media Processing Software Development Reference*.

FISHEYE_ATTR_S

For details, see chapter 11 "Fisheye Subsystem" in the *HiMPP IPC V2.0 Media Processing Software Development Reference*.

VPSS_MOD_PARAM_S

[Description]

Defines the structure of the VPSS module parameters.

[Syntax]

```
typedef struct hiVPSS_PARAM_MOD_S
{
    HI_BOOL bOneBufForLowDelay;
} VPSS_MOD_PARAM_S;
```

[Member]

Member	Description
bOneBufForLowDelay	Whether the VPSS online low-delay channels use the same VB (single buffer solution). This parameter corresponds to the module parameter bOneBufferforLowDelay .

[Note]

None

[See Also]

- [HI_MPI_VPSS_SetModParam](#)
- [HI_MPI_VPSS_GetModParam](#)

VPSS_GRP_VPPNRB19CORE_S

[Description]

Defines the information of some 3DNR parameters.

[Syntax]

```
typedef tV19zNRbCore VPSS_GRP_VPPNRB19CORE_S;
```



```
typedef struct
{
    HI_U8    EN: 1;
    HI_U8    ISH: 7;
    HI_U8    SBS;
    HI_U8    SDS;
    HI_U8    IDZ: 7;
    HI_U8    STYP: 1;
    HI_U8    SBT;
    HI_U8    IEB;
    HI_U8    SDT;
    HI_U8    ITX;
    HI_U8    MSBF;
    HI_U8    SBF;
    HI_U8    MSTH;
    HI_U8    STH;
    HI_U8    MSDZ;
    HI_U8    SDZ;
    HI_U8    MSHT;
    HI_U8    SHT;
    HI_U8    MSHP;
    HI_U8    SHP;
    HI_U8    MTFR;
    HI_U8    TFR;
    HI_U8    MATH;
    HI_U8    TFS;
    HI_U8    MODZ;
    HI_U8    MATE;
} tV19zNRbCore;
```

[Member]

Member	Description
EN	3DNR enable Value range: [0, 1]
ISH	Value range: [0, 64] You are advised to retain the value.
SBS	Strength of the filtering for NR in the bright region Value range: [0, 255]
SDS	Strength of the filtering for NR in the dark region Value range: [0, 255]



Member	Description
IDZ	Value range: [0, 100] You are advised to retain the value.
STYP	Value range: [0, 1] You are advised to retain the value.
SBT	Value range: [0, 255] You are advised to retain the value.
IEB	Value range: [0, 63] You are advised to retain the value.
SDT	Value range: [0, 255] You are advised to retain the value.
ITX	Value range: [0, 255] You are advised to retain the value.
MSBF	Value range: [0, 128] You are advised to retain the value.
SBF	Value range: [0, 128] You are advised to retain the value.
MSTH	Value range: [0, 100] You are advised to retain the value.
STH	Threshold of the big edge detection. A larger value indicates that more pixels are considered as flat by the big edge detection unit. When spatial filtering is performed on more pixels, the picture becomes smoother and more picture details are lost. Value range: [0, 100]
MSDZ	Strength of texture protection in the motion region. A larger value indicates stronger strength of the texture protection in the motion region, a clearer motion region, and more obvious noises. Value range: [0, 255]
SDZ	Value range: [0, 255] You are advised to retain the value.
MSHT	Value range: [0, 64] You are advised to retain the value.
SHT	Value range: [0, 64] You are advised to retain the value.
MSHP	Value range: [0, 255] You are advised to retain the value.



Member	Description
SHP	Value range: [0, 255] You are advised to retain the value.
MTFR	Value range: [0, 63] You are advised to retain the value.
TFR	Relative strength of the temporal filtering Value range: [0, 63]
MATH	Threshold for the motion detection. A larger value indicates that more pixels are considered as static by the motion detection unit. When temporal filtering is performed on more pixels, the picture noises become less obvious. Value range: [0, 100]
TFS	Absolute strength of the temporal filtering Value range: [0, 15]
MODZ	Value range: [0, 100] You are advised to retain the value.
MATE	Motion threshold. When the value of MATH is fixed, a larger MATE value indicates that more pixels are considered as static by the motion detection unit. When temporal filtering is performed on more pixels, the picture noises become less obvious. Value range: [0, 8]

[Note]

You can retain the values of the parameters with the note "You are advised to retain the value". If the values of these parameters are changed, the NR effect is unpredictable.

[See Also]

- [VPSS_GRP_VPPNRB_S](#)
- [HI_MPI_VPSS_SetNRBParam](#)
- [HI_MPI_VPSS_GetNRBParam](#)

VPSS_GRP_VPPNRB_S

[Description]

Defines the information of the 3DNR parameters so that the information can be set and obtained.

[Syntax]

```
typedef tV19zNRb VPSS_GRP_VPPNRB_S;  
typedef struct
```



```
{  
    VPSS_GRP_VPPNRB19CORE_S  Unit[4];  
    HI_U8  PBW: 1;  
    HI_U8  ClassicEn: 1;  
    HI_U8  PSF: 1;  
    HI_U8  _reserved_: 1;  
    HI_U8  PROW: 4;  
    HI_U8  MOMD: 6;  
    HI_U8  SFyEx: 2;  
    HI_U8  MamiMax;  
    HI_U8  PSBS;  
    HI_U8  PSDS;  
    HI_U8  RefMode;  
    HI_U8  TFC;  
    HI_U8  SFC;  
} tV19zNRb;
```

[Member]

Member	Description
Unit[4]	Four groups of main 3DNR parameters
PBW	Value range: [0, 1] You are advised to retain the value.
ClassicEn	Value range: [0, 1] You are advised to retain the value.
PSF	Value range: [0, 1] You are advised to retain the value.
reserved	Value range: [0, 1] You are advised to retain the value.
PROW	Value range: [0, 15] You are advised to retain the value.
MOMD	Value range: [0, 48] You are advised to retain the value.



Member	Description
SFyEx	<p>Correction on a side effect of the spatial filtering. When the picture noises are obvious, the spatial filtering causes the visible tiny grid-shaped pseudo textures in the flat region.</p> <ul style="list-style-type: none">• When SFyEx is set to 0, the pseudo texture is not processed.• When SFyEx is set to 1, the pseudo texture and the details are reduced.• When SFyEx is set to 2, the pseudo texture and the details are further reduced. <p>Value range: [0, 2]</p>
MamiMax	<p>Value range: [0, 255] You are advised to retain the value.</p>
PSBS	<p>Value range: [0, 32] You are advised to retain the value.</p>
PSDS	<p>Value range: [0, 32] You are advised to retain the value.</p>
RefMode	<p>Value range: [0, 3] You are advised to retain the value.</p>
TFC	<p>Strength of the color difference temporal filtering Value range: [0, 32]</p>
SFC	<p>Strength of the color difference spatial filtering Value range: [0, 255]</p>

[Note]

You can retain the values of the parameters with the note "You are advised to retain the value." If the values of these parameters are changed, the NR effect is unpredictable.

[See Also]

- [HI_MPI_VPSS_SetNRBParam](#)
- [HI_MPI_VPSS_GetNRBParam](#)

VPSS_GRP_VPPNRBCORE_S

[Description]

Defines the information of some 3DNR parameters.

[Syntax]

```
typedef struct
```



```
{  
    HI_S8 ISO;  
    HI_U8 SFC;  
    HI_U8 TFC;  
  
    HI_U8 SHPi;  
    HI_U8 SBSi;  
    HI_U8 SBTi;  
    HI_U8 SDSi;  
    HI_U8 SDTi;  
    HI_U8 MDZi;  
  
    HI_U8 SHPj;  
    HI_U8 SBSj;  
    HI_U8 SBTj;  
    HI_U8 SDSj;  
    HI_U8 SDTj;  
    HI_U8 MDZj;  
  
    HI_U8 SHPk;  
    HI_U8 SBSk;  
    HI_U8 SBTk;  
    HI_U8 SDSk;  
    HI_U8 SDTk;  
  
    HI_U16 SBFi : 2;  
    HI_U16 SBFj : 2;  
    HI_U16 SBFk : 2;  
    HI_U16 MATH : 10;  
  
    HI_U16 TFSi : 4;  
    HI_U16 TFSj : 4;  
    HI_U16 TFSk : 4;
```



```
HI_U16 PSFS : 4;

HI_U16 TFRi : 5;
HI_U16 TFRj : 5;
HI_U16 TFRk : 5;
HI_U16 Post : 1;

} VPSS_GRP_VPPNRBCORE_S;
```

[Member]

Member	Description
ISO	Reserved
SFC	Strength of spatial filtering for the color difference component Value range: [0,255]
TFC	Strength of temporal filtering for the color difference component Value range: [0,32]
SHPi, SHPj, SHPk	Edge enhancing mode and NR mode Value range: [0, 127], [0, 127]/[0, 64], [0, 64]. For details, see the Note field.
SBSi, SBSj, SBSk	Strength of absolute spatial filtering in the bright region Value range: [0, 255], [0, 255], [0, 255]
SBTi, SBTj, SBTk	Threshold for detecting edges in the bright region of the picture through spatial filtering Value range: [0, 64], [0, 64], [0, 64]
SDSi, SDSj, SDSk	Strength of absolute spatial filtering in the dark region Value range: [0, 255], [0, 255], [0, 255]
SDTi, SDTj, SDSk	Threshold for detecting edges in the dark region of the picture through spatial filtering Value range: [0, 64], [0, 64], [0, 64]
MDZi, MDZj	Threshold value used by the 3DNR to estimate the motion degree of each pixel in the input picture and obtain the motion index of each pixel. When the motion index of a pixel is less than or equal to MDZ , this pixel is considered to belong to the static region. When the motion index of a pixel is greater than MDZ , this pixel is considered to belong to the motion region. Value range: [0, 127], [0, 127]



Member	Description
SBFi, BFj, SBFk	Strength of removing sharp noises Value range: [0, 3], [0, 3], [0, 3]
MATH	Threshold value used to obtain the motion index of each pixel. The pixel motion degree estimated by MATH is more precise than that estimated by MDZi and MDZj . When the motion index of a pixel is less than or equal to MATH , this pixel is considered to belong to the static region. When the motion index of a pixel is greater than MATH , this pixel is considered to belong to the motion region. Value range: [0, 511]
TFSi, TFSj, TFSk	Degree to which the output picture has the fewest noises Value range: [0, 15], [0, 15], [0, 15]
PSFS	Degranulation strength Value range: [0, 8]
TFRi, TFRj, TFRk	Strength of the anti-smearing mechanism. A larger value indicates weaker anti-smearing strength and greater temporal filtering strength. Value range: [0, 31], [0, 31], [0, 31]
Post	Switch of the front-end enhancing mode and back-end enhancing mode Value range: [0, 1]

[Note]

- The relationship between **SHPj** and **Post** is as follows:
 - When **Post** is **0**, the value range of **SHPj** is [0, 64].
 - When **Post** is **1**, the value range of **SHPj** is [0, 127].
 - If the value of **Post** needs to be changed, you are advised to set **SHPj** to a value less than 64 and then change the value of **Post** from **1** to **0**.
- You can retain the values of the parameters with the note "You are advised to retain the value". If the values of these parameters are changed, the NR effect is unpredictable.

[See Also]

- [VPSS_GRP_VPPNRBEX_S](#)
- [HI_MPI_VPSS_SetNRV3Param](#)
- [HI_MPI_VPSS_GetNRV3Param](#)

VPSS_GRP_VPPNRBEX_S

[Description]



Defines the information of the 3DNR parameters so that the information can be set and obtained.

[Syntax]

```
typedef struct\n{\n    VPSS_GRP_VPPNRBEX_S iNRb;\n\n    HI_U8 MDAF      : 3;\n    HI_U8 PostROW   : 5;\n    HI_U8 MATW      : 2;\n    HI_U8 ExTfThr   : 5;\n    HI_U8 MABW      : 1;\n    HI_U8 TextThr;\n    HI_U8 MTFS;\n}\nVPSS_GRP_VPPNRBEX_S;
```

[Member]

Member	Description
iNRb	Some 3DNR parameters
MDAF	Value range: [0, 7] You are advised to retain the value.
PostROW	Value range: [0, 31] You are advised to retain the value.
MATW	Value range: [0, 3] You are advised to retain the value.
ExTfThr _	Value range: [0, 31] You are advised to retain the value.
MABW	Value range: [0, 1] You are advised to retain the value.
TextThr	Value range: [0, 255] You are advised to retain the value.
MTFS	Value range: [0, 255] You are advised to retain the value.

[Note]

You can retain the values of the parameters with the note "You are advised to retain the value". If the values of these parameters are changed, the NR effect is unpredictable.



[See Also]

- [VPSS_GRP_VPPNRBCORE_S](#)
- [HI_MPI_VPSS_SetNRV3Param](#)
- [HI_MPI_VPSS_GetNRV3Param](#)

NRS_PARAM_V1_S

[Description]

Defines the parameters of the standard 3DNR interface for Hi3518E V200.

[Syntax]

```
typedef VPSS_NR_PARAM_V1_S NRS_PARAM_V1_S;
```

[Member]

See [VPSS_NR_PARAM_V1_S](#).

[Note]

None

[See Also]

- [VPSS_NR_PARAM_V1_S](#)
- [VPSS_GRP_NRS_PARAM_S](#)
- [HI_MPI_VPSS_SetGrpNRSPParam](#)
- [HI_MPI_VPSS_GetGrpNRSPParam](#)

NRB_SF_PARAM_V1_S

[Description]

Defines spatial filtering parameters of the advanced 3DNR interface for Hi3518E V200.

[Syntax]

```
typedef struct
{
    HI_U8 SDS;           /* [0, 255] */
    HI_U8 SDT;           /* [0, 64] */
    HI_U8 EDM;           /* [0, 3] */
    HI_U8 SHP;           /* [0, 32] */
    HI_U8 SBS;           /* [0, 255] */
    HI_U8 SBT;           /* [0, 64] */
    HI_U8 SFB;           /* [0, 255] */
    HI_U8 SBF;           /* [0, 3] */
} NRB_SF_PARAM_V1_S;
```

[Member]



Member	Description
SDS	Strength of absolute spatial filtering in the dark region Value range: [0, 255]
SDT	Value range: [0, 64] You are advised to retain the value.
EDM	Value range: [0, 3] You are advised to retain the value.
SHP	Edge enhancing mode and NR mode Value range: [0, 32]
SBS _	Strength of absolute spatial filtering in the bright region Value range: [0, 255]
SBT	Value range: [0, 64] You are advised to retain the value.
SFB	Value range: [0, 255] You are advised to retain the value.
SBF	Strength of removing sharp noises Value range: [0, 3]

[Note]

You can retain the values of the parameters with the note "You are advised to retain the value". If the values of these parameters are changed, the NR effect is unpredictable. For details, see the *3DNR Parameter Configuration Description*.

[See Also]

- [NRB_PARAM_V1_S](#)
- [VPSS_GRP_NRB_PARAM_S](#)
- [HI_MPI_VPSS_SetGrpNRBParam](#)
- [HI_MPI_VPSS_GetGrpNRBParam](#)

NRB_TF_PARAM_V1_S

[Description]

Defines temporal filtering parameters of the advanced 3DNR interface for Hi3518E V200.

[Syntax]

```
typedef struct
{
    HI_U8 TFR;          /*[0, 31]*/
    HI_U8 TFP;          /*[0, 31]*/
    HI_U8 TFS;          /*[0, 15]*/
}
```



```
HI_U8 MSHP;          /* [0, 16] */
HI_U8 MTFR;          /* [0, 16] */
HI_U8 MDZ;           /* [0, 127] */
} NRB_TF_PARAM_V1_S;
```

[Member]

Member	Description
TFR	Strength of the anti-smearing mechanism. A larger value indicates weaker anti-smearing strength and greater temporal filtering strength. Value range: [0, 31]
TFP	Smearing control factor. A smaller value indicates that smearing is less obvious. Value range: [0, 31] You are advised to retain the value.
TFS	Degree to which the output picture has the fewest noises Value range: [0, 15] You are advised to retain the value.
MSHP	Absolute strength of spatial filtering in the motion region. A larger value indicates lower strength. Value range: [0, 16]
MTFR_	Absolute strength of temporal filtering in the motion region. A larger value indicates higher strength. Value range: [0, 16]
MDZ	Threshold value used by the 3DNR module to estimate the motion degree of each pixel in the input picture and obtain the motion index of each pixel. When the motion index of a pixel is less than or equal to MDZ , this pixel is considered to belong to the static region. When the motion index of a pixel is greater than MDZ , this pixel is considered to belong to the motion region. Value range: [0, 127]

[Note]

You can retain the values of the parameters with the note "You are advised to retain the value". If the values of these parameters are changed, the NR effect is unpredictable. For details, see the *3DNR Parameter Configuration Description*.

[See Also]

- [NRB_PARAM_V1_S](#)
- [VPSS_GRP_NRB_PARAM_S](#)
- [HI_MPI_VPSS_SetGrpNRBParam](#)
- [HI_MPI_VPSS_GetGrpNRBParam](#)



NRB_PARAM_V1_S

[Description]

Defines the parameters of the advanced 3DNR interface for Hi3518E V200.

[Syntax]

```
typedef struct
{
    HI_S16 IES;          /* [-255, 64] */
    HI_U8 ISHT;          /* [0, 64] */
    HI_U8 SFC;           /* [0, 80] */
    HI_U8 IEX[3];        /* [0, 127] */
    HI_U8 TFC;           /* [0, 32] */

    NRB_SF_PARAM_V1_S sf[3];
    NRB_TF_PARAM_V1_S tf[2];

    HI_U16 MATH : 9;    /* [0, 511] */
    HI_U16 MDAF : 3;    /* [0, 7] */
    HI_U16 MATW : 3;    /* [0, 5] */
    HI_U16 MABW : 1;    /* [0, 1] */

    HI_U8 PSFS;          /* [0, 8] */

} NRB_PARAM_V1_S;
```

[Member]

Member	Description
IES	Edge sharpening strength Value range: [-255, +64]
ISHT	Parameter that works with IES . When IES is greater than 0 , ISHT is set to 32 . When IES is less than or equal to 0 , ISHT is set to 0 . Value range: [0, 64] You are advised to retain the value.
SFC	Strength of spatial filtering for the color difference component Value range: [0, 80]
IEX	Value range: [0, 127] You are advised to retain the value.
TFC	Strength of temporal filtering for the color difference component Value range: [0, 32]
sf	Multi-level spatial filtering parameter



Member	Description
tf	Multi-level temporal filtering parameter
MATH	Threshold value used to obtain the motion index of each pixel. The pixel motion degree estimated by MATH is more precise than that estimated by MDZ . When the motion index of a pixel is less than or equal to MATH , this pixel is considered to belong to the static region. When the motion index of a pixel is greater than MATH , this pixel is considered to belong to the motion region. Value range: [0, 511]
MDAF	Value range: [0, 7] You are advised to retain the value.
MATW	Size of the motion estimation time window Value range: [0, 5] You are advised to retain the value.
MABW	Size of the motion estimation space window Value range: [0, 1] You are advised to retain the value.
PSFS	Degranulation strength Value range: [0, 8] You are advised to retain the value.

[Note]

You can retain the values of the parameters with the note "You are advised to retain the value". If the values of these parameters are changed, the NR effect is unpredictable. For details, see the *3DNR Parameter Configuration Description*.

[See Also]

- [HI_MPI_VPSS_SetGrpNRBParam](#)
- [HI_MPI_VPSS_GetGrpNRBParam](#)

NRB_PARAM_V2_S

[Description]

Defines the parameters of the advanced 3DNR interface for Hi3519 V100.

[Syntax]

```
typedef VPSS_GRP_VPPNRB_S NRB_PARAM_V2_S;
```

[Member]

See [VPSS_GRP_VPPNRB_S](#).

[Note]

None



[See Also]

- [VPSS_GRP_VPPNRB_S](#)
- [VPSS_GRP_NRB_PARAM_S](#)
- [HI_MPI_VPSS_SetGrpNRBParam](#)
- [HI_MPI_VPSS_GetGrpNRBParam](#)

VPSS_NR_VER_E

[Description]

Defines the version of the 3DNR interface.

[Syntax]

```
typedef enum hiVPSS_NR_VER_E
{
    VPSS_NR_V1 = 1,
    VPSS_NR_V2 = 2,
    VPSS_NR_V3 = 3,
    VPSS_NR_BUTT
} VPSS_NR_VER_E;
```

[Member]

Member	Description
VPSS_NR_V1	Version 1
VPSS_NR_V2	Version 2
VPSS_NR_V3	Version 3

[Note]

None

[See Also]

- [VPSS_GRP_NRS_PARAM_S](#)
- [VPSS_GRP_NRB_PARAM_S](#)

VPSS_GRP_NRS_PARAM_S

[Description]

Defines the parameters of the standard 3DNR interface.

[Syntax]

```
typedef struct hiVPSS_GRP_NRS_PARAM_S
{
    VPSS_NR_VER_E enNRVer;
    union
```



```
{  
    NRS_PARAM_V1_S stNRSPParam_V1; /* interface S V1 for Hi3518EV200 */  
};  
  
}VPSS_GRP_NRS_PARAM_S;
```

[Member]

Member	Description
enNRVer	Version of the standard 3DNR interface
stNRSPParam_V1	Parameter of standard interface version 1, which is used only by Hi3518E V200 currently

[Note]

Chip	Parameters of Interface Version 1	Parameters of Interface Version 2	Parameters of Interface Version 3
Hi3516A	Not supported	Not supported	Not supported
Hi3518E V200	stNRSPParam_V1	Not supported	Not supported
Hi3519 V100	Not supported	Not supported	Not supported

[See Also]

- [NRS_PARAM_V1_S](#)
- [HI_MPI_VPSS_SetGrpNRSPParam](#)
- [HI_MPI_VPSS_GetGrpNRSPParam](#)

VPSS_GRP_NRB_PARAM_S

[Description]

Defines the parameters of the advanced 3DNR interface.

[Syntax]

```
typedef struct hiVPSS_GRP_NRB_PARAM_S  
{  
    VPSS_NR_VER_E enNRVer;  
    union  
    {  
        NRB_PARAM_V1_S stNRBParam_V1; /* interface B V1 for Hi3518EV200 */  
        NRB_PARAM_V2_S stNRBParam_V2; /* interface B V2 for Hi3519V100 */  
    };  
};  
  
}VPSS_GRP_NRB_PARAM_S;
```



[Member]

Member	Description
enNRVer	Version of the advanced 3DNR interface
stNRBParam_V1	Parameter of advanced interface version 1, which is used only by Hi3518E V200 currently
stNRBParam_V2	Parameter of advanced interface version 2, which is used only by Hi3519 V100 currently

[Note]

Chip	Parameters of Interface Version 1	Parameters of Interface Version 2	Parameters of Interface Version 3
Hi3516A	Not supported	Not supported	Not supported
Hi3518E V200	stNRBParam_V1	Not supported	Not supported
Hi3519 V100	Not supported	stNRBParam_V2	Not supported

[See Also]

- [NRB_PARAM_V1_S](#)
- [NRB_PARAM_V2_S](#)
- [HI_MPI_VPSS_SetGrpNRBParam](#)
- [HI_MPI_VPSS_GetGrpNRBParam](#)

5.5 Error Codes

Table 5-3 describes the error codes of VPSS MPIS.

Table 5-3 Error codes of VPSS MPIS

Error Code	Macro Definition	Description
0xA0078001	HI_ERR_VPSS_INVALID_DEVID	The VPSS group ID is invalid.
0xA0078002	HI_ERR_VPSS_INVALID_CHNID	The VPSS channel ID is invalid.
0xA0078003	HI_ERR_VPSS_ILLEGAL_PARAM	The VPSS parameter is invalid.
0xA0078004	HI_ERR_VPSS_EXIST	A VPSS group is created.
0xA0078005	HI_ERR_VPSS_UNEXIST	No VPSS group is created.
0xA0078006	HI_ERR_VPSS_NULL_PTR	The pointer of the input parameter is null.



Error Code	Macro Definition	Description
0xA0078008	HI_ERR_VPSS_NOT_SUPPORT	The operation is not supported.
0xA0078009	HI_ERR_VPSS_NOT_PERM	The operation is forbidden.
0xA007800C	HI_ERR_VPSS_NOMEM	The memory fails to be allocated.
0xA007800D	HI_ERR_VPSS_NOBUF	The buffer pool fails to be allocated.
0xA007800E	HI_ERR_VPSS_BUF_EMPTY	The picture queue is empty.
0xA0078010	HI_ERR_VPSS_NOTREADY	The VPSS is not initialized.
0xA0078012	HI_ERR_VPSS_BUSY	The VPSS is busy.



Contents

6 VENC.....	6-1
6.1 Overview	6-1
6.2 Functions	6-2
6.2.1 Data Encoding Flowchart.....	6-2
6.2.2 VENC Channels	6-3
6.2.3 Bit Rate Control	6-3
6.2.4 Advanced Frame Skipping Reference Modes	6-4
6.2.5 Color-to-Gray.....	6-5
6.2.6 Cropping Encoding	6-5
6.2.7 ROI.....	6-6
6.2.8 Encoding Non-ROIs at a Low Frame Rate	6-7
6.2.9 JPEG Snapshot Modes	6-7
6.2.10 Intra-P Frame Refresh	6-7
6.2.11 Configuration Modes of Encoding Stream Frames	6-8
6.2.12 Configuration Modes of Encoding Stream Buffers.....	6-9
6.2.13 Memory Allocation Mode for the Reference Frame and Encoding Reconstruction Frame	6-10
6.2.14 Calculation of the VB for Encoding Frames	6-11
6.2.15 Advanced Frame Skipping Reference Mode for the Virtual I Frame	6-12
6.3 API Reference	6-13
6.4 Data Structures	6-136
6.5 Error Codes	6-229



Figures

Figure 6-1 Data encoding of the VENC	6-2
Figure 6-2 Function division of a VENC channel	6-3
Figure 6-3 Schematic diagram of advanced frame skipping reference modes	6-5
Figure 6-4 Cropping encoding.....	6-6
Figure 6-5 ROI	6-7
Figure 6-6 Configuration modes of encoding stream frames	6-9
Figure 6-7 Reference relationship when only virtual I frames are used	6-12
Figure 6-8 Reference relationship when the virtual I frames are used in 2x frame skipping reference mode..	6-12
Figure 6-9 Reference relationship when the virtual I frames are used in 4x frame skipping reference mode..	6-13
Figure 6-10 Stream packet structure	6-35
Figure 6-11 Checking whether the stream buffer is wrapped	6-43
Figure 6-12 Frame processing modes.....	6-124



Tables

Table 6-1 Encoding formats.....	6-1
Table 6-2 Memory allocation mode When the reference frame shares the memory for storing the luminance data with the encoding reconstruction frame.....	6-11
Table 6-3 Methods of calculating the value of each memory item	6-11
Table 6-4 Widths and heights of VENC channels.....	6-17
Table 6-5 Limitations on encoder attributes	6-18
Table 6-6 Public attributes of three RC modes	6-19
Table 6-7 Typical configuration of the average bit rate	6-19
Table 6-8 Search widow size	6-63
Table 6-9 Default sizes of the search window of the H.264 encoder under various resolutions for the Hi3516A/Hi3518E V200.....	6-64
Table 6-10 Default trans parameter values for different profile types	6-71
Table 6-11 Default entropy encoding parameter values for different profile types	6-75
Table 6-12 Calculation of u32RefreshLineNum.....	6-132
Table 6-13 Error codes for the VENC APIs.....	6-229



6 VENC

6.1 Overview

The video encoder (VENC) supports real-time encoding on multiple channels. Each VENC channel is independent. The encoding protocols and encoding profiles of VENC channels may be different. The VENC can also schedule the region module to overlay and cover the contents of the encoded pictures. The input sources of the VENC include:



CAUTION

The VENC module is supported by Hi3516A V100/Hi3518E V200/. Hi3518E V201/Hi3516C V200

- Pictures that are read in user mode.
- Pictures that are captured by the VIU and then processed by the VPSS.
- Pictures that are captured by the VIU.

Table 6-1 lists the encoding formats supported by the Hi3516A and Hi3518E V200.

Table 6-1 Encoding formats

Chip	H.264			JPEG	MOTION JPEG	H.265	MPEG-4
	Baseline Profile	Main Profile	High Profile			Main Profile	
Hi3516A	Supported	Supported	Supported	Supported	Supported	Supported	Not supported
Hi3518E V200	Supported	Supported	Supported	Supported	Supported	Not supported	Not supported

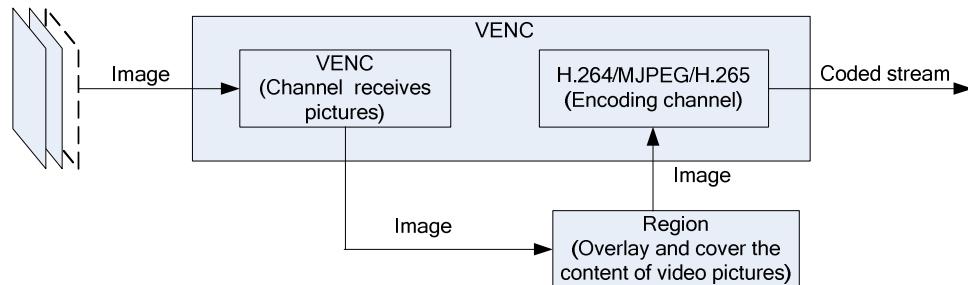


6.2 Functions

6.2.1 Data Encoding Flowchart

Figure 6-1 shows data encoding of the VENC.

Figure 6-1 Data encoding of the VENC



A typical encoding procedure includes receiving input pictures, covering and overlaying pictures, encoding pictures, and outputting streams.

The VENC consists of an encoding channel submodule and an encoding protocol submodule (H.264/H.265, JPEG, and MJPEG are supported).

The channel submodule can receive YUV pictures in the format of semi-planar YUV 4:2:0 or semi-planar YUV 4:2:2. For the H.264/H.265 protocols, only the semi-planar YUV 4:2:0 format is supported; for the JPEG and MJPEG protocols, the semi-planar YUV 4:2:0 or semi-planar YUV 4:2:2 format is supported. Hi3518E V200 supports the single-component input (only the Y component). The channel submodule receives external raw picture data regardless of the picture source.

After receiving a picture, the channel compares the picture size with the VENC channel size. The VENC performs operations in the following cases:

- If the input picture size is greater than the VENC channel size, the VENC zooms out on the picture by using the VGS to fit into the VENC channel, and then encodes the picture.
- If the input picture size is smaller than the VENC channel size, the VENC discards the picture. The VENC does not support zoom in.
- If the input picture size is the same as the VENC channel size, the VENC encodes the picture directly.

NOTE

- The frame rate control function of the channel is disabled by default. You can enable this function by calling related MPIs. The rate controller (RC) also supports frame rate control. The frame rate control function of the RC is recommended, which ensures that the bit rate does not change severely.
- For the H.264 encoding of Hi3518E V200, when the format of the input picture is changed from non-single-component format to single-component format, chrominance residue occurs on the picture before the next I frame is encoded due to the quantization error of the inter-frame prediction. You are advised to call `HI_MPI_VENC_Resetchn` to reset the channel before switching the format of the input picture from non-single-component format to single-component format.

A REGION module can cover and overlay pictures.

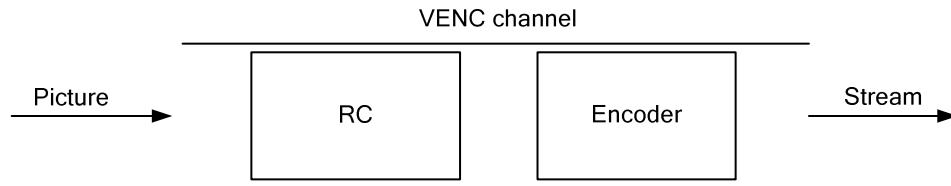
After video pre-processing and region management, the picture is sent to channel complying with a specific protocol, completing video encoding and stream output.



6.2.2 VENC Channels

A VENC channel is a basic container that stores various user configurations of VENC channels and manages the internal resources on VENC channels. A VENC channel converts pictures into streams and the conversion is completed by an RC and an encoder. The encoder completes only the encoding function; therefore, it is an encoder in narrow sense. The RC controls and adjusts the encoding parameters, controlling the output bit rate.

Figure 6-2 Function division of a VENC channel



6.2.3 Bit Rate Control

An RC mainly controls the encoding bit rate.

From the aspect of informatics, lower compression rate of a picture obtains higher picture quality. In actual scenarios, when the picture quality is stable, the encoding bit rate may fluctuate; when the encoding bit rate is stable, the picture quality may change. The following is an example using H.264 encoding. The protocol evaluates the picture quality based on the quantizer parameter (QP). Typically, when the QP value becomes small, the picture quality is improved and the encoding bit rate increases; when the QP value becomes large, the picture quality deteriorates and the encoding bit rate decreases.

Rate control must be specific to consecutive encoding streams. Therefore, the JPEG channel does not support bit rate control.

The RC provides four modes for the H.264/H.265 channel to adjust the picture quality and bit rate:

- Constant bit rate (CBR) mode
- Variable bit rate (VBR) mode
- Adaptive variable bit rate (AVBR) mode
- FixQp mode

The MJPEG VENC channel supports the CBR, VBR, and FixQp modes to adjust the picture quality and bit rate.

Hi3518E V200 does not support H.265 encoding. Therefore, it cannot control the bit rate of H.265 streams.

CBR

CBR means that a stable encoding bit rate is ensured within the bit rate statistical time. A stable bit rate is evaluated by the following two indexes which can be specified by users when a VENC channel is created.

- Bit rate statistical time (**u32StatTime**)



The statistical time unit is second. A longer statistical time indicates that the impact on bit rate adjustment exerted by the bit rate change of each frame is less, the bit rate is adjusted more slowly, and the picture quality changes more slightly.

- Adjustment amplitude of row-level bit rate control (**u32RowQpDelta**)

u32RowQpDelta indicates the maximum row-level adjustment range within one frame. The unit is the macroblock row. A larger adjustment amplitude indicates that the allowed QP row-level adjustment range is larger and the bit rate is more stable. In the scenarios where the picture complexity is unevenly distributed, the picture quality may be different in different regions if **u32RowQpDelta** is too large.

VBR

VBR means that the encoding bit rate can fluctuate within the bit rate statistical period. In this way, stable encoding picture quality can be ensured. Taking the H.264 encoding as an example, the VENC module provides four configurable parameters (**MaxQp**, **MinQp**, **MaxBitrate**, and **ChangePos**). **MaxQp** and **MinQp** are used to control the picture quality range, **MaxBitrate** is used to limit the maximum encoding bit rate within the bit rate statistical period, and **ChangePos** is used to control the bit rate base line for starting QP adjustment. When the encoding bit rate is greater than (**MaxBitrate** x **ChangePos**), the picture QP value is adjusted towards **MaxQp**. If the QP value reaches **MaxQp**, the QP value is fixed at the maximum value and **MaxBitrate** does not take effect. In this case, the encoding bit rate may exceed **MaxBitrate**. When the encoding bit rate is less than (**MaxBitrate** x **ChangePos**), the picture QP value is adjusted towards **MinQp**. If the QP value reaches **MinQp**, the encoding bit rate reaches the maximum value and the picture quality is the best.

AVBR

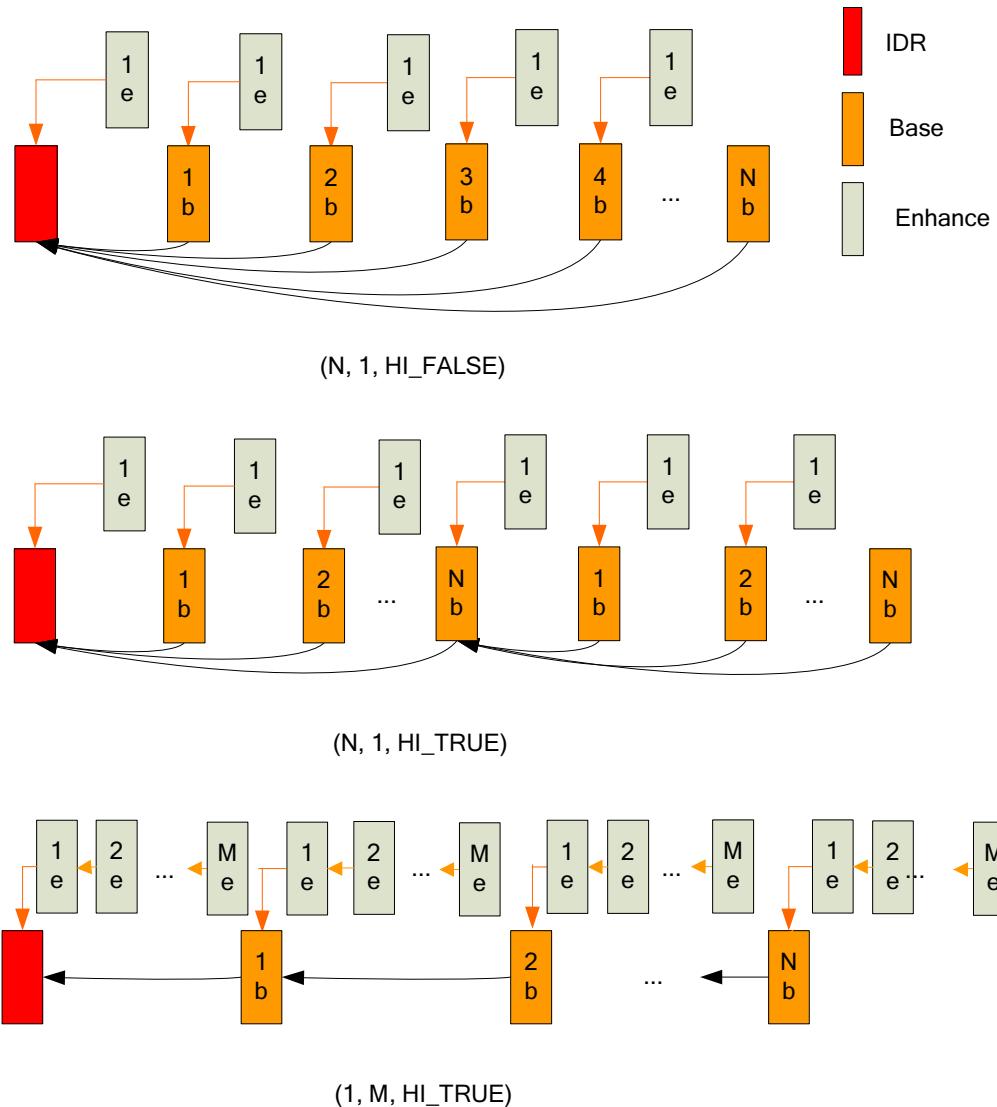
AVBR means that the encoding bit rate can fluctuate within the bit rate statistical period. In this way, stable encoding picture quality can be ensured. The RC detects the motion status of the current scenario, and improves the encoding bit rate in the motion scenario and reduces the target bit rate in the static scenario. Taking the H.264 encoding as an example, the VENC module provides three configurable parameters: **MaxBitrate**, **ChangePos**, and **MinStillPercent**. **MaxBitrate** indicates the maximum bit rate in the motion scenario. **MaxBitrate * ChangePos * MinStillPercent** indicates the minimum bit rate in the static scenario. The target bit rate is adjusted between the maximum bit rate and the minimum bit rate based on the motion degree. **MaxQp** and **MinQp** are used to control the picture quality range. The bit rate control takes the QP clamping as the highest priority, and bit rate control becomes invalid if the range of [MinQp, MaxQp] is exceeded.

FixQp

FixQp is a fixed QP value. Within the bit rate statistics, FixQp indicates that the QP values of all the macroblocks of encoded pictures are the same, the user-defined QP value is used, and the QP values of I frame and P frame can be set separately.

6.2.4 Advanced Frame Skipping Reference Modes

The **u32Base**, **u32Enhance**, and **bEnablePred** parameters are involved in advanced frame skipping reference modes. For details about these parameters, see the description of [VENC_PARAM_REF_S](#). [Figure 6-3](#) shows the schematic diagram of advanced frame skipping reference modes.

Figure 6-3 Schematic diagram of advanced frame skipping reference modes

6.2.5 Color-to-Gray

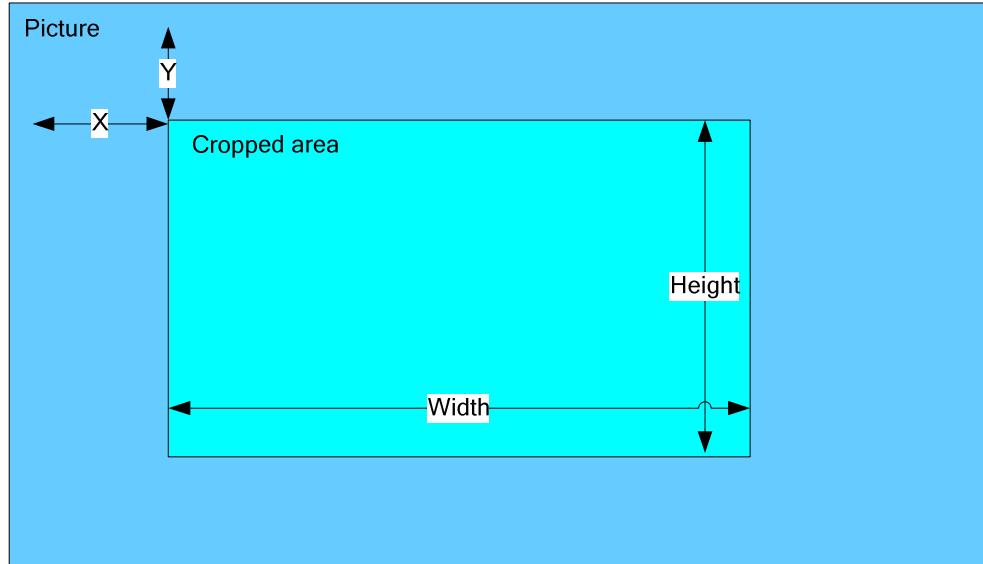
The color-to-gray function enables the VENC to convert color pictures into gray pictures for encoding. For details, see the description of [HI_MPI_VENC_SetColor2Grey](#).

6.2.6 Cropping Encoding

Cropping encoding indicates that the VENC crops a part of a picture for encoding. You can set the start position (X and Y), width, and height. For details, see the description of [HI_MPI_VENC_GetCrop](#) and [HI_MPI_VENC_SetCrop](#).



Figure 6-4 Cropping encoding



6.2.7 ROI

You can set a region of interest (ROI) to control the picture QP value of the region, distinguishing this region from other regions. The system allows you to set the ROI only for the H.264/H.265 channel and provides eight ROIs.

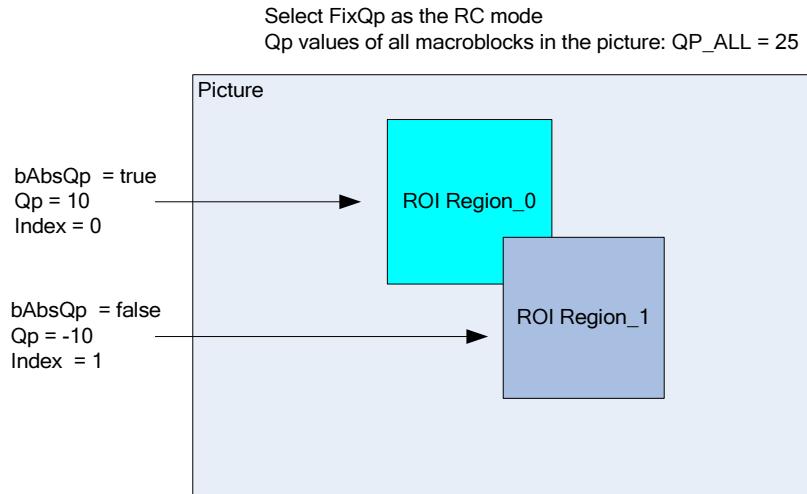
The eight regions can be overlaid with each other and the overlay priority ranges from 0 to 7. The overlay priority indicates the final QP value of picture regions when the regions are overlaid. The final QP value is set according to the region with the highest priority. An ROI can be configured with an absolute QP and a relative QP.

- Absolute QP: indicates the QP value of an ROI set by users.
- Relative QP: indicates the QP value generated during bit rate control plus the offset of the QP values set by users.

[Figure 6-5](#) shows an encoded picture in FixQp mode. The picture QP value is 25, that is, the QP values of all macroblocks of the picture are 25. ROI 0 is configured with absolute QP mode, the QP value is 10, and the index is 0; ROI 1 is configured with relative QP mode, the QP value is -10, and the index is 1. When the two regions are overlaid, the QP is set according to the region with the highest priority (ROI 1). Except the overlaid part, the QP of ROI 0 is 10 and therefore the QP of ROI 1 is 15 ($25 - 10$).



Figure 6-5 ROI



6.2.8 Encoding Non-ROIs at a Low Frame Rate

Encoding non-ROIs at a low frame rate indicates encoding ROIs at a normal frame rate. You can set the frame rate of non-ROIs as required. For details, see the description of [HI_MPI_VENC_SetRoiBgFrameRate](#) and [HI_MPI_VENC_GetRoiBgFrameRate](#).

6.2.9 JPEG Snapshot Modes

The JPEG VENC channels support two modes: snapshot all mode and flash snapshot mode.

- Snapshot all mode: After the channels start to receive pictures, all received pictures are encoded.
- Flash snapshot mode: After the channels start to receive channels, only the pictures captured when the camera flash is on are encoded.

6.2.10 Intra-P Frame Refresh

By refreshing the Islice/intra-macroblock rows in the P frame, smooth stream encoding is ensured and the size of an I frame can be close to that of a P frame. When the network bandwidth is limited (such as the wireless network), the intra-P frame refresh can reduce the heavy network traffic caused by large I frames, reduce network transmission delay, and decrease the probability of network transmission errors.

6.2.10.1 Usage

To use the intra-P frame refresh function, perform the following steps:

- Step 1** Configure the refresh enable (**bRefreshEnable**) and Islice conversion enable (**bISliceEnable**) by calling [HI_MPI_VENC_SetIntraRefresh](#).
- After the refresh function is enabled, the SDK automatically implements intra-macroblock refresh from the first frame of each group of pictures (GOP) from top to bottom based on the configured number of rows to be refreshed (**u32RefreshLineNum**). The refresh interval is one GOP cycle.



- After Islice conversion is enabled, the SDK automatically converts the intra-frame prediction macroblock in the first frame to be refreshed into an Islice. If the Islice is refreshed, the stream compatibility is better, whereas the CPU usage is higher and more memory space needs to be allocated (for internal conversion; the size is twice the stream buffer size).

Step 2 Adjust **VENC_RC_PARAM_S: stParamH264Cbr.s32IPQPDelta** to control the size of the start frame of I macroblock refresh (the advanced bit rate parameters still take effect after [HI_MPI_VENC_SetIntraRefresh](#) is called). The recommended value is -1.

----End

6.2.10.2 Precautions

- The dynamic library **libslice_trans.so** is required for Islice conversion. You need to copy this library to the corresponding dynamic library directory.
- The number of rows to be refreshed (**u32RefreshLineNum**) must be set to an appropriate value by calling [HI_MPI_VENC_SetIntraRefresh](#) to ensure that all the macroblock rows in the picture are refreshed once within one GOP cycle.
- The number of rows to be refreshed (**u32RefreshLineNum**) is related to the encoding parameter GOP and frame skipping reference parameters. Therefore, after the encoding attributes and advanced frame skipping reference parameters are configured by calling the corresponding MPIs, [HI_MPI_VENC_SetIntraRefresh](#) needs to be called to reconfigure **u32RefreshLineNum**.
- [HI_MPI_VENC_SetIntraRefresh](#) is valid only for Hi3516A V100 and Hi3518E V200.
- [HI_MPI_VENC_SetIntraRefresh](#) is valid only for the H.264 protocol.
- If the refresh function (**bISliceEnable**) is enabled by calling [HI_MPI_VENC_SetIntraRefresh](#), the encoding streams must be obtained by frame. Islice conversion is not supported when streams are obtained by packet.
- If the refresh function (**bISliceEnable**) is enabled by calling [HI_MPI_VENC_SetIntraRefresh](#), the frame in which the Islice is refreshed automatically splits slices based on the size of the refreshed Islice. In this case, the configured slice split MPI does not take effect.

6.2.11 Configuration Modes of Encoding Stream Frames

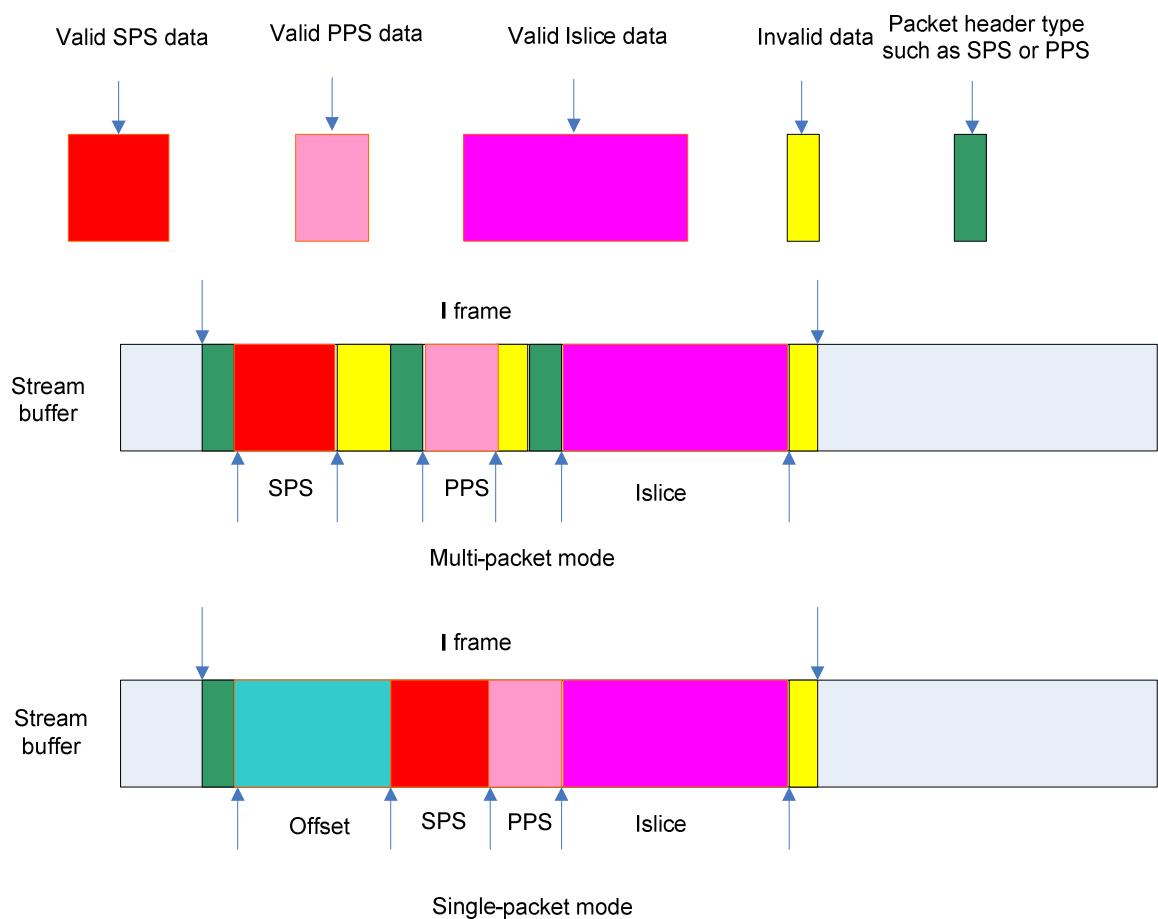
The encoding stream frames can be configured in single-packet mode and multi-packet mode (when the slice split interface and the user data insertion interface are not called), as shown in [Figure 6-6](#).

- Multi-packet mode: For H.264 encoding, when frames are I frames, [HI_MPI_VENC_GetStream](#) is called. An I frame contains four network abstraction layer (NAL) packets, including a sequence parameter set (SPS) packet, a picture parameter set (PPS) packet, an SEI packet, and an Islice packet (assuming that there is only one PPS packet and the four NAL packets are independent and differ in the packet type). For JPEG encoding, a frame of picture contains two packets, including a picture parameter packet and a picture data packet (the two packets are independent and differ in the packet type).
- Single-packet mode: For H.264 encoding, when frames are I frames, [HI_MPI_VENC_GetStream](#) is called. An I frame contains one Islice-type NAL packet (the packet contains data of SPS, PPS, SEI, and Islice). For JPEG encoding, a frame of picture contains only one picture data packet (the packet contains data of the picture parameter packet).

The configuration mode can be selected by setting the **OneStreamBuffer** parameter when .ko drivers are loaded. If **OneStreamBuffer** is 1, the single-packet mode is selected. If **OneStreamBuffer** is 0, the multi-packet mode is selected. The default value of **OneStreamBuffer** is 0.

If the packet split interface such as `HI_MPI_VENC_SetH264SliceSplit` is called, a frame is split into multiple slices. If the single-packet mode is selected, the first Islice packet of an I frame contains data of SPS, PPS, and SEI, but other Islice packets of this I frame do not contain such data. For H.264 encoding, data of SPS, PPS, and SEI exists only in the first Islice packet of an I frame and merges into one Islice packet. For JPEG/MJPEG encoding, data of the picture parameter packet exists only in the first data packet of a frame and merges into one data packet.

Figure 6-6 Configuration modes of encoding stream frames



6.2.12 Configuration Modes of Encoding Stream Buffers

The encoding stream buffer supports two configuration modes: normal mode and memory reduction mode.

- **Normal mode:** In consideration of jumbo frames, the minimum size of the stream buffer is (Channel width x Channel height x 3/4) for H.264 and H.265 encoding and (Channel width x Channel height) for JPEG and MJPEG encoding.



- Memory reduction mode: The minimum size of the stream buffer is (32 x 1024) bytes. The stream buffer size must be set to an appropriate value in this mode. Otherwise, frames may be re-encoded continuously or discarded due to the insufficiency of the stream buffer.

The mode can be specified by setting corresponding module parameters when .ko drivers are loaded. If the parameter value is 1, the memory reduction mode is used; if the parameter value is 0, the normal mode is used.

- Module parameter of **hi35xx_h264e.ko**: **H264eMiniBufMode**
- Module parameter of **hi35xx_h265e.ko**: **H265eMiniBufMode**
- Module parameter of **hi35xx_jpege.ko**: **JpegeMiniBufMode**

6.2.13 Memory Allocation Mode for the Reference Frame and Encoding Reconstruction Frame

The reference frame can share the memory for storing the luminance data with the encoding reconstruction frame. The memory for storing the chrominance data needs to be allocated for the reconstruction frame and reference frame separately. The memory allocation mode for the reference frame and reconstruction frame can be specified by setting **H264eRcnEqualRef** when **h264e.ko** is loaded. If **H264eRcnEqualRef** is set to **0**, the memory is allocated separately for the reference frame and reconstruction frame by default. If **H264eRcnEqualRef** is set to **1**, the reference frame shares the memory for storing the luminance data with the reconstruction frame.

When the reference frame shares the memory for storing the luminance data with the reconstruction frame, the following issues occur:

- The reference frame is overwritten by the reconstruction frame during encoding. Therefore, only I frames can be inserted when the jumbo frame appears, bit rate overshoot occurs, or the stream buffer is full.
- One more memory for storing one frame is required in 2x frame skipping reference mode.

When the reference frame shares the memory for storing the luminance data with the encoding reconstruction frame, the memory for storing the chrominance data of the encoding reference frame is allocated in VB mode and is calculated as follows: CSize = align(MaxPicWidth,16) x align(MaxPicHeight,16)/2. The memory for storing the luminance data of the reference frame is allocated from the media memory zone (MMZ) and is calculated as follows:

- When the reference frame is compressed:
BlkSize = HeadSize + YSize. HeadSize and YSize are calculated as follows:
HeadSize = (align(MaxPicWidth,16) + 255)/256 x align(MaxPicHeight,16) x 2
YSize = (align(MaxPicWidth,16) + 48) x align(MaxPicHeight,16)
- When the reference frame is uncompressed:
BlkSize = YSize. YSize is calculated as follows:
YSize = align(MaxPicWidth,16) x align(MaxPicHeight,16)



Table 6-2 Memory allocation mode When the reference frame shares the memory for storing the luminance data with the encoding reconstruction frame

H264eRcnEqualRef	Required Memory Size
1	When the reference frame is compressed: BlkSize = HeadSize + YSize + 2 x CSize When the reference frame is uncompressed: BlkSize = Ysize +2 x CSize
0	When the reference frame is compressed: BlkSize =2 x (HeadSize + YSize + CSize) When the reference frame is uncompressed: BlkSize = 2 x (Ysize + CSize)

NOTE

Currently only the reference frame compression mode is supported.

6.2.14 Calculation of the VB for Encoding Frames

The size of each VB for encoding frames (reference frames and reconstruction frames) is calculated as follows:

$$\text{FrameSize} = \text{YHeaderSize} + \text{CHeaderSize} + \text{YSize} + \text{CSize} + \text{PmeSize} + \text{TmvSize}$$

Table 6-3 Methods of calculating the value of each memory item

Memory Item	H.264		H.265
YHeaderSize	Compression	$(\text{align}(\text{width},256) \gg 8) * (\text{align}(\text{height},16) \gg 4) * 32$	$(\text{align}(\text{width},64) \gg 6) * (\text{align}(\text{height},64) \gg 6) * 32$
	Non-compression	0	0
YSize	Compression	$((\text{align}(\text{width},16) \gg 4) - 1) \gg 2 + 1 * ((\text{align}(\text{width},16) \gg 4) - 1) + 1 * 4 * 256$	$\text{align}(\text{width},64) * \text{align}(\text{height},64)$
	Non-compression	$\text{align}(\text{width},16) * \text{align}(\text{height},16)$	$\text{align}(\text{width},16) * \text{align}(\text{height},16)$
CHeaderSize	$\text{align}(\text{width},16) * \text{align}(\text{height},16)$		YHeaderSize
CSize	$\text{YSize}/2$		$\text{YSize}/2$
PmeSize	0		$(\text{align}(\text{width},64) \gg 6) * (\text{align}(\text{height},64) \gg 6) \ll 8$
TmvSize	0		$(\text{align}(\text{width},64) \gg 6) * (\text{align}(\text{height},64) \gg 6) \ll 6$

 NOTE

- Currently only the reference frame compression mode is supported.
- If the PrivateVB mode is used for encoding, the encoder uses **MaxFrameSize** to allocate the VBs to the reference frames and reconstruction frames, which is obtained by substituting **MaxPicWidth** and **MaxPicWidth** into the formula for calculating the VB size.

6.2.15 Advanced Frame Skipping Reference Mode for the Virtual I Frame

The virtual I frame is added based on the advanced frame skipping reference mode. The virtual I frame is inserted at a fixed interval. The virtual I frame refers only to the IDR frame. For details, see [VENC_PARAM_REF_EX_S](#).

[Figure 6-7](#) describes the reference relationship when only virtual I frames are used. [Figure 6-8](#) describes the reference relationship when the virtual I frames are used in 2x frame skipping reference mode. [Figure 6-9](#) describes the reference relationship when the virtual I frames are used in 4x frame skipping reference mode.

Figure 6-7 Reference relationship when only virtual I frames are used

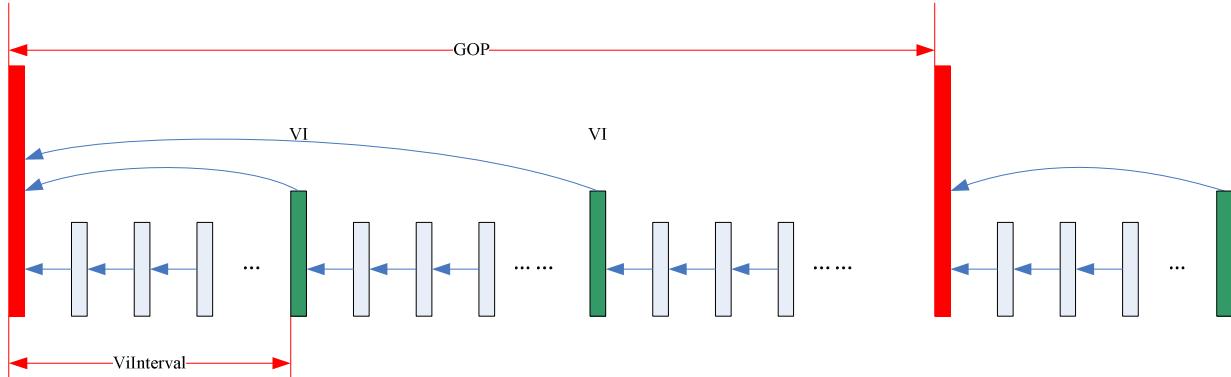


Figure 6-8 Reference relationship when the virtual I frames are used in 2x frame skipping reference mode

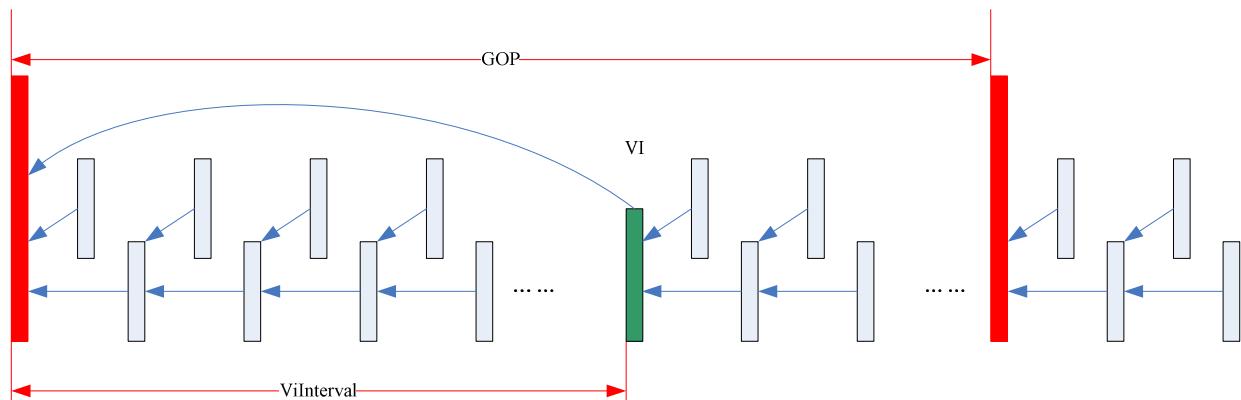
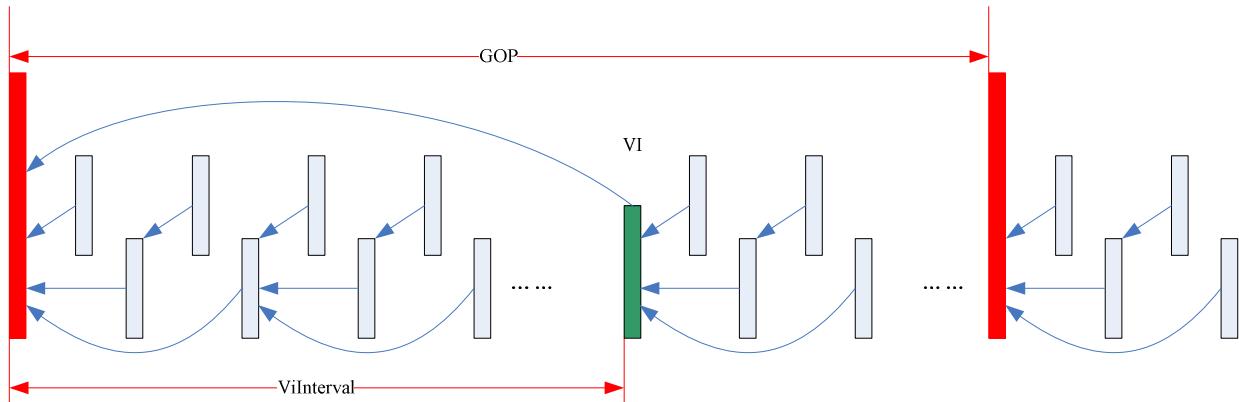




Figure 6-9 Reference relationship when the virtual I frames are used in 4x frame skipping reference mode



6.3 API Reference

The VENC has the following functions:

- Creates and destroys a VENC channel.
- Resets a VENC channel.
- Starts or stops picture reception.
- Sets and obtains VENC channel attributes.
- Obtains and releases streams.

The VENC provides the following MPIs:

- [HI_MPI_VENC_CreateChn](#): Creates a VENC channel.
- [HI_MPI_VENC_DestroyChn](#): Destroys a VENC channel.
- [HI_MPI_VENC_Resetchn](#): Resets a VENC channel.
- [HI_MPI_VENC_StartRecvPic](#): Enables a VENC channel to start receiving input pictures.
- [HI_MPI_VENC_StartRecvPicEx](#): Enables a VENC channel to start receiving input pictures and automatically stop the VENC channel receiving input pictures after the specified number of frames is exceeded.
- [HI_MPI_VENC_StopRecvPic](#): Stops a VENC channel receiving input pictures.
- [HI_MPI_VENC_Query](#): Queries the status of a VENC channel.
- [HI_MPI_VENC_SetChnAttr](#): Sets attributes of a VENC channel.
- [HI_MPI_VENC_GetChnAttr](#): Obtains attributes of a VENC channel.
- [HI_MPI_VENC_GetStream](#): Obtains encoded streams.
- [HI_MPI_VENC_ReleaseStream](#): Releases a stream buffer.
- [HI_MPI_VENC_GetStreamBufInfo](#): Obtains the physical address and size of a stream buffer.
- [HI_MPI_VENC_InsertUserData](#): Inserts user data.
- [HI_MPI_VENC_SendFrame](#): Sends raw pictures for encoding.



- [**HI_MPI_VENC_SetMaxStreamCnt**](#): Sets the maximum number of frames in a stream buffer.
- [**HI_MPI_VENC_GetMaxStreamCnt**](#): Obtains the maximum number of frames in a stream buffer.
- [**HI_MPI_VENC_RequestIDR**](#): Requests an intermediate data rate (IDR) frame.
- [**HI_MPI_VENC_EnableIDR**](#): Enables an IDR frame.
- [**HI_MPI_VENC_SetH264IdrPicId**](#): Sets the idr_pic_id of an IDR frame.
- [**HI_MPI_VENC_GetH264IdrPicId**](#): Obtains the idr_pic_id of an IDR frame.
- [**HI_MPI_VENC_GetFd**](#): Obtains the device file handle of a VENC channel.
- [**HI_MPI_VENC_CloseFd**](#): Closes the device file handle of a VENC channel.
- [**HI_MPI_VENC_SetRoiCfg**](#): Sets the ROI configuration of a channel.
- [**HI_MPI_VENC_GetRoiCfg**](#): Obtains the ROI configuration of a channel.
- [**HI_MPI_VENC_SetRoiBgFrameRate**](#): Sets the frame rate attribute of non-ROIs in a VENC channel.
- [**HI_MPI_VENC_GetRoiBgFrameRate**](#): Obtains the frame rate attribute of non-ROIs in a VENC channel.
- [**HI_MPI_VENC_SetH264SliceSplit**](#): Sets the slice attribute of an H.264 channel.
- [**HI_MPI_VENC_GetH264SliceSplit**](#): Obtains the slice attribute of an H.264 channel.
- [**HI_MPI_VENC_SetH264InterPred**](#): Sets the inter-prediction attribute of an H.264 channel.
- [**HI_MPI_VENC_GetH264InterPred**](#): Obtains the inter-prediction attribute of an H.264 channel.
- [**HI_MPI_VENC_SetH264IntraPred**](#): Sets the intra-prediction attribute of an H.264 channel.
- [**HI_MPI_VENC_GetH264IntraPred**](#): Obtains the intra-prediction attribute of an H.264 channel.
- [**HI_MPI_VENC_SetH264Trans**](#): Sets the transformation and quantization attribute of an H.264 channel.
- [**HI_MPI_VENC_GetH264Trans**](#): Obtains the transformation and quantization attribute of an H.264 channel.
- [**HI_MPI_VENC_SetH264Entropy**](#): Sets the entropy encoding mode for an H.264 channel.
- [**HI_MPI_VENC_GetH264Entropy**](#): Obtains the entropy encoding mode for an H.264 channel.
- [**HI_MPI_VENC_SetH264Poc**](#): Sets the POC type of an H.264 channel.
- [**HI_MPI_VENC_GetH264Poc**](#): Obtains the POC type of an H.264 channel.
- [**HI_MPI_VENC_SetH264Dblk**](#): Sets the de-blocking type of an H.264 channel.
- [**HI_MPI_VENC_GetH264Dblk**](#): Obtains the de-blocking type of an H.264 channel.
- [**HI_MPI_VENC_SetH264Vui**](#): Sets the video usability information (VUI) parameters of an H.264 channel.
- [**HI_MPI_VENC_GetH264Vui**](#): Obtains the VUI parameter settings of an H.264 channel.
- [**HI_MPI_VENC_SetH265Vui**](#): Sets the VUI parameters of an H.265 VENC channel.
- [**HI_MPI_VENC_GetH265Vui**](#): Obtains the VUI parameter settings of an H.265 VENC channel.
- [**HI_MPI_VENC_SetJpegParam**](#): Sets the parameter set of a JPEG channel.



- [**HI_MPI_VENC_GetJpegParam**](#): Obtains the parameter set of a JPEG channel.
- [**HI_MPI_VENC_SetMjpegParam**](#): Sets the advanced parameters of an MJPEG channel.
- [**HI_MPI_VENC_GetMjpegParam**](#): Obtains the configurations of the advanced parameters of an MJPEG channel.
- [**HI_MPI_VENC_SetFrameRate**](#): Sets the frame rate control attribute of a VENC channel.
- [**HI_MPI_VENC_GetFrameRate**](#): Obtains the frame rate control attribute of a VENC channel.
- [**HI_MPI_VENC_SetRcParam**](#): Sets the RC advanced parameters of a VENC channel.
- [**HI_MPI_VENC_GetRcParam**](#): Obtains the configurations of the advanced parameters for the RC of a VENC channel.
- [**HI_MPI_VENC_SetRefParam**](#): Sets the Advanced frame skipping reference parameters for an H.264/H.265 VENC channel
- [**HI_MPI_VENC_GetRefParam**](#): Obtains the Advanced frame skipping reference parameters for an H.264/H.265 VENC channel
- [**HI_MPI_VENC_SetColor2Grey**](#): Enables or disables the color-to-gray function of a VENC channel.
- [**HI_MPI_VENC_GetColor2Grey**](#): Obtains the enable status of the color-to-gray function of a VENC channel.
- [**HI_MPI_VENC_SetCrop**](#): Sets the cropping attribute of a VENC channel.
- [**HI_MPI_VENC_GetCrop**](#): Obtains the cropping attribute of a VENC channel.
- [**HI_MPI_VENC_SetJpegSnapMode**](#): Sets the snapshot mode of a JPEG snapshot channel.
- [**HI_MPI_VENC_GetJpegSnapMode**](#): Obtains the snapshot mode of a JPEG snapshot channel.
- [**HI_MPI_VENC_SetH265SliceSplit**](#): Sets the slice split attribute for H.265 encoding.
- [**HI_MPI_VENC_GetH265SliceSplit**](#): Obtains the slice split attribute for H.265 encoding.
- [**HI_MPI_VENC_SetH265PredUnit**](#): Sets the prediction unit (PU) attribute for H.265 encoding.
- [**HI_MPI_VENC_GetH265PredUnit**](#): Obtains the PU attribute for H.265 encoding.
- [**HI_MPI_VENC_SetH265Trans**](#): Sets the transformation and quantization attribute for H.265 encoding.
- [**HI_MPI_VENC_GetH265Trans**](#): Obtains the transformation and quantization attribute for H.265 encoding.
- [**HI_MPI_VENC_SetH265Entropy**](#): Sets the entropy encoding attribute of an H.265 channel.
- [**HI_MPI_VENC_GetH265Entropy**](#): Obtains the entropy encoding attribute of an H.265 channel.
- [**HI_MPI_VENC_SetH265Sao**](#): Sets the sample adaptive offset (SAO) attribute for H.265 encoding.
- [**HI_MPI_VENC_GetH265Sao**](#): Obtains the SAO attribute for H.265 encoding.
- [**HI_MPI_VENC_SetH265Dbblk**](#): Sets the deblocking attribute for H.265 encoding.
- [**HI_MPI_VENC_GetH265Dbblk**](#): Obtains the deblocking attribute for H.265 encoding.
- [**HI_MPI_VENC_SetH265Timing**](#): Sets the timing attribute of VPS packets for H.265 encoding.
- [**HI_MPI_VENC_GetH265Timing**](#): Obtains the timing attribute of VPS packets for H.265 encoding.



- [HI_MPI_VENC_SetFrameLostStrategy](#): Sets the frame discarding polices when the instantaneous bit rate is above the threshold.
- [HI_MPI_VENC_GetFrameLostStrategy](#): Obtains the frame discarding polices when the instantaneous bit rate is above the threshold.
- [HI_MPI_VENC_SetSuperFrameCfg](#): Sets the processing mode of jumbo frames for the encoding channel.
- [HI_MPI_VENC_GetSuperFrameCfg](#): Obtains the processing mode of jumbo frames for the encoding channel.
- [HI_MPI_VENC_SetChnlPriority](#): Sets the channel priority.
- [HI_MPI_VENC_GetChnlPriority](#): Obtains the channel priority.
- [HI_MPI_VENC_SetRcPriority](#): Sets the RC priority.
- [HI_MPI_VENC_GetRcPriority](#): Obtains the RC priority.
- [HI_MPI_VENC_GetIntraRefresh](#): Obtains the parameter for refreshing I macroblocks.
- [HI_MPI_VENC_SetIntraRefresh](#): Sets the parameter for refreshing I macroblocks.
- [HI_MPI_VENC_SetRefParamEx](#): Sets the advanced frame skipping reference parameters for an H.264/H.265 encoding channel with virtual I frames.
- [HI_MPI_VENC_GetRefParamEx](#): Obtains the advanced frame skipping reference parameters for an H.264/H.265 encoding channel with virtual I frames.
- [HI_MPI_VENC_SetModParam](#): Sets the module parameters related to encoding.
- [HI_MPI_VENC_GetModParam](#): Obtains the module parameters related to encoding.

HI_MPI_VENC_CreateChn

[Description]

Creates a VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_CreateChn(VENC_CHN *VeChn, const VENC_CHN_ATTR_S *pstAttr);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstAttr	Pointer to the VENC channel attributes	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."



[Requirement]

- Header files: hi_comm_venc.h, hi_comm_rc.h, mpi_venc.h
- Library file: libmpi.a

[Difference]

The widths and heights of VENC channels vary depending on chip type. See [Table 6-4](#).

Table 6-4 Widths and heights of VENC channels

Chip	Width or Height (Pixel)	H.264	JPEG	MJPEG	H.265
Hi3516A	MIN_WIDTH	160	32	32	128
	MAX_WIDTH	2592	8192	8192	2592
	MIN_HEIGHT	64	32	32	128
	MAX_HEIGHT	2592	8192	8192	2592
	MIN_ALIGN	2	4	4	2
Hi3518E V200	MIN_WIDTH	160	32	32	Not supported
	MAX_WIDTH	1920	8192	8192	Not supported
	MIN_HEIGHT	64	32	32	Not supported
	MAX_HEIGHT	1920	8192	8192	Not supported
	MIN_ALIGN	2	4	4	Not supported

[Note]

- The attributes of a VENC channel include the attributes of the encoder and RC attributes.
- To set the attributes of the encoder, select encoding protocols and set attributes of each protocol.
- The maximum width and height defined in the encoder attribute and the channel width and height must meet the following conditions:
 - MaxPicWidth \in [MIN_WIDTH, MAX_WIDTH]
 - MaxPicHeight \in [MIN_HEIGHT, MAX_HEIGHT]
 - PicWidth \in [MIN_WIDTH, MAX_WIDTH]
 - PicHeight \in [MIN_HEIGHT, MAX_HEIGHT]
 - The maximum width, maximum height, channel width, or channel height must be an integral multiple of MIN_ALIGN.
- MIN_WIDTH, MAX_WIDTH, MIN_HEIGHT, and MAX_HEIGHT indicate the minimum width, maximum width, minimum height, and maximum height of a VENC channel respectively. MIN_ALIGN indicates the minimum alignment unit (in pixel).
- The encoder attributes including the VENC buffer depth and mode of obtaining streams must be set. [Table 6-5](#) describes the limitations on encoder attributes based on protocols.



NOTE

Before calculating the depths of the MJPEG and JPEG buffers, ensure that **MaxPicwidth** and **MaxPicheight** are 16-bit-aligned.

Table 6-5 Limitations on encoder attributes

Encoding Protocol	Encoding Mode	Depth of the Stream Buffer	Stream Acquisition Mode	Encoding Profile
H.264	Frame	Buffer \geq MaxPicwidth x MaxPicheight x 3/4	Frame/Slice	Baseline Mainprofile Highprofile
MJPEG	Frame	Buffer \geq MaxPicwidth x MaxPicheight x 1	Frame/Ecs	-
JPEG	Frame	Buffer \geq MaxPicwidth x MaxPicheight x 1	Frame/Ecs	Baseline
H.265	Frame	Buffer \geq MaxPicwidth x MaxPicheight x 3/4	Frame/Slice	Main profile

- Recommended width and height of a VENC channel: 1920 x 1080 (1080p), 1280 x 720 (720p), 960 x 540, 640 x 360, 704 x 576, 704 x 480, 352 x 288, and 352 x 240.
- Encoder attributes are static attributes except the channel width (**u32PicWidth**) and channel height (**u32PicHeight**). After a VENC channel is created successfully, the static attributes cannot be modified unless the channel is destroyed and then recreated. For details about the precautions to be taken during configuration, see the description of [HI_MPI_VENC_SetChnAttr](#).
- For H.264/H.265 encoding, the VB of the encoding frames consists of **YHeaderSize**, **CHeaderSize**, **YSize**, **CSize**, **PmeSize**, and **TmvSize**. The encoder calculates the VB size and allocates the memory based on the maximum width and height by default. When configuring the channel width and height, ensure that the size of each VB part calculated based on the channel width and height does not exceed that calculated based on the maximum width and height. For details about how to calculate the VB size, see section [6.2.14 "Calculation of the VB for Encoding Frames."](#)
- For JPEG/MJPEG encoding, the channel width and height must meet the following requirements: $u32PicWidth \times u32PicHeight \leq MaxPicWidth \times MaxPicHeight$.
- The RC mode (one of RC attributes) must be set first. The JPEG capture channel does not need to be configured with RC attributes. Other protocol channels (H.264, H.265, and MJPEG are supported) must be configured with RC attributes. The RC mode of the RC must match the protocol type of the encoder.
- During H.264 encoding, H.265 encoding, or MJPEG encoding the bit control modes CBR, VBR, and FIXQP are supported. For different protocols, the attribute variables in the same RC mode are the same. [Table 6-6](#) describes the public attributes of three RC modes.



Table 6-6 Public attributes of three RC modes

RC Mode	GOP	StatTime (s)	FrmRate (SrcFrmRate/DstFrmRate)
CBR	≥ 1	≥ 1	$\text{SrcFrmRate} \geq \text{DstFrmRate}$
VBR	≥ 1	≥ 1	$\text{SrcFrmRate} \geq \text{DstFrmRate}$
FixQp	≥ 1	≥ 1	$\text{SrcFrmRate} \geq \text{DstFrmRate}$

- **SrcFrmRate** must be set to the actual frame rate at which TimeRef is generated. The RC calculates the actual frame rate and controls the bit rate based on the value of **SrcFrmRate**. If the source of the picture to be encoded is VI, **SrcFrmRate** is set to the actual output frame rate of the VI because TimeRef is generated when the VI outputs frames. If the VI and VPSS are in online mode, **SrcFrmRate** is set to the actual output frame rate of the VPSS because TimeRef is generated when the VPSS outputs frames. If the VI and VPSS are in offline mode, the frame rate of the VPSS is controlled by group, and the source frame rate of the VPSS group must be set to the same value as **SrcFrmRate**. In this case, **SrcFrmRate** is set to the actual output frame rate of the VI because TimeRef is generated when the VI outputs frames. If the output frame rate of the VI channel is 30 and the VENC does not control the frame rate, **SrcFrmRate** is set to **30**. If the output frame rate of the VI channel is 30 and the VENC controls the frame rate, the source frame rate and **SrcFrmRate** must be set to **30**. When you set **DstFrmRate**, its data structure is defined as the fractional type HI_FR32 that is the same as HI_U32. That is, the upper 16 bits indicate the denominator and the lower 16 bits indicate the numerator. To set **DstFrmRate** to an integer, set the upper 16 bits to zeros. For example, if you set **SrcFrmRate** to **25** and **DstFrmRate** to **12**, 12 frames will be selected from 25 frames for encoding, the other 13 frames are discarded. If you set **SrcFrmRate** to **25** and **DstFrmRate** to **15/2**, the encoder will encode 15 frames in two seconds.
- **DstFrmRate** can be set as follows:
 - If you want the integral frame rate 25, set **DstFrmRate** to **25**.
 - If you want the fractional frame rate 15/2, set **DstFrmRate** to **[15 + (2 << 16)]**.
- In addition to the attributes described in [Figure 6-2](#), the average bit rate (in kbit/s) and bit rate fluctuation level must be set for the CBR. The average bit rate relates to the width and height of a VENC channel and picture frame rate. [Table 6-7](#) describes the typical average bit rate. Assume that the full encoding frame rate is 25 fps. When the frame rate is not full (for example, 10 fps), you can divide the following bit rate by 2.5 (the product 25 divided by 10) to obtain the actual bit rate.

Table 6-7 Typical configuration of the average bit rate

Width and Height of a Picture/Bit Rate Level	D1 (720 x 576)	720p (1280 x 720)	1080p (1920 x 1080)
Low bit rate	< 400 kbit/s	< 800 kbit/s	< 2000 kbit/s
Medium bit rate	400–1000 kbit/s	800–4000 kbit/s	2000–8000 kbit/s
High bit rate	> 1000 kbit/s	> 4000 kbit/s	> 8000 kbit/s

- There are five fluctuation levels for the bit rate. A higher level indicates a wider range of bit rate fluctuation. If the fluctuation level is high, the picture quality is more stable when



the picture contents of adjacent frames vary significantly. The high level applies to the scenario in which the bandwidth margin is large. If the fluctuation level is low, the encoding bit rate is more stable. When the picture contents of adjacent frames vary significantly, the picture quality at a low level may be worse than that at a high level. The low level applies to the scenario in which the bandwidth margin is small. The function of setting the fluctuation level is reserved and not used currently.

- In VBR mode, besides the attributes described in [Figure 6-2](#), you also need to set **MaxBitRate**, **MaxQp**, and **MinQp**. **MaxBitRate** is the maximum bit rate of a VENC channel within the bit rate statistics period; **MaxQp** is the maximum QP value; **MinQp** is the minimum QP value.
- In FixQp mode, besides the attributes described in [Figure 6-2](#), you also need to set **IQp** and **PQp**. **IQp** is the fixed QP value for pictures in the I frame. **PQp** is the fixed QP value for pictures in the P frame. When setting the **IQp** and **PQp**, you can increase or decrease them based on the current bandwidth. To reduce the respiratory effect, ensure that the **IQp** is greater than the **PQp** by 2 or 3.
- Hi3518E V201 supports only the single stream. That is, only one H.264 encoding channel is allowed to be created.
- The JPEG and MJPEG channels cannot process the frames transmitted from the VPSS low delay channel.

[Example]

```
HI_S32 StartVenc(HI_VOID)
{
    HI_S32 s32Ret;
    VI_CHN ViChn = 0;
    VENC_CHN VeChn = 0;
    VENC_CHN_ATTR_S stAttr;
    MPP_CHN_S stSrcChn, stDestChn;

    /* set h264 channel video encoding attribute */
    stAttr.stVeAttr.enType = PT_H264;
    stAttr.stVeAttr.stAttrH264e.u32PicWidth = u32PicWidth;
    stAttr.stVeAttr.stAttrH264e.u32PicHeight = u32PicHeigh;
    stAttr.stVeAttr.stAttrH264e.u32MaxPicWidth = u32MaxPicWidth;
    stAttr.stVeAttr.stAttrH264e.u32MaxPicHeight = u32MaxPicHeigh;
    stAttr.stVeAttr.stAttrH264e.u32Profile = 2;
    ..... // omit other video encode assignments here.

    /* set h264 channel rate control attribute */
    stAttr.stRcAttr.enRcMode = VENC_RC_MODE_H264CBR ;
    stAttr.stRcAttr.stAttrH264Cbr.u32BitRate = 10*1024;
    stAttr.stRcAttr.stAttrH264Cbr.fr32DstFrmRate = 30;
    stAttr.stRcAttr.stAttrH264Cbr.u32SrcFrmRate = 30;
    stAttr.stRcAttr.stAttrH264Cbr.u32Gop = 30;
    stAttr.stRcAttr.stAttrH264Cbr.u32FluctuateLevel = 1;
    stAttr.stRcAttr.stAttrH264Cbr.u32StatTime = 1;
    ..... // omit other rate control assignments here.
```



```
s32Ret = HI_MPI_VENC_CreateChn(VeChn, &stAttr);
if (HI_SUCCESS != s32Ret)
{
    printf("HI_MPI_VENC_CreateChn err 0x%x\n", s32Ret);
    return HI_FAILURE;
}
s32Ret = HI_MPI_VENC_StartRecvPic(VeChn);
if (s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VENC_StartRecvPic err 0x%x\n", s32Ret);
    return HI_FAILURE;
}

..... // omit other code here.

return HI_SUCCESS;
}.
```

[See Also]

None

HI_MPI_VENC_DestroyChn

[Description]

Destroys a VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_DestroyChn(VENC_CHN VeChn);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h



- Library file: libmpi.a

[Note]

- If a channel that does not exist is reset, an error code indicating failure is returned.
- Before destroying a VENC channel, ensure that picture reception is stopped. Otherwise, an error code indicating failure is returned.
- If the on-screen display (OSD) function is enabled, you need to call HI_MPI_RGN_DetachFrmChn to detach the OSD region from the current encoding channel before destroying the encoding channel.

[Example]

```
HI_S32 StopVenc(HI_VOID)
{
    HI_S32 s32Ret;
    VENC_CHN VeChn = 0;

    s32Ret = HI_MPI_VENC_StopRecvPic(VeChn);
    if (s32Ret != HI_SUCCESS)
    {
        printf("HI_MPI_VENC_StopRecvPic err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }
    s32Ret = HI_MPI_VENC_DestroyChn(VeChn);
    if (s32Ret != HI_SUCCESS)
    {
        printf("HI_MPI_VENC_DestroyChn err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }
    return HI_SUCCESS;
}
```

[See Also]

None

HI_MPI_VENC_Resetchn

[Description]

Resets a VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_ResetChn(VENC_CHN VeChn);
```

[Parameter]



Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM]	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- If a channel that does not exist is reset, the error code [HI_ERR_VENC_UNEXIST](#) is returned.
- If you reset a channel whose function of receiving picture is not stopped, an error code indicating failure is returned.

[Example]

```
HI_S32 ResetVenc(HI_VOID)
{
    HI_S32 s32Ret;
    VI_CHN ViChn = 0;
    VENC_CHN VeChn = 0;
    VENC_CHN_ATTR_S stAttr;
    MPP_CHN_S stSrcChn, stDestChn;
    /* set h264 channel video encode attribute */
    stAttr.stVeAttr.enType = PT_H264;
    stAttr.stVeAttr.stAttrH264e.u32PicWidth = u32PicWidth;
    stAttr.stVeAttr.stAttrH264e.u32PicHeight = u32PicHeigh;
    stAttr.stVeAttr.stAttrH264e.u32MaxPicWidth = u32MaxPicWidth;
    stAttr.stVeAttr.stAttrH264e.u32MaxPicHeight = u32MaxPicHeigh;
    stAttr.stVeAttr.stAttrH264e.u32Profile = 2;
    .... // omit other video encode assignments here.
    /* set h264 channel rate control attribute */
    stAttr.stRcAttr.enRcMode = VENC_RC_MODE_H264CBR ;
    stAttr.stRcAttr.stAttrH264Cbr.u32BitRate = 10*1024;
    stAttr.stRcAttr.stAttrH264Cbr.fr32DstFrmRate = 30;
    stAttr.stRcAttr.stAttrH264Cbr.u32SrcFrmRate = 30;
```



```
stAttr.stRcAttr.stAttrH264Cbr.u32Gop          = 30;
stAttr.stRcAttr.stAttrH264Cbr.u32FluctuateLevel = 1;
stAttr.stRcAttr.stAttrH264Cbr.u32StatTime       = 1;
..... // omit other rate control assignments here.

s32Ret = HI_MPI_VENC_CreateChn(VeChn, &stAttr);
if (HI_SUCCESS != s32Ret)
{
    printf("HI_MPI_VENC_CreateChn err 0x%x\n", s32Ret);
    return HI_FAILURE;
}

s32Ret = HI_MPI_VENC_StartRecvPic(VeChn);
if (s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VENC_StartRecvPic err 0x%x\n", s32Ret);
    return HI_FAILURE;
}

s32Ret = HI_MPI_VENC_StopRecvPic(VeChn);
if (s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VENC_StopRecvPic err 0x%x\n", s32Ret);
    return HI_FAILURE;
}

s32Ret = HI_MPI_VENC_ReSetChn(VeChn);
if (s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VENC_ReSetChn err 0x%x\n", s32Ret);
    return HI_FAILURE;
}
..... // omit other code here.
return HI_SUCCESS;
}
```

[See Also]

None

HI_MPI_VENC_StartRecvPic

[Description]

Enables a VENC channel to start receiving pictures.

[Syntax]



```
HI_S32 HI_MPI_VENC_StartRecvPic(VENC_CHN VeChn);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- If a VENC channel is not created, error [HI_ERR_VENC_UNEXIST](#) is returned.
- This MPI does not check whether the function of receiving picture is enabled. That is, the function can be repeatedly enabled, and no error is returned.
- The encoder starts receiving pictures only after this function is enabled.

[Example]

See the example of [HI_MPI_VENC_CreateChn](#).

[See Also]

None

HI_MPI_VENC_StartRecvPicEx

[Description]

Enables a VENC channel to start receiving input pictures and automatically stop the VENC channel receiving input pictures after the specified number of frames is exceeded.

[Syntax]

```
HI_S32 HI_MPI_VENC_StartRecvPicEx(VENC_CHN VeChn, VENC\_RECV\_PIC\_PARAM\_S*pstRecvParam);
```

[Parameter]



Parameter	Description	Input/Output
VeChn	VENC channel ID. Value range: [0, VENC_MAX_CHN_NUM)	Input
pstRecvParam	Pointer to the received picture parameter structure. This pointer indicates the number of frames to be received.	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code.

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- If the VENC channel is not created, [HI_ERR_VENC_UNEXIST](#) is returned.
- If the VENC channel is enabled to start receiving pictures without stop by calling [HI_MPI_VENC_StartRecvPic](#) or has not received sufficient pictures by calling [HI_MPI_VENC_StartRecvPicEx](#) last time, an error code is returned if [HI_MPI_VENC_StartRecvPicEx](#) is called again, indicating that this operation is prohibited.
- This MPI is used to receive and encode consecutive N frames. When N is 0, [HI_MPI_VENC_StartRecvPicEx](#) is equivalent to [HI_MPI_VENC_StartRecvPic](#).
- If [HI_MPI_VENC_StartRecvPic](#) has been called to receive pictures before (picture reception is stopped) and you want to call [HI_MPI_VENC_StartRecvPicEx](#) to start encoding again, you are advised to call [HI_MPI_VENC_Resetchn](#) to clear the buffered pictures and streams before calling [HI_MPI_VENC_StartRecvPicEx](#).
- If a JPEG snapshot channel is created, you are advised to call [HI_MPI_VENC_StartRecvPicEx](#).

[Example]

```
HI_S32 JpegSnapProcess(HI_VOID)
{
    HI_S32 s32Ret;
    VENC_CHN VeChn = 0;
    VENC_CHN_ATTR_S stAttr;
    VENC_RECV_PIC_PARAM_S stRecvParam;
    /*Set the video encoding attributes of the JPEG channel. */
    stAttr.stVeAttr.enType    = PT_JPEG;
    stAttr.stVeAttr.stAttrJpeg.u32PicWidth = u32PicWidth;
```



```
    stAttr.stVeAttr.stAttrJpeg.u32PicHeight = u32PicHeigh;
    stAttr.stVeAttr.stAttrJpeg.u32MaxPicWidth = u32MaxPicWidth;
    stAttr.stVeAttr.stAttrJpeg.u32MaxPicHeight = u32MaxPicHeigh;
..... // omit other video encode assignments here.

// Create a JPEG channel.

    s32Ret = HI_MPI_VENC_CreateChn(VeChn, &stAttr);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_CreateChn err 0x%x\n",s32Ret);
        return HI_FAILURE;
    }

// start snapping

    stRecvParam.s32RecvPicNum = 2;

    s32Ret = HI_MPI_VENC_StartRecvPicEx(VeChn, &stRecvParam);
    if (s32Ret != HI_SUCCESS)
    {
        printf("HI_MPI_VENC_StartRecvPicEx err 0x%x\n",s32Ret);
        return HI_FAILURE;
    }

..... // wait until all pictures have been encoded.

    s32Ret = HI_MPI_VENC_StopRecvPic(VeChn);
    if (s32Ret != HI_SUCCESS)
    {
        printf("HI_MPI_VENC_StopRecvPic err 0x%x\n",s32Ret);
        return HI_FAILURE;
    }

// Destroy the JPEG channel.

    s32Ret = HI_MPI_VENC_DestroyChn(VeChn);
    if (s32Ret != HI_SUCCESS)
    {
        printf("HI_MPI_VENC_DestroyChn err 0x%x\n",s32Ret);
        return HI_FAILURE;
    }

    return HI_SUCCESS;
}
```

[See Also]

None

HI_MPI_VENC_StopRecvPic

[Description]

Stops a VENC channel receiving input pictures.

[Syntax]



```
HI_S32 HI_MPI_VENC_StopRecvPic(VENC_CHN VeChn);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- If a VENC channel is not created, an error code indicating failure is returned.
- This MPI does not check whether picture reception is stopped. That is, picture reception can be repeatedly stopped, and no error is returned.
- This MPI is used to stop a VENC channel receiving pictures. Before being destroyed or reset, the channel must stop receiving pictures.
- When this MPI is called, only receiving raw data is stopped but the stream buffer is not cleared.
- Picture reception that starts by calling [HI_MPI_VENC_StartRecvPic](#) or [HI_MPI_VENC_StartRecvPicEx](#) can be stopped by calling [HI_MPI_VENC_StopRecvPic](#).

[Example]

See the example of [HI_MPI_VENC_DestroyChn](#).

[See Also]

None

HI_MPI_VENC_Query

[Description]

Queries the status of a VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_Query(VENC_CHN VeChn, VENC_CHN_STAT_S *pstStat);
```



[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstStat	Pointer to the status of a VENC channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- If a VENC channel is not created, an error code indicating failure is returned.
- This MPI is called to query the status of the encoder. The **pstStat** parameter contains the following information:
 - u32LeftPics** indicates the number of frames to be encoded.
Before resetting a VENC channel, you can check whether there are pictures to be encoded, preventing the frames from being cleared during reset.
 - u32LeftStreamBytes** indicates the number of remaining bytes in the stream buffer.
Before resetting a VENC channel, you can check whether there are streams to be processed, preventing the streams from being cleared during reset.
 - u32LeftStreamFrames** indicates the number of remaining frames in the stream buffer.
Before resetting a VENC channel, you can check whether there are streams that are not obtained, preventing the streams from being cleared during reset.
 - u32CurPacks** indicates the number of stream packets in the current frames. Before calling [HI_MPI_VENC_GetStream](#), ensure that **u32CurPacks** is greater than 0.
When streams are obtained by packet, the current frame may not be a complete frame (part of the frame is obtained). When streams are obtained by frame, this parameter indicates the number of packets in the current frame (if it is not a complete frame, this parameter is set to 0). When obtaining streams by frame, you need to check the number of stream packets of a complete frame. Typically, after the select function is successfully called, you can query the number of stream packets in the selected frame, which is specified by the **u32CurPacks** parameter.
 - u32LeftRecvPics** indicates the number of frames to be received after [HI_MPI_VENC_StartRecvPicEx](#) is called.



- **u32LeftEncPics** indicates the number of frames to be encoded after [HI_MPI_VENC_StartRecvPicEx](#) is called.

The number of frames to be received or encoded is 0 if [HI_MPI_VENC_StartRecvPicEx](#) is not called.

[Example]

See [HI_MPI_VENC_GetStream](#).

[See Also]

None

HI_MPI_VENC_SetChnAttr

[Description]

Sets attributes of a VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetChnAttr(VENC_CHN VeChn, const VENC_CHN_ATTR_S*  
pstAttr);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstAttr	Pointer to the VENC channel attributes	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- The attributes of a VENC channel such as maximum width and maximum height cannot be dynamically changed.
- The attributes of a VENC channel include the encoder attributes and RC attributes.
- When you attempt to set attributes for a nonexistent VENC channel, an error code indicating failure is returned.



- If **pstAttr** is null, an error code indicating failure is returned.
- The limitations for setting the size of the pictures to be encoded are the same as those for creating a channel. For details, see [Table 6-5](#).
- A VENC channel has dynamic attributes and static attributes. The dynamic attributes are configured when the channel is created and can be modified before the channel is destroyed. The static attributes are configured when the channel is created and cannot be modified after the channel is created.
- This MPI is called to configure only dynamic attributes. If you attempt to configure static attributes by using this MPI, an error code indicating a failure is returned. The static attributes include encoding protocol, method of obtaining streams (streams are obtained by frame or by packet), and the maximum width and height of encoded pictures. Static attributes are specified by each protocol module. For details, see [VENC_CHN_ATTR_S](#).
- The RC attribute (included in VENC channel attributes) can be configured, and the bit rate control mode is VBR. When the value of **u32MinIQp** (advanced bit rate control parameter) is less than the value of **u32MinQp** (RC attribute), calling **HI_MPI_VENC_SetChnAttr** changes the value of **u32MinIQp** to that of **u32MinQp**.
- After the channel height and width in the encoder attributes are configured, all parameters for the encoding channel are restored to the default values except the channel priority, the OSD function is disabled, and the stream buffer and picture buffer queue are cleared.
- To reconfigure the stream resolution when the OSD function is enabled, perform the following steps:

Step 1 Call [HI_MPI_VENC_StopRecvPic](#) to stop the encoding channel from receiving pictures.

Step 2 Call [HI_MPI_RGN_DetachFrmChn](#) to detach the OSD region from the encoding channel.

Step 3 Call [HI_MPI_VENC_SetChnAttr](#) to reconfigure the encoding stream resolution.

Step 4 Call [HI_MPI_RGN_AttachToChn](#) to overlay the OSD region to the encoding channel.

Step 5 Call [HI_MPI_VENC_StartRecvPic](#) to start receiving pictures.

----End

[Example]

None

[See Also]

None

HI_MPI_VENC_GetChnAttr

[Description]

Obtains attributes of a VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetChnAttr(VENC_CHN VeChn, VENC_CHN_ATTR_S*pstAttr);
```

[Parameter]



Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstAttr	Pointer to the VENC channel attributes	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- When you attempt to obtain attributes of a nonexistent VENC channel, error [HI_ERR_VENC_UNEXIST](#) is returned.
- If **pstAttr** is null, an error code indicating failure is returned.

[Example]

None

[See Also]

None

HI_MPI_VENC_GetStream

[Description]

Obtains encoded streams.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetStream(VENC_CHN VeChn, VENC\_STREAM\_S *pstStream,  
HI_S32 s32MilliSec);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstStream	Pointer to the stream structure	Output



Parameter	Description	Input/Output
s32MilliSec	Timeout period for obtaining streams Value range: $[-1, +\infty)$ • -1 : block mode • 0 : non-block mode • > 0 : timeout period	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: `hi_comm_venc.h`, `mpi_venc.h`
- Library file: `libmpi.a`

[Note]

- If a VENC channel is not created, an error code indicating failure is returned.
- If `pstStream` is null, the error code `HI_ERR_VENC_NULL_PTR` is returned.
- If the `s32MilliSec` is smaller than -1 , the error code `HI_ERR_VENC_ILLEGAL_PARAM` is returned.
- Streams can be obtained in timeout mode. In addition, the select or poll function can be called.
 - If `s32MilliSec` is 0 , streams are obtained in non-block mode. To be specific, if there is no data in the buffer, `HI_ERR_VENC_BUF_EMPTY` is returned.
 - If `s32MilliSec` is -1 , streams are obtained in block mode. To be specific, if there is no data in the buffer, the code indicating that streams are successfully obtained is returned only after there is data in the buffer.
 - If `s32MilliSec` is greater than 0 , streams are obtained in timeout mode. To be specific, if there is no data in the buffer, the code indicating that streams are successfully obtained is returned when there is data within the configured period. Otherwise, a code indicating failure due to timeout is returned.
- Streams can be obtained by packet or frame. When streams are obtained by packet:
 - For the H.264 protocol, an NAL unit is obtained each time this MPI is called.
 - For the JPEG protocol (including JPEG capture and MJPEG), an ECS or a picture parameter stream packet is obtained each time this MPI is called.
 - For the H.265 protocol, an NAL unit is obtained each time this MPI is called.
- The stream structure `VENC_STREAM_S` contains the following information:
 - Pointer to a stream packet `pstPack`



This parameter specifies the memory space of the **VENC_PACK_S**. The space is allocated by the caller. When streams are obtained by packet, the space is not smaller than the size specified by the **VENC_PACK_S**; when streams are obtained by frame, the space is not smaller than $n \times$ size specified by the **VENC_PACK_S**. Here, n indicates the number of stream packets in the current frame and can be obtained by calling the query MPI after the Select invocation.

- Number of stream packets **u32PackCount**

This parameter specifies the value of **VENC_PACK_S** in **pstPack**. When streams are obtained by packet, **u32PackCount** should not be smaller than 1; when streams are obtained by frame, **u32PackCount** should not be less than the number of packets in the current frame. After this MPI is successfully called, the value of **u32PackCount** is the number of packets specified in **pstPack**.

- Sequence number **u32Seq**

When streams are obtained by frame, it is the sequence number of a frame; when streams are obtained by packet, it is the sequence number of a packet.

- Stream features **stH624Info/stJpegInfo/stMpeg4Info/stH265Info**. The data structure is a union, including stream features corresponding to various protocols. The output stream feature information is used for upper-layer applications of users.

- If streams are not fetched for a long period of time, the stream buffer becomes full. If the stream buffer corresponding to a VENC channel is full, encoding is not started until streams are fetched. Stream encoding starts only when the buffer for encoding is sufficient.
- You are advised to call **HI_MPI_VENC_GetStream** and **HI_MPI_VENC_ReleaseStream** in pairs and release the stream buffer as soon as possible. Otherwise, the stream buffer is full when streams are obtained in user mode. As a result, encoding stops.
- When **OneStreamBuffer** is 1, if one frame of stream is obtained, the address for only one stream packet (excluding user data) is obtained. That is, **u32PackCount** is 1, and the address is **pstPack[0].pu8Addr.u32Offset** is added to **VENC_PACK_S** for specifying the address for the valid data in a frame stream and **pstPack[0].pu8Addr** offset.
- You are advised to obtain streams by calling the select function and perform the following steps:

Step 1 Call **HI_MPI_VENC_Query** to query the status of the encoding channel.

Step 2 Check the values of **u32CurPacks** and **u32LeftStreamFrames**, and ensure that they are greater than 0.

Step 3 Call the malloc function to allocate **u32CurPacks** packet information data structures.

Step 4 Call **HI_MPI_VENC_GetStream** to obtain the encoding stream.

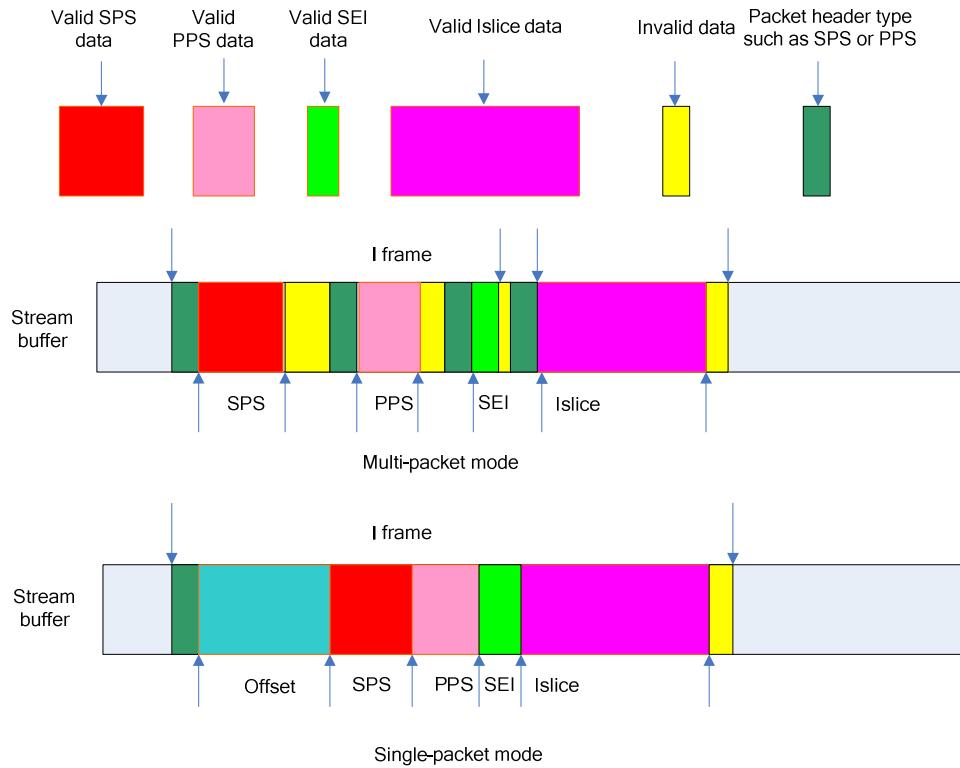
Step 5 Call **HI_MPI_VENC_ReleaseStream** to release the stream buffer.

----End

- The H.264 protocol is used as an example, as shown in [Figure 6-10](#). If **pstPack[0].u32DataNum = 3**, there are three types of NAL packets, including SPS, PPS, and SEI. **pstPack[0].stPackInfo[0].u32PackType.enH264EType = H264E_NALU_SPS**; **pstPack[0].stPackInfo [1].u32PackType.enH264EType = H264E_NALU_PPS**; **pstPack[0].stPackInfo [2].u32PackType.enH264EType = H264E_NALU_SEI**. The NAL packet types for other protocols are similar.



Figure 6-10 Stream packet structure



The configuration mode can be selected by setting the **OneStreamBuffer** parameter when .ko drivers of the protocols are loaded. If **OneStreamBuffer** is set to 1, the single-packet mode is selected. If **OneStreamBuffer** is set to 0, the multi-packet mode is selected. The default value of **OneStreamBuffer** is 0.

- Streams can be obtained in cache mode. The mode can be specified by configuring **VencBufferCache** when the VENC .ko files are loaded. If **VencBufferCache** is set to 1, streams are obtained in cache mode. If **VencBufferCache** is set to 0, streams are obtained in non-cache mode. **VencBufferCache** is set to 0 by default.

[Example]

```
HI_S32 VencGetH264Stream(HI_VOID)
{
    HI_S32 i;
    HI_S32 s32Ret;
    HI_S32 s32VencFd;
    HI_U32 u32FrameIdx = 0;
    VENC_CHN VeChn = 0;
    VENC_CHN_STAT_S stStat;
    VENC_STREAM_S stStream;
    fd_set read_fds;
    FILE *pFile = NULL;
```



```
pFile = fopen("stream.h264", "wb");
if(pFile == NULL)
{
    return HI_FAILURE;
}

s32VencFd = HI_MPI_VENC_GetFd(VeChn);
do{
    FD_ZERO(&read_fds);
    FD_SET(s32VencFd, &read_fds);

    s32Ret = select(s32VencFd+1, &read_fds, NULL, NULL, NULL);
    if (s32Ret < 0)
    {
        printf("select err\n");
        return HI_FAILURE;
    }
    else if (0 == s32Ret)
    {
        printf("time out\n");
        return HI_FAILURE;
    }
    else
    {
        if (FD_ISSET(s32VencFd, &read_fds))
        {
            s32Ret = HI_MPI_VENC_Query(VeChn, &stStat);
            if (s32Ret != HI_SUCCESS)
            {
                return HI_FAILURE;
            }
        }
    }
}

/*********************************************
suggest to check both u32CurPacks and u32LeftStreamFrames at
the same time, for example:
if(0 == stStat.u32CurPacks || 0 == stStat.u32LeftStreamFrames)
{
    continue;
}
*****************************************/
if(0 == stStat.u32CurPacks)
{
    continue;
}
```



```
    stStream.pstPack = (VENC_PACK_S*)
        malloc(sizeof(VENC_PACK_S)*stStat.u32CurPacks);
    if (NULL == stStream.pstPack)
    {
        return HI_FAILURE;
    }

    stStream.u32PackCount = stStat.u32CurPacks;
    s32Ret = HI_MPI_VENC_GetStream(VeChn, &stStream, -1);
    if (HI_SUCCESS != s32Ret)
    {
        free(stStream.pstPack);
        stStream.pstPack = NULL;
        return HI_FAILURE;
    }

    for (i=0; i< stStream.u32PackCount; i++)
    {
        fwrite(stStream.pstPack[i].pu8Addr+
stStream.pstPack[i].u32Offset,
               1, stStream.pstPack[i].u32Len-
stStream.pstPack[i].u32Offset, pFile);

    }

    s32Ret = HI_MPI_VENC_ReleaseStream(VeChn,&stStream);
    if (HI_SUCCESS != s32Ret)
    {
        free(stStream.pstPack);
        stStream.pstPack = NULL;
        return HI_FAILURE;
    }

    free(stStream.pstPack);
    stStream.pstPack = NULL;
}

}

u32FrameIdx++;
}while (u32FrameIdx < 0xff);

fclose(pFile);
return HI_SUCCESS;
}
```



For details, see the related sample codes.

[See Also]

None

HI_MPI_VENC_ReleaseStream

[Description]

Releases a stream buffer.

[Syntax]

```
HI_S32 HI_MPI_VENC_ReleaseStream(VENC_CHN VeChn, VENC_STREAM_S  
*pstStream);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstStream	Pointer to the stream structure	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- If a VENC channel is not created, error [HI_ERR_VENC_UNEXIST](#) is returned.
- If **pstStream** is null, error [HI_ERR_VENC_NULL_PTR](#) is returned.
- This MPI must work with the [HI_MPI_VENC_GetStream](#) MPI. You are advised to release the obtained stream buffer. Otherwise, the stream buffer may be full, affecting encoding. In addition, you must comply with the principle of "first obtained, first released".
- When a VENC channel is reset, all stream packets that are not released are invalid and the stream buffer cannot be used or released any more.
- When you release invalid streams, error [HI_ERR_VENC_ILLEGAL_PARAM](#) is returned.



[Example]

See [HI_MPI_VENC_GetStream](#).

[See Also]

None

HI_MPI_VENC_GetStreamBufInfo

[Description]

Obtains the physical address and size of a stream buffer.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetStreamBufInfo(VENC_CHN VeChn,  
VENC_STREAM_BUF_INFO_S *pstStreamBufInfo)
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstStreamBufInfo	Pointer to stream buffer information	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- If no VENC channel is created, [HI_ERR_VENC_UNEXIST](#) is returned.
- If **pstStreamBufInfo** is null, [HI_ERR_VENC_NULL_PTR](#) is returned.
- If you want to use the physical address for a frame stream, you need to obtain the start physical address and size of the stream buffer by calling this MPI for checking whether the stream is wrapped. If the following condition is met, the frame stream is wrapped and two physical addresses are obtained: **pstPack[0].u32PhyAddr + u32Len > pstStreamBufInfo->u32PhyAddr + pstStreamBufInfo->u32BufSize**. This ensures that the correct physical address is used. **pstPack[0]** is the variable of [VENC_STREAM_S](#), and **u32Len** indicates the length of a frame stream. See [Figure 6-11](#).



[Example]

```
HI_S32 VencGetH264StreamBufInfo(HI_VOID)
{
    HI_S32 i;
    HI_S32 s32Ret;
    HI_S32 s32VencFd;
    HI_U32 u32FrameIdx = 0;
    VENC_CHN VeChn = 0;
    VENC_CHN_STAT_S stStat;
    VENC_STREAM_S stStream;
    fd_set read_fds;
    VENC_STREAM_BUF_INFO_S stStreamBufInfo;
    HI_U32 u32Left;
    HI_U32 u32SrcPhyAddr, u32DestPhyAddr;

    s32Ret = HI_MPI_VENC_GetStreamBufInfo (VeChn, &stStreamBufInfo);
    if (HI_SUCCESS != s32Ret)
    {
        return HI_FAILURE;
    }

    do{
        s32Ret = HI_MPI_VENC_Query(VeChn, &stStat);
        if (s32Ret != HI_SUCCESS)
        {
            return HI_FAILURE;
        }

        stStream.pstPack = (VENC_PACK_S*)
            malloc(sizeof(VENC_PACK_S)*stStat.u32CurPacks);
        if (NULL == stStream.pstPack)
        {
            return HI_FAILURE;
        }

        stStream.u32PackCount = stStat.u32CurPacks;
        s32Ret = HI_MPI_VENC_GetStream(VeChn, &stStream, -1);
        if (HI_SUCCESS != s32Ret)
        {
            free(stStream.pstPack);
            stStream.pstPack = NULL;
            return HI_FAILURE;
        }
    }
```



```
//DMA transfer, use only the physical address
for (i=0; i< stStream.u32PackCount; i++)
{
    if(stStream.pstPack[i].u32PhyAddr+
       stStream.pstPack[i].u32Len >=
       stStreamBufInfo.u32PhyAddr +
       stStreamBufInfo.u32BufSize)
    {
        if(stStream.pstPack[i].u32PhyAddr+
           stStream.pstPack[i].u32Offset >=
           stStreamBufInfo.u32PhyAddr +
           stStreamBufInfo.u32BufSize)
        {
            // (a) branch in picture
            u32SrcPhyAddr = stStreamBufInfo.u32PhyAddr+
                ((stStream.pstPack[i].u32PhyAddr+
                  stStream.pstPack[i].u32Offset) -
                 (stStreamBufInfo.u32PhyAddr +
                  stStreamBufInfo. u32BufSize));

            DMA_TransFer(u32SrcPhyAddr, u32DestPhyAddr,
                         stStream.pstPack[i].u32Len-
                         stStream.pstPack[i].u32Offset);
        }
        else
        {
            // (b) branch in picture
            u32Left = (stStreamBufInfo.u32PhyAddr +
                        stStreamBufInfo.u32BufSize) -
                        stStream.pstPack[i].u32PhyAddr;
            DMA_TransFer(stStream.pstPack[i].u32PhyAddr +
                         stStream.pstPack[i].u32Offset,
                         u32DestPhyAddr,u32Left -
                         stStream.pstPack[i].u32Offset);
            DMA_TransFer(stStreamBufInfo.u32PhyAddr,
                         u32DestPhyAddr, stStream.pstPack[i].u32Len-
                         u32Left);
        }
    }
    else
    {
        // (c) branch in picture
        DMA_TransFer (stStream.pstPack[i].u32PhyAddr +
```



```
    stStream.pstPack[i].u32Offset,
    stStream.pstPack[i].u32Len-
    stStream.pstPack[i].u32Offset);
}
}

s32Ret = HI_MPI_VENC_ReleaseStream(VeChn, &stStream);
if (HI_SUCCESS != s32Ret)
{
    free(stStream.pstPack);
    stStream.pstPack = NULL;
    return HI_FAILURE;
}

free(stStream.pstPack);
stStream.pstPack = NULL;

u32FrameIdx++;
}while (u32FrameIdx < 0xff);
return HI_SUCCESS;
}
```

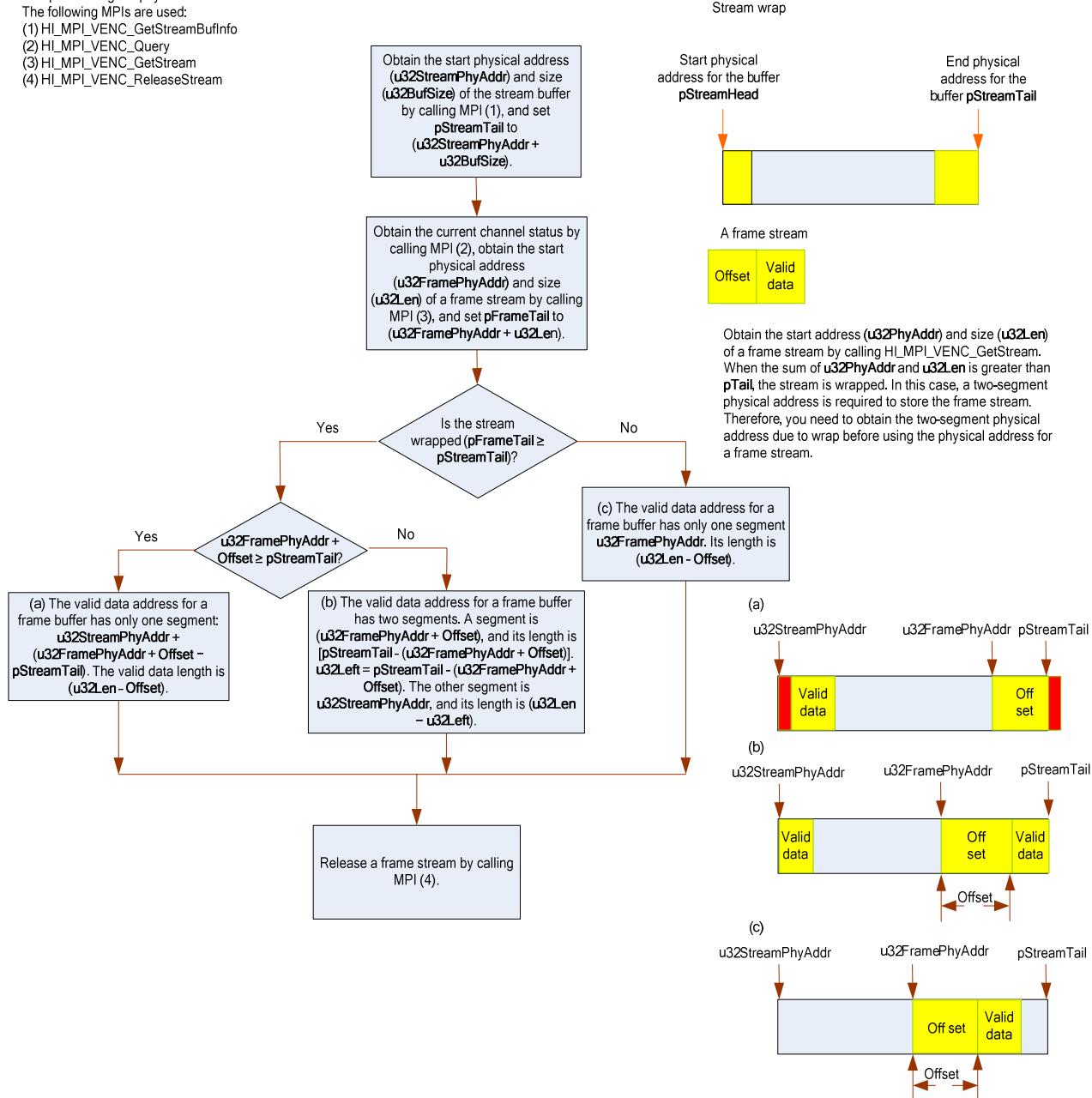


Figure 6-11 Checking whether the stream buffer is wrapped

Sample of using the physical address for a stream buffer.

The following MPIs are used:

- (1) HI_MPI_VENC_GetStreamBufInfo
- (2) HI_MPI_VENC_Query
- (3) HI_MPI_VENC_GetStream
- (4) HI_MPI_VENC_ReleaseStream



[See Also]

None

HI_MPI_VENC_InsertUserData

[Description]

Inserts user data.



[Syntax]

```
HI_S32 HI_MPI_VENC_InsertUserData(VENC_CHN VeChn, HI_U8 *pu8Data, HI_U32 u32Len);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pu8Data	Pointer to user data	Input
u32Len	User data length, in byte Value range: (0, 1024]	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- If a VENC channel is not created, an error code indicating failure is returned.
- If **pu8Data** is null, an error code indicating failure is returned.
- User data can be inserted by following the H.264/H.265 and MJPEG/JPEG protocols.
- For the H.264 channel, at most four memory spaces are provided for buffering user data, and the size of each data segment is not greater than 1 KB. If more than four data segments are inserted or a user data segment is greater than 1 KB, an error is returned. Each user data segment is inserted as an SEI packet before the latest picture stream packet. After a user data segment is encoded and sent, the memory space in the H.264 channel for buffering the user data is cleared for storing new user data.
- For the JPEG/MJPEG channel, at most one memory space is provided for buffering 1 KB user data. The user data is inserted into picture streams as APP segment (0xFFEF). After user data is encoded and sent, the memory space in the JPEG/MJPEG channel for buffering the user data is cleared for storing new user data.

[Example]

```
HI_U8 au8UserData[] = "hisilicon2011";
s32Ret = HI_MPI_VENC_InsertUserData(VeChn, au8UserData,
                                     sizeof(au8UserData));
```



```
if(HI_SUCCESS != s32Ret)
{
    printf("HI_MPI_VENC_InsertUserData err 0x%xn",s32Ret);
}
```

[See Also]

None

HI_MPI_VENC_SendFrame

[Description]

Sends raw pictures for encoding.

[Syntax]

```
HI_S32 HI_MPI_VENC_SendFrame(VENC_CHN VeChn, VIDEO_FRAME_INFO_S *pstFrame,
HI_S32 s32MilliSec)
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstFrame	Pointer to a raw picture structure	Input
s32MilliSec	Timeout period for sending pictures Value range: [-1, +∞) • -1: block mode • 0: non-block mode • > 0: timeout period	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI allows you to send pictures to a VENC channel.



- If the **s32MilliSec** is smaller than **-1**, the error code **HI_ERR_VENC_ILLEGAL_PARAM** is returned.
- The source pictures must be semi-planar YUV 4:2:0 or semi-planar YUV 4:2:2 pictures. The H.264/H.265 VENC channel receives semi-planar420 pictures, and the JPEG or MJPEG VENC channel receives semi-planar420 or semi-planar422 pictures.
- The source picture size must be greater than or equal to the VENC channel size.
- Before calling this MPI to send pictures, ensure that the corresponding VENC channel is created and enabled.

[Example]

For details, see the related sample codes.

[See Also]

None

HI_MPI_VENC_SetMaxStreamCnt

[Description]

Sets the maximum number of frames in a stream buffer.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetMaxStreamCnt(VENC_CHN VeChn, HI_U32 u32MaxStrmCnt);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
u32MaxStrmCnt	Maximum number of frames in a stream buffer	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: `hi_comm_venc.h`, `mpi_venc.h`
- Library file: `libmpi.a`

[Note]

- This API is used to set the maximum number of frames in a stream buffer.



- If the number of stream frames in the buffer reaches the maximum value, the current pictures to be encoded are not encoded because the stream buffer is full.
- If the number of stream frames in the buffer reaches the maximum number of stream frames, the picture to be encoded is discarded.
- By default, the maximum number of stream frames is set to **200** by the system when a VENC channel is created.
- This MPI must be called after a VENC channel is created and before the channel is destroyed. This MPI takes effect before a frame is encoded and can be called repeatedly. You are advised to set the maximum number of stream frames after a VENC channel is created and before frames are encoded. Dynamic settings are not recommended.

[Example]

None

[See Also]

None

HI_MPI_VENC_GetMaxStreamCnt

[Description]

Obtains the maximum number of frames in a stream buffer.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetMaxStreamCnt (VENC_CHN VeChn, HI_U32  
*pu32MaxStrmCnt);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pu32MaxStrmCnt	Pointer to the maximum number of frames in a stream buffer	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a



[Note]

None

[Example]

None

[See Also]

None

HI_MPI_VENC_RequestIDR

[Description]

Requests IDR frames.

[Syntax]

```
HI_S32 HI_MPI_VENC_RequestIDR(VENC_CHN VeChn, HI_BOOL bInstant);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
bInstant	IDR frame instant encoding enable	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- If a VENC channel is not created, an error code indicating failure is returned.
- After the request for IDR frames or I frames is received, if **bInstant** is **0**, IDR frames or I frames are encoded after the current frame is encoded; if **bInstant** is **1**, IDR frames or I frames are encoded immediately regardless of the frame rate.
- An IDR frame request or I frame request can be sent in compliance with the H.264/H.265 protocol.



- This MPI is not affected by frame rate control, and an IDR frame or I frame is encoded each time this MPI is called. The frame rate and bit rate are unstable if this MPI is frequently called.

[Example]

```
HI_S32 s32Ret;
VENC_CHN VeChn = 0;
s32Ret = HI_MPI_VENC_RequestIDR(VeChn, HI_TRUE);
if(HI_SUCCESS != s32Ret)
{
    printf("HI_MPI_VENC_RequestIDR err 0x%xn", s32Ret);
}
```

[See Also]

None

HI_MPI_VENC_EnableIDR

[Description]

Enables an IDR frame.

[Syntax]

```
HI_S32 HI_MPI_VENC_EnableIDR(VENC_CHN VeChn, HI_BOOL bEnableIDR);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
bEnableIDR	Whether the IDR frame is enabled	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]



- If no channel is created, an error code indicating failure is returned.
- If the IDR frame is disabled, no IDR frame or I frame can be encoded in the next frame unless the IDR frame is enabled.
- This MPI supports only the H.264/H.265 encoding protocol.

[Example]

```
HI_S32 s32Ret;
VENC_CHN VeChn = 0;
s32Ret = HI_MPI_VENC_EnableIDR(VeChn);
if(HI_SUCCESS != s32Ret)
{
    printf("HI_MPI_VENC_EnableIDR err 0x%xn",s32Ret);
}
```

[See Also]

None

HI_MPI_VENC_SetH264IdrPicId

[Description]

Sets the idr_pic_id of an IDR frame.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetH264IdrPicId(VENC_CHN VeChn,
VENC_H264_IDRPICID_CFG_S* pstH264eIdrPicIdCfg);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstH264eIdrPicIdCfg	Pointer to the idr_pic_id	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a



[Note]

- If no channel is created, an error code indicating failure is returned.
- The idr_pic_id depends on the following two parameters:
 - **enH264eIdrPicIdMode**: This parameter is used to set the mode of setting the idr_pic_id of the IDR frame. If **enH264eIdrPicIdMode** is **H264E_IDR_PICTURE_ID_MODE_AUTO**, the idr_pic_id is automatically generated in the internal encoding process. If **enH264eIdrPicIdMode** is **H264E_IDR_PICTURE_ID_MODE_USR**, the idr_pic_id is user-defined.
 - **u32H264eIdrPicId**: This parameter is used to specify the idr_pic_id. **u32H264eIdrPicId** is valid only when **enH264eIdrPicIdMode** is **H264E_IDR_PICTURE_ID_MODE_USR**.
- This MPI supports only the H.264 encoding protocol.
- This MPI can be called after a VENC channel is created or before a VENC channel is destroyed. The configured idr_pic_id takes effect before the next frame is encoded. In addition, this MPI can be repeatedly called. You are advised to set the idr_pic_id after a VENC channel is successfully created and before encoding starts. That is, it is not recommended that the idr_pic_id be dynamically changed during encoding.

[Example]

None

[See Also]

None

HI_MPI_VENC_GetH264IdrPicId

[Description]

Obtains the idr_pic_id of an IDR frame.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetH264IdrPicId(VENC_CHN VeChn,  
VENC_H264_IDRPICID_CFG_S* pstH264eIdrPicIdCfg);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstH264IdrPicIdCfg	Pointer to the idr_pic_id	Output

[Return Value]

Return Value	Description
0	Success



Return Value	Description
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- If no channel is created, an error code indicating failure is returned.
- This MPI can be called after a VENC channel is created or before a VENC channel is destroyed.

[Example]

None

[See Also]

None

HI_MPI_VENC_GetFd

[Description]

Obtains the device file handle of a VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetFd(VENC_CHN VeChn);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input

[Return Value]

Return Value	Description
A positive value	Valid
A non-positive value	Invalid

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a



[Note]

None

[Example]

None

[See Also]

None

HI_MPI_VENC_CloseFd

[Description]

Closes the device file handle of a VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_CloseFd(VENC_CHN VeChn);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input

[Return Value]

Return Value	Description
0	Success
-1	Failure

[Error Code]

None

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

None

[Example]

None

[See Also]

None



HI_MPI_VENC_SetRoiCfg

[Description]

Sets the ROI attributes of an H.264/H.265 channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetRoiCfg (VENC_CHN VeChn, VENC_ROI_CFG_S  
*pstVencRoiCfg);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstVencRoiCfg	ROI attributes of the H.264 streams	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to set the ROI attribute of an H.264/H.265 channel.
- The ROI parameters include:
 - u32Index:** The system can be configured with eight ROIs for each H.264 channel, internally numbered 0 to 7 for management. The value of this parameter indicates the index number of an ROI set by users. ROIs can be overlaid. When ROIs are overlaid, the priority of the ROIs increases with the index number.
 - bEnable:** Whether the current ROI is enabled.
 - bAbsQp:** Whether the current ROI uses an absolute QP or a relative QP.
 - s32Qp:** When **bAbsQp** is set to **true**, this parameter indicates the QP value of all macroblocks in an ROI; when **bAbsQp** is set to **false**, this parameter indicates the relative QP value of all macroblocks in an ROI.
 - stRect:** Specifies the coordinate position and size of the current ROI. The initial position along the horizontal and vertical axes must be within a picture and a multiple of 16. The height and width of the ROI must be a multiple of 16. The ROI must be within a picture.



- This MPI is an enhanced MPI. By default, the ROI function is not enabled. You must call this MPI to enable the ROI function.
- This MPI can be set after a VENC channel is created and before the channel is destroyed. This MPI takes effect after a next frame is encoded.
- You are advised to call this MPI after a VENC channel is created and before frames are encoded, to minimize times of calling this MPI during frame encoding.
- You are advised to call the [HI_MPI_VENC_GetRoiCfg](#) MPI to obtain the ROI configuration of the current channel before calling this MPI.

[Example]

```
HI_S32 SetRoi(HI_VOID)
{
    HI_S32 s32Ret = HI_FAILURE;
    VENC_ROI_CFG_S     stRoiCfg;
    VENC_CHN_ATTR_S   stChnAttr;
    HI_S32 index = 0;
    VENC_CHN VeChnId = 0;
    //...omit other thing
    s32Ret = HI_MPI_VENC_GetChnAttr(VeChnId, &stChnAttr);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_GetChnAttr err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }
    s32Ret = HI_MPI_VENC_GetRoiCfg(VeChnId, index, &stRoiCfg);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_GetRoiCfg err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }

    stSrcVencRoiCfg.bEnable          = HI_TRUE;
    stSrcVencRoiCfg.bAbsQp           = HI_TRUE;
    stSrcVencRoiCfg.s32Qp            = 10;
    stSrcVencRoiCfg.stRect.s32X      = 16;
    stSrcVencRoiCfg.stRect.s32Y      = 16;
    stSrcVencRoiCfg.stRect.u32Width  = 16;
    stSrcVencRoiCfg.stRect.u32Height = 16;
    stSrcVencRoiCfg.u32Index         = 0;
    s32Ret = HI_MPI_VENC_SetRoiCfg(VeChnId, &stSrcVencRoiCfg);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_SetRoiCfg err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }
}
```



```
    return HI_SUCCESS;  
}
```

[See Also]

None

HI_MPI_VENC_GetRoiCfg

[Description]

Obtains the ROI attribute of an H.264/H.265 channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetRoiCfg(VENC_CHN VeChn, HI_U32 u32Index,  
VENC_ROI_CFG_S *pstVencRoiCfg);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
u32Index	Index of an ROI on an H.264/H.265 channel	Input
pstVencRoiCfg	Configuration of the ROI	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to obtain the index ROI attribute of an H.264/H.265 channel.
- This MPI must be called after a VENC channel is created and before the channel is destroyed.
- You are advised to call this MPI after a VENC channel is created and before frames are encoded, to minimize times of calling this MPI during frame encoding.

[Example]

See [HI_MPI_VENC_GetRoiCfg](#).



[See Also]

None

HI_MPI_VENC_SetRoiBgFrameRate

[Description]

Sets the frame rate attribute of non-ROIs in an H.264/H.265 VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetRoiBgFrameRate(VENC_CHN VeChn, const  
VENC_ROIBG_FRAME_RATE_S *pstRoiBgFrmRate);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstRoiBgFrmRate	Frame rate control for non-ROIs	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpapi.a

[Note]

- This MPI is used to set the frame rate control parameters for non-ROIs in an H.264/H.265 VENC channel.
- Before calling this MPI, you must enable ROIs.
- This MPI is an advanced interface. The frame rate attributes of non-ROIs are not enabled by default. If you want to use these attributes, you need to enable them by calling this MPI.
- This MPI can be called after a VENC channel is created and before the channel is destroyed. If the MPI is called during encoding, the operation takes effect when the next frame is encoded.
- When the channel frame rate control attribute is configured, ensure that the input frame rate (**SrcFrmRate**) is greater than the output frame rate (**DstFrmRate**) and **DstFrmRate** is greater than or equal to 0, or ensure that both **SrcFrmRate** and **DstFrmRate** are set to **-1**.



- You are advised to call this MPI after creating a channel but before starting encoding. This reduces the number of times of calling this MPI during encoding.
- Before calling this MPI, you are advised to obtain the frame rate of the non-ROIs in the current VENC channel by calling [HI_MPI_VENC_GetRoiBgFrameRate](#).
- When the frame rate of the non-ROI is lower than that of the ROI, you are advised not to use the frame skipping reference mode or SVC-T encoding. In frame skipping reference mode or SVC-T encoding mode, if the frame rate of the non-ROI is lower than that of the ROI, the target frame rate of the non-ROI may be greater than the configured target frame rate because the non-ROI at the base layer cannot be encoded as Pskip blocks.

[Example]

```
HI_S32 SetRoiFrameRate(HI_VOID)
{
    HI_S32 s32Ret = HI_FAILURE;
    VENC_ROI_CFG_S     stRoiCfg;
    VENC_CHN_ATTR_S    stChnAttr;
    VENC_ROIBG_FRAME_RATE_S stRoiBgFrameRate;
    HI_S32 index = 0;
    VENC_CHN VeChnId = 0;

    //...omit other thing
    s32Ret = HI_MPI_VENC_GetChnAttr(VeChnId, &stChnAttr);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_GetChnAttr err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }

    s32Ret = HI_MPI_VENC_GetRoiCfg(VeChnId, index, &stRoiCfg);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_GetRoiCfg err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }

    stSrcVencRoiCfg.bEnable      = HI_TRUE;
    stSrcVencRoiCfg.bAbsQp       = HI_TRUE;
    stSrcVencRoiCfg.s32Qp        = 10;
    stSrcVencRoiCfg.stRect.s32X  = 16;
    stSrcVencRoiCfg.stRect.s32Y  = 16;
    stSrcVencRoiCfg.stRect.u32Width = 16;
    stSrcVencRoiCfg.stRect.u32Height = 16;
    stSrcVencRoiCfg.u32Index     = 0;
    s32Ret = HI_MPI_VENC_SetRoiCfg(VeChnId, &stSrcVencRoiCfg);
    if (HI_SUCCESS != s32Ret)
    {
```



```
        printf("HI_MPI_VENC_GetRoiCfg err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }

    stRoiBgFrameRate.s32SrcFrmRate = 30;
    stRoiBgFrameRate.s32DstFrmRate = 15;
    s32Ret = HI_MPI_VENC_SetRoiBgFrameRate (VeChnId, &stRoiBgFrameRate);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_SetRoiBgFrameRate err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }
    return HI_SUCCESS;
}
```

[See Also]

None

HI_MPI_VENC_GetRoiBgFrameRate

[Description]

Obtains the frame rate attribute of non-ROIs in an H.264/H.265 VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetRoiBgFrameRate(VENC_CHN VeChn,
VENC_ROIBG_FRAME_RATE_S *pstRoiBgFrmRate);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstRoiBgFrmRate	Frame rate of non-ROIs	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h



- Library file: libmpi.a

[Note]

- This MPI can be called after a VENC channel is created and before the channel is destroyed.
- You are advised to call this MPI after creating a channel but before starting encoding. This reduces the number of times of calling this MPI during encoding.

[Example]

See the example of [HI_MPI_VENC_SetRoiBgFrameRate](#).

[See Also]

None

HI_MPI_VENC_SetH264SliceSplit

[Description]

Sets the slice attribute of an H.264 channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetH264SliceSplit(VENC_CHN VeChn, const  
VENC_PARAM_H264_SLICE_SPLIT_S * pstSliceSplit);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstSliceSplit	Slice mode of an H.264 stream	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to set the slice attribute of an H.264 channel.
- The slice parameters include:



- **bSplitEnable:** Whether the current frame is sliced.
 - **u32SplitMode:** indicates a slice mode. When this parameter is set to **0**, the frame is sliced by byte; when this parameter is set to **1**, the frame is sliced by macroblock row. This parameter cannot be set to be greater than 1.
This parameter is effective only when **bSplitEnable** is set to **true**.
 - **u32SliceSize:** indicates the size of a slice. The meaning of this parameter varies with the value of **u32SplitMode**. When **u32SplitMode** is set to **0**, this parameter indicates the average number of byte in each slice. The encoder encodes a picture from the top down and from the left right in the raster order by macroblock. Each slice is segmented based on the value of this parameter. The actual size of a slice, however, may not be strictly consistent with the value set by users. An error may exist and the offset is always positive. This is because when the last slice is encoded, the remaining byte of the macroblocks are regarded as a complete slice though the number of bits is less than the value of this parameter. When **u32SplitMode** is set to **1**, this parameter indicates the number of macroblock rows occupied by a slice. The unit of this parameter is row of macroblocks. The remaining rows of macroblocks are regarded as a slice though the number of rows of macroblocks is less than the value of this parameter.
- This MPI is an enhanced interface. You can call it as required. However, you are advised not to call this MPI. **bSplitEnable** is set to **false by default**.
 - This MPI can be set after a VENC channel is created and before the channel is destroyed. This MPI takes effect after a next frame is encoded.
 - You are advised to call this MPI after a VENC channel is created and before frames are encoded, to minimize times of calling this MPI during frame encoding.
 - You are advised to call [HI_MPI_VENC_GetH264SliceSplit](#) to obtain the slice configuration of the current channel before calling this MPI.

[Example]

```
HI_S32 SetSliceSplit(HI_VOID)
{
    HI_S32 s32Ret = HI_FAILURE;
    VENC_PARAM_H264_SLICE_SPLIT_S stSlice;
    VENC_CHN VeChnId = 0;

    //...omit other thing

    s32Ret = HI_MPI_VENC_GetH264SliceSplit(VeChnId, &stSlice);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_GetH264SliceSplit err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }

    stSlice.bSplitEnable = HI_TRUE;
    stSlice.u32SplitMode = 0;
    stSlice.u32SliceSize = 2048;
    s32Ret = HI_MPI_VENC_SetH264SliceSplit(VeChnId, &stSlice);
    if (HI_SUCCESS != s32Ret)
```



```
{  
    printf("HI_MPI_VENC_SetH264SliceSplit err 0x%x\n", s32Ret);  
    return HI_FAILURE;  
}  
  
return HI_SUCCESS;  
}
```

[See Also]

None

HI_MPI_VENC_SetH264SliceSplit

[Description]

Obtains the slice attribute of an H.264 channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetH264SliceSplit(VENC_CHN VeChn,  
VENC_PARAM_H264_SLICE_SPLIT_S *pstSliceSplit);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstSliceSplit	Slice mode of an H.264 stream	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to set the slice attribute of an H.264 channel.
- This MPI must be called after a VENC channel is created and before the channel is destroyed.



- You are advised to call this MPI after a VENC channel is created and before frames are encoded, to minimize times of calling this MPI during frame encoding.

[Example]

See [HI_MPI_VENC_SetH264SliceSplit](#).

[See Also]

None

HI_MPI_VENC_SetH264InterPred

[Description]

Sets the inter-prediction attribute of an H.264 channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetH264InterPred(VENC_CHN VeChn, const  
VENC_PARAM_H264_INTER_PRED_S *pstH264InterPred);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstH264InterPred	Inter-frame prediction attribute of an H.264 channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Difference]

The search widow size of the H.264 encoder varies depending on chip type. See [Table 6-8](#).

Table 6-8 Search widow size

Chip	MIN_HW	MAX_HW	MIN_VW	MAX_VW
Hi3516A	0	11	0	3
Hi3518E V200	0	11	0	3



MIN_HW, MAX_HW, MIN_VW, and MAX_VW indicate the minimum horizontal width, maximum horizontal width, minimum vertical width, and maximum vertical width respectively.

- The search window size of the H.264 encoder also varies depending on resolution. See [Table 6-9](#).

Table 6-9 Default sizes of the search window of the H.264 encoder under various resolutions for the Hi3516A/Hi3518E V200

PicWidth (Pixel)	u32HWSize	u32VWSize
≤ 720	5	2
(720, 1280]	7	3
(1280, 1920]	11	2
(1920, 2592]	11	1

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to set the inter-prediction configuration of an H.264 channel.
- The inter-prediction parameters include:
 - **u32HWSize**: size of a horizontal search window during inter-prediction
 - **u32VWSize**: size of a vertical search window during inter-prediction
 - **bInter16x16PredEn**: 16x16 inter-prediction enable
 - **bInter16x8PredEn**: 16x8 inter-prediction enable
 - **bInter8x16PredEn**: 8x16 inter-prediction enable
 - **bInter8x8PredEn**: 8x8 inter-prediction enable
 - **bExedgeEn**: inter-prediction edge extension enable. When inter-prediction is performed on the macroblocks on the boundary of a picture, the search window may exceed the picture boundary. In this case, you can enable edge extension to extend the picture edges for inter-prediction. If edge extension is disabled, inter-prediction is not performed on the part of the search window beyond the picture.
- This MPI is an advanced interface. You can call it as required. However, you are advised not to call this MPI. The inter-prediction attribute of an H.264 channel has default values. By default, the size of the search window varies with the picture resolution and the other five flags are enabled.
- The prediction enable of **bInter16x16PredEn** can only be enabled.
- This MPI can be set after a VENC channel is created and before the channel is destroyed. This MPI takes effect after a next I frame is encoded.
- You are advised to call this MPI after a VENC channel is created and before frames are encoded, to minimize times of calling this MPI during frame encoding.
- You are advised to call the [HI_MPI_VENC_GetH264InterPred](#) MPI to obtain the InterPred configuration of the current channel before calling this MPI.



- The limitations on the size of a horizontal search window are as follows:
 - The horizontal window width must meet the following condition: **MIN_HW** ≤ horizontal window width ≤ **MAX_HW**
 - The picture width (in macroblock) must be two macroblocks greater than the actual width of a search window. The actual width (in macroblock) of a search window is calculated as follows: Width of a search window = $(1 + \text{u32HWSize}) \times 2 + 1$. If the width of the search window is set to 10 pixels, the actual width of a picture must be greater than 384. For example, the actual width can be 386 pixels. The width is calculated as follows: $[(1 + 10) \times 2 + 1 + 1] \times 16 = 384$.
- The limitations on the size of a vertical search window are as follows:
 - The vertical window width must meet the following condition: **MIN_VW** ≤ vertical window width ≤ **MAX_VW**
 - The picture height (in macroblock) must be one macroblock greater than the actual height of a search window. Actual height of a search window (in macroblock) is calculated as follows: Height of a search window = $(1 + \text{u32VWSIZE}) \times 2 + 1$. If the height of the search window is set to 4 pixels, the actual picture height must be greater than 176 pixels. For example, the height can be 178 pixels. The height is calculated as follows: $[(1 + 4) \times 2 + 1] \times 16 = 176$.
 - For the Hi3516A/Hi3518E V200, if the picture width is less than or equal to 720 pixels, the width of the vertical search window must be less than or equal to 2 pixels. If the picture width is greater than 720 pixels but is less than and equal to 1280 pixels, the width of the vertical search window must be less than or equal to 3 pixels. If the picture width is greater than 1280 pixels but is less than and equal to 1920 pixels, the width of the vertical search window must be less than or equal to 2 pixels. If the picture width is greater than 1920 pixels, the width of the vertical search window must be less than or equal to 1 pixel.
- **Table 6-9** describes the typical configurations of the search window for the Hi3516A and Hi3518E V200. The configurations must meet the preceding limitations on the sizes of the horizontal and vertical search windows. For some pictures with low resolutions, if the default configurations do not meet the limitations on the sizes of the horizontal and vertical search windows, the sizes that are calculated based on limitations by taking the minimum resolution (160 x 64) supported by the chip as an example are used as the default sizes of the horizontal and vertical search windows. For example, the default size of the horizontal search window is 5 for a picture (180 x 288) of the Hi3516A. According to the preceding limitations, the minimum picture width is calculated as follows: Minimum picture width = $[(1 + 5) \times 2 + 1 + 1] \times 16 = 224$. The actual picture width must be greater than 224, which does not meet the requirements. In this case, the default size of the horizontal search window needs to be changed to 2. The same rule applies to the size of the vertical search window.

[Example]

```
HI_S32 SetInterPred(HI_VOID)
{
    HI_S32 s32Ret = HI_FAILURE;
    VENC_PARAM_H264_INTER_PRED_S stInterPred;
    VENC_CHN VeChnId = 0;
    //...omit other thing
    s32Ret = HI_MPI_VENC_GetH264InterPred(VeChnId, &stInterPred);
    if (HI_SUCCESS != s32Ret)
    {
```



```
    printf("HI_MPI_VENC_GetH264InterPred err 0x%x\n", s32Ret);
    return HI_FAILURE;
}

stInterPred.bExtedgeEn      = HI_TRUE;
stInterPred.bInter16x16PredEn = HI_TRUE;
stInterPred.bInter16x8PredEn = HI_FALSE;
stInterPred.bInter8x16PredEn = HI_FALSE;
stInterPred.bInter8x8PredEn = HI_FALSE;
stInterPred.u32HWSIZE       = 4;
stInterPred.u32VWSIZE       = 2;
s32Ret = HI_MPI_VENC_SetH264InterPred(VeChnId, &stInterPred);
if (HI_SUCCESS != s32Ret)
{
    printf("HI_MPI_VENC_SetH264InterPred err 0x%x\n", s32Ret);
    return HI_FAILURE;
}

return HI_SUCCESS;
}
```

[See Also]

None

HI_MPI_VENC_GetH264InterPred

[Description]

Obtains the inter-prediction attribute of an H.264 channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetH264InterPred(VENC_CHN VeChn,
VENC_PARAM_H264_INTER_PRED_S *pstH264InterPred);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstH264InterPred	Inter-frame prediction attribute of an H.264 channel	Output

[Return Value]

Return Value	Description
0	Success



Return Value	Description
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to obtain the inter-prediction mode of an H.264 channel.
- This MPI must be called after a VENC channel is created and before the channel is destroyed.
- You are advised to call this MPI after a VENC channel is created and before frames are encoded, to minimize times of calling this MPI during frame encoding.

[Example]

See [HI_MPI_VENC_SetH264InterPred](#).

[See Also]

None

HI_MPI_VENC_SetH264IntraPred

[Description]

Sets the intra-prediction attribute of an H.264 channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetH264IntraPred(VENC_CHN VeChn, const  
VENC_PARAM_H264_INTRA_PRED_S *pstH264IntraPred);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstH264IntraPred	Intra-frame prediction attribute of an H.264 channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For



Return Value	Description
	details, see section 6.5 "Error Codes."

[Difference]

The Hi3516A/Hi3518E V200 supports IP camera prediction. You can enable or disable this function. This function is enabled by default.

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to set the intra-prediction configuration of an H.264 channel.
- The intra-prediction parameters include:
 - **bIntra16x16PredEn**: a flag used to enable 16x16 intra-prediction.
 - **bIntraNxNPredEn**: a flag used to enable NxN intra-prediction. Here, NxN indicates 4x4 or 8x8.
 - **bIpemEn**: a flag used to enable IPCM intra-prediction.
 - **constrained_intra_pred_flag**: For details, see the H.264 protocol.
- This MPI is an advanced interface. You can call it as required. However, you are advised not to call this MPI. The intra-prediction attribute of an H.264 channel has default values. By default, **bIntra16x16PredEn**, **bIntraNxNPredEn**, and **bIPCPredEn** are enabled and **constrained_intra_pred_flag** is set to **0**. The default values vary depending on chip type.
- Either **bIntra16x16PredEn** or **bIntraNxNPredEn** must be enabled. The system does not disable the two flags at the same time.
- This MPI can be set after a VENC channel is created and before the channel is destroyed. This MPI takes effect after a second I frame is encoded.
- You are advised to call this MPI after a VENC channel is created and before frames are encoded, to minimize times of calling this MPI during frame encoding.
- You are advised to call the [HI_MPI_VENC_GetH264IntraPred](#) MPI to obtain the intra-prediction configuration of the current channel before calling this MPI.

[Example]

```
HI_S32 SetIntraPred(HI_VOID)
{
    HI_S32 s32Ret = HI_FAILURE;
    VENC_PARAM_H264_INTRA_PRED_S stIntraPred;
    VENC_CHN VeChnId = 0;

    //...omit other thing

    s32Ret = HI_MPI_VENC_GetH264IntraPred(VeChnId, &stIntraPred);
    if (HI_SUCCESS != s32Ret)
    {

```



```
    printf("HI_MPI_VENC_GetH264IntraPred err 0x%x\n", s32Ret);
    return HI_FAILURE;
}

stIntraPred.bIntra16x16PredEn = HI_TRUE;
stIntraPred.bIntraNxNPredEn = HI_FALSE;
stIntraPred.bIpclEn = HI_FALSE;
stIntraPred.constrained_intra_pred_flag = 0;
s32Ret = HI_MPI_VENC_SetH264IntraPred(VeChnId, &stIntraPred);
if (HI_SUCCESS != s32Ret)
{
    printf("HI_MPI_VENC_SetH264IntraPred err 0x%x\n", s32Ret);
    return HI_FAILURE;
}

return HI_SUCCESS;
}
```

[See Also]

None

HI_MPI_VENC_GetH264IntraPred

[Description]

Obtains the intra-prediction attribute of an H.264 channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetH264IntraPred(VENC_CHN VeChn,
VENC_PARAM_H264_INTRA_PRED_S *pstH264IntraPred);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstH264IntraPred	Intra-frame prediction attribute of an H.264 channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details,



Return Value	Description
	see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to obtain the intra-prediction mode of an H.264 channel.
- This MPI must be called after a VENC channel is created and before the channel is destroyed.
- You are advised to call this MPI after a VENC channel is created and before frames are encoded, to minimize times of calling this MPI during frame encoding.

[Example]

See [HI_MPI_VENC_SetH264IntraPred](#).

[See Also]

None

HI_MPI_VENC_SetH264Trans

[Description]

Sets the transformation and quantization attribute of an H.264 channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetH264Trans(VENC_CHN VeChn, const  
VENC_PARAM_H264_TRANS_S *pstH264Trans);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstH264Trans	Transformation and quantization attribute of an H.264 channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."



[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to set the conversion and quantization configurations of an H.264 channel.
- The transformation and quantization parameters include:
 - **u32IntraTransMode:** transformation attribute of an intra-macroblock. When this parameter is set to **0**, 4x4 and 8x8 transformations are supported for the intra-macroblock; when this parameter is set to **1**, only 4x4 transformation is supported for the intra-macroblock; when this parameter is set to **2**, only 8x8 transformation is supported for the intra-macroblock. 4x4 transformation (this parameter is set to **1**) is available to the H.264 baseline profile and main profile. 8x8 transformation is available only to the H.264 high profile.
 - **u32InterTransMode:** transformation attribute of an inter-macroblock. When this parameter is set to **0**, 4x4 and 8x8 transformations are supported for the intra-macroblock; when this parameter is set to **1**, only 4x4 transformation is supported for the intra-macroblock; when this parameter is set to **2**, only 8x8 transformation is supported for the intra-macroblock. 4x4 transformation (this parameter is set to **1**) is available to the H.264 baseline profile and main profile. 8x8 transformation is available only to the H.264 high profile.
 - **bScalingListValid:** Whether **InterScalingList8x8** and **IntraScalingList8x8** are valid. In the baseline and main profiles, the H.264 protocol does not support 8x8 transformation. Therefore, this parameter does not take effect. In the high profile, H.264 channels allow you to deliver a quantization table. If this parameter is set to **false**, the system uses the quantization table specified by the H.264 protocol. Otherwise, the quantization table delivered by users is used.
 - **InterScalingList8x8:** You provide a quantization table by using this array when 8x8 inter-prediction is performed on an inter-macroblock.
 - **IntraScalingList8x8:** You provide a quantization table by using this array when 8x8 transformation is performed on an intra-macroblock.
 - **chroma_qp_index_offset:** For details, see the H.264 protocol.
- This MPI is an advanced interface. You can call it as required. However, you are advised not to call this MPI. The transformation and quantization attribute of an H.264 channel have default values. By default, the system sets the parameters based on the profile type.

Table 6-10 Default trans parameter values for different profile types

Profile	u32IntraTransMode	u32InterTransMode	bScalingListValid
Baseline/main profile	1	1	false
High profile	0	0	false

- This MPI can be set after a VENC channel is created and before the channel is destroyed. This MPI takes effect after the next I frame is encoded.



- You are advised to call this MPI after a VENC channel is created and before frames are encoded. This minimizes the times of calling this MPI during frame encoding.
- You are advised to call the [HI_MPI_VENC_GetH264Trans](#) MPI to obtain the trans configuration of the current channel before calling this MPI.

[Example]

```
HI_S32 SetTrans(HI_VOID)
{
    HI_S32 s32Ret = HI_FAILURE;
    VENC_PARAM_H264_TRANS_S stTrans;
    VENC_CHN VeChnId = 0;

    //...omit other thing

    s32Ret = HI_MPI_VENC_GetH264Trans(VeChnId, &stTrans);
    if (HI_SUCCESS != s32Ret)
    {
        printf("stTrans err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }

    stTrans.u32IntraTransMode      = 2;
    stTrans.u32InterTransMode     = 2;
    stTrans.bScalingListValid     = HI_FALSE;
    stTrans.chroma_qp_index_offset = 2;
    s32Ret = HI_MPI_VENC_SetH264Trans(VeChnId, &stTrans);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_SetH264Trans err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }

    return HI_SUCCESS;
}
```

[See Also]

None

[HI_MPI_VENC_GetH264Trans](#)

[Description]

Obtains the transformation and quantization attribute of an H.264 channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetH264Trans(VENC_CHN VeChn, VENC\_PARAM\_H264\_TRANS\_S
```



```
*pstH264Trans);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstH264Trans	Transformation and quantization attribute of an H.264 channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to obtain the transformation and quantization configurations of an H.264 channel.
- This MPI must be called after a VENC channel is created and before the channel is destroyed.
- You are advised to call this MPI after a VENC channel is created and before frames are encoded, to minimize times of calling this MPI during frame encoding.

[Example]

See [HI_MPI_VENC_SetH264Trans](#).

[See Also]

None

HI_MPI_VENC_SetH264Entropy

[Description]

Sets the entropy encoding mode for an H.264 channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetH264Entropy(VENC_CHN VeChn, const  
VENC_PARAM_H264_ENTROPY_S *pstH264EntropyEnc);
```



[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstH264EntropyEnc	Entropy encoding mode for an H.264 channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to set the entropy encoding configuration of an H.264 channel.
- The entropy encoding parameters include:
 - u32EntropyEncModeI**: the entropy mode of I frame. When this parameter is set to **0**, the I frame is encoded in **context adaptive variable length coding (CAVLC)** mode; when this parameter is set to **1**, the I frame is encoded in context-based adaptive binary arithmetic coding (CABAC) mode.
 - u32EntropyEncModeP**: the entropy mode of P frame. When this parameter is set to **0**, the P frame is encoded in CAVLC mode; when this parameter is set to **1**, the P frame is encoded in CABAC mode.
 - cabac_stuff_en**: enables CABAC encoding stuff. By default, it is set to **0**. For details, see the H.264 protocol.
 - Cabac_init_idc**: indexes CABAC initialization. By default, it is set to **0**. For details, see the H.264 protocol.
- The entropy modes of I frame and P frame can be set independently.
- The baseline profile does not support CABAC encoding but supports CAVLC encoding. The main profile and high profile support both the CABAC and CAVLC encoding modes.
- Compared with the CAVLC encoding, CABAC encoding requires greater calculation amount but generates fewer streams. If system performance is low, you are advised to use CAVLC encoding for I frame and CABAC encoding for P frame.
- This MPI is an advanced interface. You can call it as required. However, you are advised not to call this MPI. The entropy encoding mode for an H.264 channel has a default value. By default, the system sets parameters based on the profile type.



Table 6-11 Default entropy encoding parameter values for different profile types

Profile	u32EntropyEncModeI	u32EntropyEncModeP
Baseline	0	0
Main profile/High profile	1	1

- This MPI can be set after a VENC channel is created and before the channel is destroyed. This MPI takes effect after a next I frame is encoded.
- You are advised to call this MPI after a VENC channel is created and before frames are encoded, to minimize times of calling this MPI during frame encoding.
- You are advised to call the [HI_MPI_VENC_GetH264Entropy](#) MPI to obtain the entropy configuration of the current channel before calling this MPI.

[Example]

```
HI_S32 SetEntropy(HI_VOID)
{
    HI_S32 s32Ret = HI_FAILURE;
    VENC_PARAM_H264_ENTROPY_S stEntropy;
    VENC_CHN VeChnId = 0;

    //...omit other thing

    s32Ret = HI_MPI_VENC_GetH264Entropy(VeChnId, &stTrans);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_GetH264Entropy err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }

    stEntropy.u32EntropyEncModeI = 0;
    stEntropy.u32EntropyEncModeP = 0;
    stEntropy.cabac_stuff_en = 0;
    stEntropy.Cabac_init_idc = 0;
    s32Ret = HI_MPI_VENC_SetH264Entropy(VeChnId, &stEntropy);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_SetH264Entropy err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }

    return HI_SUCCESS;
}
```

[See Also]



None

HI_MPI_VENC_GetH264Entropy

[Description]

Obtains the entropy encoding attribute of an H.264 channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetH264Entropy(VENC_CHN VeChn,  
VENC_PARAM_H264_ENTROPY_S *pstH264EntropyEnc);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstH264EntropyEnc	Entropy encoding mode of an H.264 channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpapi.a

[Note]

- This MPI is used to obtain the entropy encoding configuration of an H.264 channel.
- This MPI must be called after a VENC channel is created and before the channel is destroyed.
- You are advised to call this MPI after a VENC channel is created and before frames are encoded, to minimize times of calling this MPI during frame encoding.

[Example]

See [HI_MPI_VENC_SetH264Entropy](#).

[See Also]

None



HI_MPI_VENC_SetH264Poc

[Description]

Sets the picture order count (POC) type of an H.264 channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetH264Poc(VENC_CHN VeChn, const VENC_PARAM_H264_POC_S *pstH264Poc);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstH264Poc	POC type of an H.264 channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to set the POC configuration of an H.264 channel.
- The POC attribute mainly indicates the POC type of H.264 streams. For details, see the H.264 protocol.
- There are three POC types, which are specified by **pic_order_cnt_type** (the value can be set to **0**, **1**, or **2**). By default, it is set to **2**.
- This MPI can be set after a VENC channel is created and before the channel is destroyed. This MPI takes effect after a next I frame is encoded.
- You are advised to call this MPI after a VENC channel is created and before frames are encoded, to minimize times of calling this MPI during frame encoding.
- You are advised to call the [HI_MPI_VENC_GetH264Poc](#) MPI to obtain the POC configuration of the current channel before calling this MPI.

[Example]

```
HI_S32 SetPoc(HI_VOID)
{
```



```
HI_S32 s32Ret = HI_FAILURE;
VENC_PARAM_H264_POC_S           stPoc;
VENC_CHN VeChnId = 0;

//...omit other thing

s32Ret = HI_MPI_VENC_GetH264Poc(VeChnId, &stPoc);
if (HI_SUCCESS != s32Ret)
{
    printf("HI_MPI_VENC_GetH264Poc err 0x%x\n", s32Ret);
    return HI_FAILURE;
}

stPoc.pic_order_cnt_type      = 0;
s32Ret = HI_MPI_VENC_SetH264Poc(VeChnId, &stPoc);
if (HI_SUCCESS != s32Ret)
{
    printf("HI_MPI_VENC_SetH264Poc err 0x%x\n", s32Ret);
    return HI_FAILURE;
}

return HI_SUCCESS;
}
```

[See Also]

None

HI_MPI_VENC_GetH264Poc

[Description]

Obtains the POC type of an H.264 channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetH264Poc(VENC_CHN VeChn, VENC_PARAM_H264_POC_S
*pstH264Poc);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstH264Poc	POC type of an H.264 channel	Output

[Return Value]



Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to obtain the POC configuration of an H.264 channel.
- This MPI must be called after a VENC channel is created and before the channel is destroyed.
- You are advised to call this MPI after a VENC channel is created and before frames are encoded, to minimize times of calling this MPI during frame encoding.

[Example]

See [HI_MPI_VENC_SetH264Poc](#).

[See Also]

None

HI_MPI_VENC_SetH264Dbblk

[Description]

Sets the de-blocking type of an H.264 channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetH264Dbblk(VENC_CHN VeChn, const  
VENC_PARAM_H264_DBLK_S *pstH264Dbblk);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstH264Dbblk	De-blocking type of an H.264 parameter	Input

[Return Value]

Return Value	Description
0	Success



Return Value	Description
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to set the de-blocking configuration of an H.264 channel.
- The de-blocking parameters include:
 - **disable_deblocking_filter_idc**: For details, see the H.264 protocol.
 - **slice_alpha_c0_offset_div2**: For details, see the H.264 protocol.
 - **slice_beta_offset_div2**: For details, see the H.264 protocol.
- The de-blocking function is enabled by default, that is, **disable_deblocking_filter_idc**, **slice_alpha_c0_offset_div2**, and **slice_beta_offset_div2** are **0** by default.
- To disable the de-blocking function, set **disable_deblocking_filter_idc** to **1**.
- This MPI can be set after a VENC channel is created and before the channel is destroyed. This MPI takes effect after a next I frame is encoded.
- You are advised to call this MPI after a VENC channel is created and before frames are encoded, to minimize times of calling this MPI during frame encoding.
- You are advised to call the [HI_MPI_VENC_GetH264Dbblk](#) MPI to obtain the de-blocking configuration of the current channel before calling this MPI.

[Example]

```
HI_S32 SetDbblk(HI_VOID)
{
    HI_S32 s32Ret = HI_FAILURE;
    VENC_PARAM_H264_DBULK_S          stDbblk;
    VENC_CHN VeChnId = 0;

    //...omit other thing

    s32Ret = HI_MPI_VENC_GetH264Dbblk(VeChnId, &stDbblk);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_GetH264Dbblk err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }

    stDbblk.disable_deblocking_filter_idc = 0;
    stDbblk.slice_alpha_c0_offset_div2   = 6;
    stDbblk.slice_beta_offset_div2      = 5;
```



```
s32Ret = HI_MPI_VENC_SetH264Dbblk(VeChnId, &stDbblk);  
if (HI_SUCCESS != s32Ret)  
{  
    printf("HI_MPI_VENC_SetH264Dbblk err 0x%x\n", s32Ret);  
    return HI_FAILURE;  
}  
  
return HI_SUCCESS;  
}
```

[See Also]

None

HI_MPI_VENC_GetH264Dbblk

[Description]

Obtains the de-blocking type of an H.264 channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetH264Dbblk(VENC_CHN VeChn, VENC_PARAM_H264_DBULK_S  
*pstH264Dbblk);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstH264Dbblk	De-blocking type of an H.264 channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to obtain the de-blocking configuration of an H.264 channel.



- This MPI must be called after a VENC channel is created and before the channel is destroyed.
- You are advised to call this MPI after a VENC channel is created and before frames are encoded, to minimize times of calling this MPI during frame encoding.

[Example]

See [HI_MPI_VENC_SetH264Dbblk](#).

[See Also]

None

HI_MPI_VENC_SetH264Vui

[Description]

Sets the VUI parameters of an H.264 VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetH264Vui(VENC_CHN VeChn, const VENC_PARAM_H264_VUI_S *pstH264Vui);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstH264Vui	VUI of an H.264 VENC channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to set the VUI configuration of an H.264 channel.
- This MPI can be set after a VENC channel is created and before the channel is destroyed. This MPI takes effect after a next I frame is encoded.
- You are advised to call this MPI after a VENC channel is created and before frames are encoded, to minimize times of calling this MPI during frame encoding.



- You are advised to call [HI_MPI_VENC_GetH264Vui](#) to obtain the VUI configuration of the current channel before calling this MPI.

[Example]

```
HI_S32 SetVui(HI_VOID)
{
    HI_S32 s32Ret = HI_FAILURE;
    VENC_PARAM_H264_VUI_S          stVui;
    VENC_CHN VeChnId = 0;

    //...omit other thing

    s32Ret = HI_MPI_VENC_GetH264Vui(VeChnId, &stVui);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_GetH264Vui err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }

    stVui.timing_info_present_flag = 1;
    s32Ret = HI_MPI_VENC_SetH264Vui(VeChnId, &stVui);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_SetH264Vui err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }

    return HI_SUCCESS;
}
```

[See Also]

None

[HI_MPI_VENC_GetH264Vui](#)

[Description]

Obtains the VUI parameter settings of an H.264 VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetH264Vui(VENC_CHN VeChn, VENC_PARAM_H264_VUI_S
*pstH264Vui);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID	Input



Parameter	Description	Input/Output
	Value range: [0, VENC_MAX_CHN_NUM)	
pstH264Vui	VUI of an H.264 VENC channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to obtain the VUI configuration of an H.264 channel.
- This MPI must be called after a VENC channel is created and before the channel is destroyed.
- You are advised to call this MPI after a VENC channel is created and before frames are encoded, to minimize times of calling this MPI during frame encoding.

[Example]

See [HI_MPI_VENC_SetH264Vui](#).

[See Also]

None

HI_MPI_VENC_SetH265Vui

[Description]

Sets the VUI parameters of an H.265 VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetH265Vui(VENC_CHN VeChn, const VENC_PARAM_H265_VUI_S *pstH265Vui);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input



Parameter	Description	Input/Output
pstH265Vui	VUI parameter of an H.265 VENC channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to configure the VUI parameters of an H.265 encoding channel.
- This MPI can be called after a VENC channel is created and before the channel is destroyed. If this MPI is called during encoding, the operation takes effect only when the next I frame is encoded.
- You are advised to call this MPI after creating a channel but before starting encoding. This reduces the number of times of calling this MPI during encoding.
- Before calling this MPI, you are advised to call [HI_MPI_VENC_GetH265Vui](#) to obtain the VUI parameter settings of the current VENC channel.

[Example]

```
HI_S32 SetVui(HI_VOID)
{
    HI_S32 s32Ret = HI_FAILURE;
    VENC_PARAM_H265_VUI_S           stVui;
    VENC_CHN VeChnId = 0;

    //...omit other thing

    s32Ret = HI_MPI_VENC_GetH265Vui(VeChnId, &stVui);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_GetH265Vui err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }

    stVui.timing_info_present_flag = 1;
    s32Ret = HI_MPI_VENC_SetH265Vui(VeChnId, &stVui);
```



```
if (HI_SUCCESS != s32Ret)
{
    printf("HI_MPI_VENC_SetH264Vui err 0x%x\n", s32Ret);
    return HI_FAILURE;
}
return HI_SUCCESS;
}
```

[See Also]

None

HI_MPI_VENC_GetH265Vui

[Description]

Obtains the VUI parameter settings of an H.265 VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetH265Vui (VENC_CHN VeChn, VENC_PARAM_H265_VUI_S
*pstH265Vui);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstH264Vui	VUI attribute of an H.265 VENC channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to obtain the VUI parameter settings of an H.265 VENC channel.
- This MPI can be called after a VENC channel is created and before the channel is destroyed.



- You are advised to call this MPI after creating a channel but before starting encoding. This reduces the number of times of calling this MPI during encoding.

[Example]

See the example of [HI_MPI_VENC_SetH265Vui](#).

[See Also]

None

HI_MPI_VENC_SetJpegParam

[Description]

Sets the advanced parameters of a JPEG channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetJpegParam(VENC_CHN VeChn, const VENC_PARAM_JPEG_S
*pstJpegParam);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstJpegParam	Advanced parameters of a JPEG channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to set the advanced parameters of a JPEG channel.
- The advanced parameters include:
 - **u32Qfactor:** The value of a quantization factor ranges from **1** to **99**. When this parameter is set to a larger value, the quantization coefficient in the quantization table becomes smaller, the obtained picture quality becomes better, and the encoding compression rate becomes lower; when this parameter is set to a smaller value, the quantization coefficient in the quantization table becomes larger, the obtained picture



quality becomes poorer, and the encoding compression rate becomes higher. For details about the relationship between this parameter and the quantization table, see the related RFC2435 standard.

- **u8YQt[64], u8CbQt[64], and u8CrQt[64]**: These parameters correspond to three quantization tables. You can set a quantization table by using these three parameters.
- **u32MCUPerECS**: indicates the number of minimum coded units (MCUs) of each ECS. When this parameter is set to **0**, all MCUs in the current frame are encoded into an ECS. The value range is $(\text{picwidth} + 15) \gg 4 \times (\text{picheight} + 15) \gg 4 \times 2$.
- To use a quantization table, set **Qfactor** to **50** when setting the quantization table.
- This MPI can be set after a VENC channel is created and before the channel is destroyed. This MPI takes effect after a next I frame is encoded.
- You are advised to call this MPI after a VENC channel is created and before frames are encoded, to minimize times of calling this MPI during frame encoding.
- You are advised to call the [HI_MPI_VENC_GetJpegParam](#) MPI to obtain the JPEG configuration of the current channel before calling this MPI.

[Example]

```
HI_S32 SetJpegParam(HI_VOID)
{
    HI_S32 s32Ret = HI_FAILURE;
    VENC_PARAM_JPEG_S stParamJpeg;
    VENC_CHN VeChnId = 0;

    //...omit other thing

    s32Ret = HI_MPI_VENC_GetJpegParam(VeChnId, &stParamJpeg);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_GetJpegParam err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }

    stParamJpeg.u32MCUPerECS = 100;
    for (i = 0; i < 64; i++)
    {
        stParamJpeg.u8YQt[i] = 16;
        stParamJpeg.u8CbQt[i] = 17;
        stParamJpeg.u8CrQt[i] = 18;
    }
    s32Ret = HI_MPI_VENC_SetJpegParam(VeChnId, &stParamJpeg);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_SetJpegParam err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }
    return HI_SUCCESS;
}
```



}

[See Also]

None

HI_MPI_VENC_GetJpegParam

[Description]

Obtains the advanced parameters of a JPEG channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetJpegParam(VENC_CHN VeChn, VENC_PARAM_JPEG_S  
*pstJpegParam);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstJpegParam	Advanced parameters of a JPEG channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to obtain the configurations of advanced parameters of a JPEG channel.
- This MPI must be called after a VENC channel is created and before the channel is destroyed.
- You are advised to call this MPI after a VENC channel is created and before frames are encoded, to minimize times of calling this MPI during frame encoding.

[Example]

See [HI_MPI_VENC_SetJpegParam](#).

[See Also]

None



HI_MPI_VENC_SetMjpegParam

[Description]

Sets the advanced parameters of an MJPEG channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetMjpegParam(VENC_CHN VeChn, const VENC_PARAM_MJPEG_S *pstMjpegParam);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstMjpegParam	Advanced parameter of an MJPEG channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to set the advanced parameters of an MJPEG channel.
- The following describes four advanced parameters:
 - u8YQt[64], u8CbQt[64], and u8CrQt[64]: The three parameters correspond to three quantization tables. You can set quantization tables by setting the parameters.
 - u32MCUPerECS: It indicates the number of MCUs in each ECS. When u32MCUPerECS is set to 0, all MCUs of the current frame are encoded as an ECS. The minimum value of u32MCUPerECS is 0, and the maximum value of u32MCUPerECS cannot be greater than {[picwidth + 15] >> 4] x {[picheight + 15] >> 4} x 2.
- To use the user-defined quantization tables, set **Qfactor** to **50** when setting quantization tables.
- This MPI must be called after an MJPEG channel is created and before the channel is destroyed. If the MPI is called during encoding, the operation takes effect only after the next frame is encoded.
- You are advised to call this MPI after creating a channel and before starting encoding. This reduces the number of times of calling this MPI during encoding.



- Before calling this MPI, you are advised to call [HI_MPI_VENC_GetMjpegParam](#) to obtain the MjpegParam configurations of the current encoding channel.

[Example]

```
HI_S32 SetMjpegParam(HI_VOID)
{
    HI_S32 s32Ret = HI_FAILURE;
    VENC_PARAM_MJPEG_S stParamMjpeg;
    VENC_CHN VeChnId = 0;
    //...omit other thing

    s32Ret = HI_MPI_VENC_GetMjpegParam(VeChnId, &stParamMjpeg);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_GetMjpegParam err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }

    stParamMjpeg.u32MCUPerECS = 100;
    for (i = 0; i < 64; i++)
    {
        stParamMjpeg.u8YQt[i] = 16;
        stParamMjpeg.u8CbQt[i] = 17;
        stParamMjpeg.u8CrQt[i] = 18;
    }
    s32Ret = HI_MPI_VENC_SetMjpegParam(VeChnId, &stParamMjpeg);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_SetMjpegParam err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }

    return HI_SUCCESS;
}
```

[See Also]

None

HI_MPI_VENC_GetMjpegParam

[Description]

Obtains the advanced parameters of an MJPEG channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetMjpegParam(VENC_CHN VeChn, VENC_PARAM_MJPEG_S
```



```
*pstMjpegParam);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstMjpegParam	Advanced parameters of an MJPEG channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpapi.a

[Note]

- This MPI is used to obtain the configurations of advanced parameters of an MJPEG channel.
- This MPI must be called after an MJPEG channel is created and before the channel is destroyed.
- You are advised to call this MPI after creating a channel and before starting encoding. This reduces the number of times of calling this MPI during encoding.

[Example]

See the example of [HI_MPI_VENC_SetMjpegParam](#).

[See Also]

None

HI_MPI_VENC_SetFrameRate

[Description]

Sets the frame rate control attribute of a VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetFrameRate(VENC_CHN VeChn, const VENC_FRAME_RATE_S  
*pstFrameRate);
```

[Parameter]



Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstFrameRate	Pointer to frame rate control attribute of a VENC channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Request]

- Header files: hi_comm_venc.h, mpi_venc.h.
- Library file: libmpi.a

[Note]

- The frame rate control attribute of a VENC channel includes the input frame rate **SrcFrmRate** and output frame rate **DstFrmRate**.
- If you set the attributes of a VENC channel that does not exist, an error code indicating failure is returned.
- If **pstFrameRate** is not set, an error code indicating failure is returned.
- Both **SrcFrmRate** and **DstFrmRate** must be greater than 0 or equal to -1 when you set the frame rate control attribute of the channel.
- When **SrcFrmRate** and **DstFrmRate** are -1, the frame rate is not controlled.
- The frame adding mode is used if the input frame rate **SrcFrmRate** is less than the output frame rate **DstFrmRate**. In this mode, **DstFrmRate** is used as the absolute output frame rate. For example, when **SrcFrmRate** is less than **DstFrmRate** and **DstFrmRate** is 30, 30 frames are output finally.
- The frame discarding (frame reduction) mode is used if the input frame rate **SrcFrmRate** is greater than or equal to the output frame rate **DstFrmRate**. In this mode, some input frames are chosen for encoding. For example, when **SrcFrmRate** is 30 and **DstFrmRate** is 15, only 15 frames of the 30 input pictures are encoded.
- In frame discarding (frame reduction) mode, when the frequency of the frontend source (such as the VI) is fixed, the controlled channel frame rate is accurate. When the frequency of the frontend source (such as the VDEC) is not fixed, the controlled channel frame rate is inaccurate.
- The JPEG encoding mode does not support the frame adding mode, that is, the input frame rate (**SrcFrmRate**) must be greater than or equal to the output frame rate (**DstFrmRate**).

[Example]

None



[See Also]

None

HI_MPI_VENC_GetFrameRate

[Description]

Obtains the frame rate control attribute of a VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetFrameRate(VENC_CHN VeChn, VENC_FRAME_RATE_S  
*pstFrameRate);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstFrameRate	Pointer to frame rate control attribute of a VENC channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Request]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpapi.a

[Note]

- If you obtain the attribute of channel that does not exist, the error code [HI_ERR_VENC_UNEXIST](#) is returned.
- If **pstFrameRate** is null, an error code indicating failure is returned.

[Example]

None

[See Also]

None



HI_MPI_VENC_SetRcParam

[Description]

Sets the advanced RC parameters of a VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetRcParam(VENC_CHN VeChn, const VENC_RC_PARAM_S *pstRcParam);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstRcParam	Pointer to the advanced RC parameters of a VENC channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- The advanced parameters have default values. You do not need to call this MPI to enable a VENC channel.
- You are advised to call [HI_MPI_VENC_GetRcParam](#) to obtain the values of the advanced parameters for the RC, modify parameters values, and call [HI_MPI_VENC_SetRcPara](#) to set advanced parameters.
- The advanced parameters for the RC are available only for the H.264/H.265/MJPEG CBR mode, H.264/H.265/MJPEG VBR mode, and H264/H265 AVBR mode.
- The advanced parameters for the RC are as follows:
 - u32Thrd[RC_TEXTURE_THR_SIZE]** and **u32ThrdP[RC_TEXTURE_THR_SIZE]**Indicates two groups of thresholds for evaluating the macroblock complexity of the I frame and that of the P frame respectively. The thresholds in each group are sorted in ascending order. The value of each threshold ranges from 0 to 255. When the bit rate is controlled based on the macroblock, the thresholds can be used to adjust the QP values of macroblocks based on picture complexity.



For the H.264 encoding of the Hi3516A, the QP value of the current macroblock is the start QP value of the macroblock row plus a number (the maximum number is 12). That is, if the picture complexity value of the current macroblock is between two thresholds, the QP value of the current macroblock is the start QP value of the macroblock row plus x. The value of x is defined as follows: x is 0 if C (picture complexity value) \leq u32Thrd[0]; x is 1 if $u32Thrd[0] < C \leq u32Thr[1]$; x is 2 if $u32Thrd[1] < C \leq u32Thr[2]$; x is 3 if $u32Thrd[2] < C \leq u32Thr[3]$; x is 4 if $u32Thrd[3] < C \leq u32Thr[4]$; x is 5 if $u32Thrd[4] < C \leq u32Thr[5]$; x is 6 if $u32Thrd[5] < C \leq u32Thr[6]$; x is 7 if $u32Thrd[6] < C \leq u32Thr[7]$; x is 8 if $u32Thrd[7] < C \leq u32Thr[8]$; x is 9 if $u32Thrd[8] < C \leq u32Thr[9]$; x is 10 if $u32Thrd[9] < C \leq u32Thr[10]$; x is 11 if $u32Thrd[10] < C \leq u32Thr[11]$; x is 12 if $C > u32Thrd[11]$.

For the H.265 encoding of the Hi3516A and H.264 encoding of Hi3518E V200, the QP value of the current macroblock is the start QP value of the macroblock row plus a number (the maximum number is 8) or minus a number (the maximum number is 4). That is, if the picture complexity value of the current macroblock is less than or equal to **u32Thr[3]**, the QP value of the current macroblock is the start QP value of the macroblock row minus x. If the picture complexity value of the current macroblock is greater than **u32Thr[3]**, the QP value of the current macroblock is the start QP value of the macroblock row plus y. The values of x and y are defined as follows: x is 4 if $C < u32Thrd[0]$; x is 3 if $u32Thrd[0] \leq C < u32Thr[1]$; x is 2 if $u32Thrd[1] \leq C < u32Thr[2]$; x is 1 if $u32Thrd[2] \leq C < u32Thr[3]$; x and y are 0 if $u32Thrd[3] \leq C \leq u32Thr[4]$; y is 1 if $u32Thrd[4] < C \leq u32Thr[5]$; y is 2 if $u32Thrd[5] < C \leq u32Thr[6]$; y is 3 if $u32Thrd[6] < C \leq u32Thr[7]$; y is 4 if $u32Thrd[7] < C \leq u32Thr[8]$; y is 5 if $u32Thrd[8] < C \leq u32Thr[9]$; y is 6 if $u32Thrd[9] < C \leq u32Thr[10]$; y is 7 if $u32Thrd[10] < C \leq u32Thr[11]$; y is 8 if $C > u32Thrd[11]$.

- **u32RowQpDelta**

Indicates the fluctuation amplitude of the start QP value of each macroblock row relative to the start QP value of the frame when the bit rate is controlled based on the macroblock. If the system requires that the bit rate fluctuates slightly, you can increase the value of **u32QpDelta** to control the bit rate accurately. However, the quality of the macroblocks of some pictures may be different. In high bit rate mode, the recommended value of **u32QpDelta** is **0**; in medium bit rate mode, the recommended value is **0** or **1**; in low bit rate mode, the recommended value is **2**, **3**, **4**, or **5**.

- **s32FirstFrameStartQp**

Indicates the start QP value of the first frame, valid in CBR, VBR, or AVBR mode. If **s32FirstFrameStartQp** is **-1**, the start QP value of the first frame is internally calculated by the encoder. If **s32FirstFrameStartQp** is set to any other valid value, the user specifies this valid value as the start QP value of the first frame. The default value is **-1**. The first frame indicates the first IDR frame after the channel is created or the GOP mode, RC mode, or resolution is switched.

The CBR parameters are as follows:

- **u32MinIprop** and **u32MaxIprop**

Indicate the minimum IP proportion and maximum IP proportion respectively. The two parameters are advanced CBR parameters. The IP proportion is the ratio of the bits of I frames to the bits of P frames. The two parameters control the range of each IP proportion. When the value of **u32MinIprop** increases, the I frame becomes distinct and the P frame becomes blurred. When the value of **u32MaxIprop** decreases, the I frame becomes blurred and the P frame becomes distinct. You are advised not to limit the IP proportion. This avoids the respiratory effect and bit rate fluctuation. The default value of **u32MinIprop** is **1**, and the default value of



u32MaxIprop is **100**. If the size of the I frame is limited, you can set the values of **u32MinIprop** and **u32MaxIprop** based on requirement on the size change of the I frame.

- **u32MaxQp&u32MinQp**

Indicate the maximum QP value and minimum QP value of the current frame respectively. The two parameters are advanced CBR parameters. The two parameters control the QP values of all pictures effectively. For example, during the macroblock bit rate control, the QP value of each picture is restricted to fall within the range of **u32MinQp** to **u32MaxQp**. The default values of **u32MinQp** and **u32MaxQp** are **10** and **51** respectively. You are advised to retain the default values if there are no special quality requirements.

- **u32MaxPPDeltaQp** and **u32MaxIPDeltaQp**

Indicate the maximum difference between the QP values of two consecutive P frames and the maximum difference between the QP values of consecutive I frame and P frame respectively. The two parameters are advanced CBR parameters. When a large area moves fast or the application scenario changes, the difference between the QP values of P frames may be too large if you want to retain stable bit rate. As a result, the mosaic effect occurs. To avoid the change picture quality, you can change the values of the two parameters to control QP values. The default value of **u32MaxPPDelta** is **3**, and the default value of **u32MaxIPdeltaQP** is **5**. You can change the default values based on the required picture quality and bit rate.

- **s32IPQDelta**

Indicates the difference between the average QP value and the QP value of the current I frame. This parameter is an advanced CBR parameter and is used to adjust oversized I frame and avoids the respiratory effect. The parameter value can be a negative value and the default value is **2**. When the value increases, the I frame becomes distinct.

- **u32RQRatio[8]**

Indicates the weight of stable bit rate and stable quality. This parameter is an advanced CBR parameter. **u32RQRatio[i]/100** indicates the weight of stable quality, and **(1 - u32RQRatio[i])/100** indicates the weight of stable bit rate. For example, **u32RQRatio[i] = 75**. This indicates that the weight of stable quality is **75%**, and the weight of stable bit rate is **25%**. In CBR mode, eight scenarios are supported: Normal, Move (motion scenario), Still, StillToMove (still-to-motion scenario), MoveToStill (motion-to-still scenario), SceneSwitch (scenario switching), SharpMove (strenuous motion scenario), and Init (program initialization scenario). The eight scenarios correspond to the index IDs 0–7 of **u32RQRatio**. The default value of **u32RQRatio []** is **{75, 75, 75, 50, 50, 20, 30, 0}**. You can set the weights during bit rate control based on the application scenario. Assume that a large area moves fast, the weight of stable quality is **30%**, and the weight of stable bit rate is **70%**. You can decrease the weight of stable quality to ensure more stable bit rate. This parameter is reserved and not used currently.

The VBR parameters are as follows:

- **s32IPQDelta**

Indicates the difference between the average QP value and the QP value of the current I frame. This parameter is an advanced VBR parameter and is used to adjust oversized I frames and prevent the respiratory effect. The default value is **2**.

- **s32ChangePos**

Indicates the ratio of the current bit rate to the maximum bit rate when the QP value starts to be adjusted. This parameter is an advanced VBR parameter, and its default value is **90**. If the macroblocks of consecutive frames on the similar positions have



significant differences but the maximum bit rate cannot be exceeded, you are advised to decrease the value of **s32ChangePos** and increase the value of **s32DeltaQP**. However, when the bit rate becomes stable, the bit rate is low and the picture quality is poor.

The AVBR parameters are as follows:

- **s32ChangePos**

Indicates the ratio of the bit rate when the QP value starts to be adjusted in AVBR mode to the maximum bit rate. This parameter is an advanced AVBR parameter, and its default value is 90. If the macroblocks of consecutive frames in the similar positions have significant differences but the maximum bit rate cannot be exceeded, you are advised to decrease the value of **s32ChangePos** and increase the value of **s32DeltaQP**. However, when the bit rate becomes stable, the bit rate is low and the picture quality is poor.

- **u32MinIprop** and **u32MaxIprop**

Indicate the minimum I/P ratio and maximum I/P ratio, respectively. The two parameters are advanced AVBR parameters. The I/P ratio is the ratio of the bits of the I frame to the bits of the P frame. The two parameters control the range of the I/P ratio. When the value of **u32MinIprop** is increased, the I frame becomes clear and the P frame becomes blurred. When the value of **u32MaxIprop** is decreased, the I frame becomes blurred and the P frame becomes clear. You are advised not to limit the I/P ratio in normal cases to avoid the respiratory effect and bit rate fluctuation. The default values of **u32MinIprop** and **u32MaxIprop** are **1** and **100**, respectively. In the scenario where the size of the I frame is limited, you can set **u32MinIprop** and **u32MaxIprop** based on the requirement on the size fluctuation of the I frame.

- **u32MaxQp** and **u32MinQp**

Indicate the maximum QP value and minimum QP value of the current frame, respectively. The two parameters are advanced AVBR parameters. The clamping effect is the strongest. All other adjustments on the picture QP value, such as the macroblock bit rate control, are restricted to fall within the range of **u32MaxIQp** and **u32MinIQp**. The default values of **u32MinQp** and **u32MaxQp** are **16** and **51**, respectively. You are advised to retain the default values if there are no special quality requirements.

- **u32MaxIQp** and **u32MinIQp**

Indicate the maximum QP value and minimum QP value of the current IDR frame, respectively. The two parameters are advanced AVBR parameters. The clamping effect is the strongest. All other adjustments on the picture QP value, such as the macroblock bit rate control, are restricted to fall within the range of **u32MaxIQp** and **u32MinIQp**. The default values of **u32MinIQp** and **u32MaxIQp** are **16** and **51**, respectively. You are advised to retain the default values if there are no special quality requirements.

- **PRcParam**

Indicates an advanced user-defined parameter of the RC. It is reserved.

- You are advised to call this MPI after creating a channel and before starting encoding. This reduces the number of times of calling this MPI during encoding.

[Example]

```
HI_S32_SetRcParam(HI_VOID)
{
    HI_S32 s32Ret = HI_FAILURE;
    VENC_RC_PARAM_S stVencRcPara;
```



```
VENC_CHN VeChnId = 0;

//...omit other thing

s32Ret = HI_MPI_VENC_GetRcParam(VeChnId, &stVencRcPara);
if (HI_SUCCESS != s32Ret)
{
    printf("HI_MPI_VENC_GetRcParam err 0x%x\n", s32Ret);
    return HI_FAILURE;
}
stVencRcPara.stParamH264Cbr.enSuperFrmMode = SUPERFRM_DISCARD;

s32Ret = HI_MPI_VENC_SetRcParam(VeChnId, &stVencRcPara);
if (HI_SUCCESS != s32Ret)
{
    printf("HI_MPI_VENC_SetRcParam err 0x%x\n", s32Ret);
    return HI_FAILURE;
}
//...omit other thing
return HI_SUCCESS;
}
```

[See Also]

None

HI_MPI_VENC_GetRcParam

[Description]

Obtains the advanced RC parameters of a VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetRcParam(VENC_CHN VeChn, VENC\_RC\_PARAM\_S
*pstRcParam);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstRcParam	Pointer to the advanced RC parameters of a VENC channel	Output

[Return Value]



Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to obtain the configurations of the advanced parameters for the RC of an H.264/H.265/MJPEG channel. For details about parameters, see the description of [VENC_RC_PARAM_S](#).
- If **pstRcParam** is null, an error code indicating failure is returned.

[Example]

None

[See Also]

None

HI_MPI_VENC_SetRefParam

[Description]

Sets the advanced frame skipping reference parameters for an H.264/H.265 VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetRefParam(VENC_CHN VeChn, const VENC_PARAM_REF_S *  
pstRefParam);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstRefParam	Advanced frame skipping reference parameters for an H.264/H.265 VENC channel	Input

[Return Value]

Return Value	Description
0	Success



Return Value	Description
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- If **pstRefParam** is null, an error code indicating failure is returned.
- When an H.264/H.265 VENC channel is created, the 1x frame skipping reference mode is used by default. If you want to change the reference mode, you are advised to call this MPI to set the reference mode after creating the VENC channel but before starting encoding. This reduces the times of calling the MPI during encoding.
- If the MPI is called during encoding, the operation takes effect only when the next I frame is encoded.
- If the profile of an H.264 VENC channel is svc-t, an error code is returned after this MPI is called, indicating that this operation is prohibited.
- If the 1x frame skipping reference mode is selected, set **bEnablePred** to **HI_TRUE**, **u32Enhance** to **0**, and **u32Base** to **1**. If the 2x frame skipping reference mode is selected, set **bEnablePred** to **HI_TRUE**, **u32Enhance** to **1**, and **u32Base** to **1**. If the 4x frame skipping reference mode is selected, set **bEnablePred** to **HI_TRUE**, **u32Enhance** to **1**, and **u32Base** to **2**.

[Example]

None

[See Also]

None

HI_MPI_VENC_GetRefParam

[Description]

Obtains the advanced frame skipping reference parameters for an H.264/H.265 VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetRefParam(VENC_CHN VeChn,  
VENC_PARAM_REF_S * pstRefParam);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input



Parameter	Description	Input/Output
pstRefParam	Advanced frame skipping reference parameters for an H.264/H.265 VENC channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

If **pstRefParam** is null, an error code indicating failure is returned.

[Example]

None

[See Also]

None

HI_MPI_VENC_SetColor2Grey

[Description]

Enables or disables the color-to-gray function of a VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetColor2Grey(VENC_CHN VeChn, const VENC_COLOR2GREY_S*  
pstChnColor2Grey)
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID	Input
pstChnColor2Grey	Parameter for enabling or disabling the color-to-gray function	Input

[Return Value]



Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to set the color-to-gray attribute of a VENC channel.
- This MPI can be called after a VENC channel is created and before the channel is destroyed. If the MPI is called during encoding, the operation takes effect only after the next I frame is encoded.

[Example]

None

[See Also]

None

HI_MPI_VENC_GetColor2Grey

[Description]

Obtains the enable status of the color-to-gray function of a VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetColor2Grey(VENC_CHN VeChn, VENC_COLOR2GREY_S*  
pstChnColor2Grey)
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID	Input
pstChnColor2Grey	Parameter for enabling or disabling the color-to-gray function	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."



[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

If **pstChnColor2Grey** is null, an error code indicating failure is returned.

[Example]

None

[See Also]

None

HI_MPI_VENC_SetCrop

[Description]

Sets the cropping attribute of a VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetCrop(VENC_CHNVeChn, const VENC_CROP_CFG_S  
*pstCropCfg);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID	Input
pstCropCfg	Channel cropping attributes	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- The VENC channel crops pictures, and then determines whether to zoom out on pictures after comparing the cropped picture size with the VENC channel size.



- This MPI must be called before a VENC channel is created and before the channel is destroyed.
- Cropping attribute parameters consist of:
 - **bEnable:** This parameter is used to enable or disable the crop function of the channel.
 - **stRect:** The parameter is used to set the crop region attributes including the start point coordinates of the crop region and the crop region size.

[Example]

None

[See Also]

None

HI_MPI_VENC_GetCrop

[Description]

Obtains the cropping attribute of a VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetCrop(VENC_CHN VeChn, VENC_CROP_CFG_S *pstCropCfg);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID	Input
pstCropCfg	Channel cropping attributes	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

This MPI must be called after a VENC channel is created and before the channel is destroyed.

[Example]

None

[See Also]



None

HI_MPI_VENC_SetJpegSnapMode

[Description]

Sets the snapshot mode of a JPEG snapshot channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetJpegSnapMode(VENC_CHN VeChn, VENC_JPEG_SNAP_MODE_E enJpegSnapMode);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID	Input
enJpegSnapMode	Channel snapshot mode	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI must be called after a JPEG VENC channel is created and before the channel is destroyed.
- This MPI applies only to the JPEG VENC channel.
- A JPEG channel can work in either of the following modes:
 - JPEG_SNAP_ALL mode. After the channel starts to receive pictures, all received pictures are encoded.
 - JPEG_SNAP_FLASH mode. After the channel starts to receive pictures, only the pictures that are captured when the camera flash is on are encoded.
- After a JPEG channel is created, it is in JPEG_SNAP_ALL mode by default. To capture pictures only when the camera flash is on, call this MPI to set the JPEG channel mode to JPEG_SNAP_FLASH.
- The JPEG_SNAP_FLASH mode is available only when the front-end camera flash works.

[Example]



None

[See Also]

None

HI_MPI_VENC_GetJpegSnapMode

[Description]

Obtains the snapshot mode of a JPEG snapshot channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetJpegSnapMode(VENC_CHN VeChn, VENC_JPEG_SNAP_MODE_E *penJpegSnapMode);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID	Input
penJpegSnapMode	Channel snapshot mode	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI applies only to the JPEG VENC channel.
- This MPI must be called after a JPEG VENC channel is created and before the channel is destroyed.

[Example]

None

[See Also]

None



HI_MPI_VENC_SetH265SliceSplit

[Description]

Sets the slice split attribute for H.265 encoding.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetH265SliceSplit(VENC_CHN VeChn, const  
VENC_PARAM_H265_SLICE_SPLIT_S * pstSliceSplit);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID	Input
pstSliceSplit	Slice split parameters for H.265 streams	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to configure the stream split mode of an H.265 VENC channel.
- The slice split attribute depends on the following four parameters:
 - bSplitEnable:** Indicates whether the slice of the current frame is split.
 - u32SplitMode:** Indicates the mode of splitting slices. If **u32SplitMode** is set to **0**, slices are split by byte; if **u32SplitMode** is set to **1**, slices are split by LCU line.**u32SplitMode** cannot be set to a value greater than or equal to 2. **u32SplitMode** is valid only when **bSplitEnable** is set to **HI_TRUE**.
 - u32SliceSize:** Indicates the slice size. The definition of **u32SliceSize** varies according to the value of **u32SplitMode**. When **u32SplitMode** is **0**, **u32SliceSize** indicates the average number of bytes in each slice. The encoder starts to encode a picture from its upper left corner in raster sequence (that is, from left to right and from top to bottom) by macroblocks. The size of each slice depends on **u32SliceSize**. The size of encoded slices may be different from the configured value of **u32SliceSize**, and the deviation is always a positive number. When the last slice is encoded and the number of bytes in the remaining LCUs is less than **u32SliceSize**, the LCUs are encoded as a slice. When **u32SplitMode** is **1**, **u32SliceSize** indicates the number of LCU lines occupied by each slice, and its unit is LCU line. When the last line of the picture is to be



encoded and the number of remaining LCU lines is less than **u32SliceSize**, the LCU lines are considered as a packet.

- **loop_filter_across_slices_enabled_flag:** Indicates whether filtering is performed on the left and top borders of slices.
- This MPI is an advanced interface. You can call it as required. However, you are advised not to call this MPI. **bSplitEnable** is **false** by default.
- This MPI can be called after a VENC channel is created and before the channel is destroyed. If the MPI is called during encoding, the operation takes effect when the next frame is encoded.
- You are advised to call this MPI after creating a channel but before starting encoding. This reduces the number of times of calling this MPI during encoding.
- Before calling this MPI, you are advised to call [HI_MPI_VENC_SetH264SliceSplit](#) to obtain the slice split configurations of the current VENC channel.

[Example]

See the example of [HI_MPI_VENC_SetH264SliceSplit](#).

[See Also]

None

HI_MPI_VENC_GetH265SliceSplit

[Description]

Obtains the slice split attribute for H.265 encoding.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetH265SliceSplit(VENC_CHN VeChn,  
VENC_PARAM_H265_SLICE_SPLIT_S * pstSliceSplit);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstSliceSplit	Slice split parameters for H.265 streams	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]



- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to obtain the slice split mode of an H.265 VENC channel.
- This MPI can be called after a VENC channel is created and before the channel is destroyed.
- You are advised to call this MPI after creating a channel but before starting encoding. This reduces the number of times of calling this MPI during encoding.

[Example]

See the example of [HI_MPI_VENC_SetH264SliceSplit](#).

[See Also]

None

HI_MPI_VENC_SetH265PredUnit

[Description]

Sets the PU attribute for H.265 encoding.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetH265PredUnit(VENC_CHN VeChn, const  
VENC_PARAM_H265_PU_S *pstPredUnit);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID	Input
pstPredUnit	PU configuration of an H.265 VENC channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to configure PU attribute of an H.265 VENC channel.



- The PU attribute depends on the following nine parameters:
 - **bPu32x32En**: Indicates the PU32x32 block enable flag.
 - **bPu16x16En**: Indicates the PU16x16 block enable flag.
 - **bPu8x8En**: Indicates the PU8x8 block enable flag.
 - **bPu4x4En**: Indicates the PU4x4 block enable flag.
 - **constrained_intra_pred_flag**: For details about the parameter definition, see the H.265 protocol.
 - **strong_intra_smoothing_enabled_flag**: For details about the parameter definition, see the H.265 protocol.
 - **pcm_enabled_flag**: For details about the parameter definition, see the H.265 protocol.
 - **pcm_loop_filter_disabled_flag**: For details about the parameter definition, see the H.265 protocol.
 - **u32MaxNumMergeCand**: For details about the parameter definition, see the H.265 protocol.
- This MPI is an advanced interface. You can call it as required. However, you are advised not to call this MPI. The MPI parameters have default values.
- Among **bPu32x32En**, **bPu16x16En**, **bPu8x8En**, and **bPu4x4En**, bPu16x16En and bPu8x8En must be set to HI_TURE.
- This MPI can be called after a VENC channel is created and before the channel is destroyed. If the MPI is called during encoding, the operation takes effect when the next I frame is encoded.
- You are advised to call this MPI after creating a channel but before starting encoding. This reduces the number of times of calling this MPI during encoding.
- Before calling this MPI, you are advised to call [HI_MPI_VENC_GetH265PredUnit](#) to obtain the PU configurations of the current VENC channel.

[Example]

None

[See Also]

None

HI_MPI_VENC_GetH265PredUnit

[Description]

Obtains the PU attribute for H.265 encoding.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetH265PredUnit(VENC_CHN VeChn, VENC_PARAM_H265_PU_S  
*pstPredUnit);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input



Parameter	Description	Input/Output
pstPredUnit	PU configuration of an H.265 VENC channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to obtain the PU attribute of an H.265 VENC channel.
- This MPI can be called after a VENC channel is created and before the channel is destroyed.
- You are advised to call this MPI after creating a channel but before starting encoding. This reduces the number of times of calling this MPI during encoding.

[Example]

None

[See Also]

None

HI_MPI_VENC_SetH265Trans

[Description]

Sets the transformation and quantization attribute for H.265 encoding.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetH265Trans(VENC_CHN VeChn, const  
VENC_PARAM_H265_TRANS_S *pstH265Trans);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID	Input
pstH265Trans	Transformation and quantization configurations of an H.265 VENC channel	Input



[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to configure transformation and quantization attribute of an H.265 VENC channel.
- The transformation and quantization attribute depends on the following four parameters:
 - **transquant_bypass_enabled_flag**: For details about the parameter definition, see the H.265 protocol.
 - **transform_skip_enabled_flag**: For details about the parameter definition, see the H.265 protocol.
 - **cb_qp_offset**: For details about the parameter definition, see **slice_cb_qp_offset** in the H.265 protocol.
 - **cr_qp_offset**: For details about the parameter definition, see **slice_cr_qp_offset** in the H.265 protocol.
- This MPI is an advanced interface. You can call it as required. However, you are advised not to call this MPI. The MPI parameters have default values.
- This MPI can be called after a VENC channel is created and before the channel is destroyed. If the MPI is called during encoding, the operation takes effect when the next I frame is encoded.
- You are advised to call this MPI after creating a channel but before starting encoding. This reduces the number of times of calling this MPI during encoding.
- Before calling this MPI, you are advised to call [HI_MPI_VENC_GetH265Trans](#) to obtain the transformation and quantization configurations of the current VENC channel.

[Example]

None

[See Also]

None

HI_MPI_VENC_GetH265Trans

[Description]

Obtains the transformation and quantization attribute for H.265 encoding.

[Syntax]



```
HI_S32 HI_MPI_VENC_GetH265Trans(VENC_CHN VeChn, VENC_PARAM_H265_TRANS_S  
*pstH265Trans);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstH265Trans	Transformation and quantization configurations of an H.265 VENC channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to obtain the transformation and quantization attribute of an H.265 VENC channel.
- This MPI can be called after a VENC channel is created and before the channel is destroyed.
- You are advised to call this MPI after creating a channel but before starting encoding. This reduces the number of times of calling this MPI during encoding.

[Example]

None

[See Also]

None

HI_MPI_VENC_SetH265Entropy

[Description]

Sets the entropy encoding attribute of an H.265 channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetH265Entropy (VENC_CHN VeChn, const  
VENC_PARAM_H265_ENTROPY_S *pstH265Entropy);
```



[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID	Input
pstH265Entropy	Entropy encoding configuration of an H.265 VENC channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to configure the entropy encoding attribute of an H.265 VENC channel.
- The entropy encoding attribute depends on the **cabac_init_flag** parameter. For details about the parameter definition, see the H.265 protocol.
- This MPI is an advanced interface. You can call it as required. However, you are advised not to call this MPI. The MPI parameters have default values.
- This MPI can be called after a VENC channel is created and before the channel is destroyed. If the MPI is called during encoding, the operation takes effect when the next frame is encoded.
- You are advised to call this MPI after creating a channel but before starting encoding. This reduces the number of times of calling this MPI during encoding.
- Before calling this MPI, you are advised to call [HI_MPI_VENC_GetH265Entropy](#) to obtain the entropy encoding configurations of the current VENC channel.

[Example]

None

[See Also]

None

HI_MPI_VENC_GetH265Entropy

[Description]

Obtains the entropy encoding attribute of an H.265 channel.

[Syntax]



```
HI_S32 HI_MPI_VENC_GetH265Entropy(VENC_CHN VeChn,  
VENC_PARAM_H265_ENTROPY_S *pstH265Entropy);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstH265Entropy	Entropy encoding configuration of an H.265 VENC channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to obtain the entropy encoding attribute of an H.265 VENC channel.
- This MPI can be called after a VENC channel is created and before the channel is destroyed.
- You are advised to call this MPI after creating a channel but before starting encoding.
This reduces the number of times of calling this MPI during encoding.

[Example]

None

[See Also]

None

HI_MPI_VENC_SetH265Dbblk

[Description]

Sets the deblocking attribute for H.265 encoding.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetH265Dbblk (VENC_CHN VeChn, const  
VENC_PARAM_H265_DBLOCK_S *pstH265Dbblk);
```



[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID	Input
pstH265Dbblk	Deblocking configuration of an H.265 VENC channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to configure the deblocking attribute of an H.265 VENC channel.
- The deblocking attribute depends on the following three parameters:
 - **slice_deblocking_filter_disabled_flag**: For details, see the H.265 protocol.
 - **slice_beta_offset_div2**: For details, see the H.265 protocol.
 - **slice_tc_offset_div2**: For details, see the H.265 protocol.
- This MPI is an advanced interface. You can call it as required. However, you are advised not to call this MPI. Deblocking is enabled by default. The default values of **slice_deblocking_filter_disabled_flag**, **slice_tc_offset_div2**, and **slice_beta_offset_div2** are **0**.
- If you want to disable deblocking, set **slice_deblocking_filter_disabled_flag** to **1**.
- This MPI can be called after a VENC channel is created and before the channel is destroyed. If the MPI is called during encoding, the operation takes effect when the next frame is encoded.
- You are advised to call this MPI after creating a channel but before starting encoding. This reduces the number of times of calling this MPI during encoding.
- Before calling this MPI, you are advised to call [HI_MPI_VENC_GetH265Dbblk](#) to obtain the deblocking configurations of the current VENC channel.

[Example]

None

[See Also]

None



HI_MPI_VENC_GetH265Dbblk

[Description]

Obtains the deblocking attribute for H.265 encoding.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetH265Dbblk(VENC_CHN VeChn, VENC_PARAM_H265_DBLK_S  
*pstH265Dbblk);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstH265Dbblk	Deblocking configuration of an H.265 VENC channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to obtain the deblocking attribute of an H.265 VENC channel.
- This MPI can be called after a VENC channel is created and before the channel is destroyed.
- You are advised to call this MPI after creating a channel but before starting encoding. This reduces the number of times of calling this MPI during encoding.

[Example]

None

[See Also]

None

HI_MPI_VENC_SetH265Sao

[Description]

Sets the SAO attribute for H.265 encoding.



[Syntax]

```
HI_S32 HI_MPI_VENC_SetH265Sao(VENC_CHN VeChn, const VENC_PARAM_H265_SAO_S
*pstH265Sao);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID	Input
pstH265Sao	SAO configuration of an H.265 VENC channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to configure the SAO attribute of an H.265 VENC channel.
- The SAO attribute depends on the following two parameters:
 - **slice_sao_luma_flag**: Indicates whether SAO filtering is performed on the luminance component of the current slice.
 - **slice_sao_chroma_flag**: Indicates whether SAO filtering is performed on the chrominance component of the current slice.
- This MPI is an advanced interface. You can call it as required. However, you are advised not to call this MPI. The SAO function is enabled by default. The default values of **slice_sao_luma_flag** and **slice_sao_chroma_flag** are 1.
- This MPI can be called after a VENC channel is created and before the channel is destroyed. If the MPI is called during encoding, the operation takes effect when the next frame is encoded.
- You are advised to call this MPI after creating a channel but before starting encoding. This reduces the number of times of calling this MPI during encoding.
- Before calling this MPI, you are advised to call [HI_MPI_VENC_GetH265Sao](#) to obtain the SAO configurations of the current VENC channel.

[Example]

None

[See Also]



None

HI_MPI_VENC_GetH265Sao

[Description]

Obtains the SAO attribute for H.265 encoding.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetH265Sao(VENC_CHN VeChn, VENC_PARAM_H265_SAO_S
*pstH265Sao);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstH265Sao	SAO configuration of an H.265 VENC channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpci.a

[Note]

- This MPI is used to obtain the SAO attribute of an H.265 VENC channel.
- This MPI can be called after a VENC channel is created and before the channel is destroyed.
- You are advised to call this MPI after creating a channel but before starting encoding. This reduces the number of times of calling this MPI during encoding.

[Example]

None

[See Also]

None



HI_MPI_VENC_SetH265Timing

[Description]

Sets the timing attribute of VPS packets for H.265 encoding.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetH265Timing(VENC_CHN VeChn, const  
VENC_PARAM_H265_TIMING_S *pstH265Timing);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID	Input
pstH265Timing	Timing configuration of an H.265 VENC channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to obtain the timing attribute of an H.265 VENC channel.
- The timing attribute depends on the following four parameters:
 - **timing_info_present_flag**: For details, see the H.265 protocol.
 - **num_units_in_tick**: For details, see the H.265 protocol.
 - **time_scale**: For details, see the H.265 protocol.
 - **num_ticks_poc_diff_one**: For details, see the H.265 protocol.
- This MPI is an advanced interface. You can call it as required. However, you are advised not to call this MPI. The timing function is disabled by default. The default value of **timing_info_present_flag** is **0**, the default values of **num_units_in_tick** and **num_ticks_poc_diff_one** are **1**, and the default value of **time_scale** is **60**.
- This MPI can be called after a VENC channel is created and before the channel is destroyed. If the MPI is called during encoding, the operation takes effect when the next I frame is encoded.
- You are advised to call this MPI after creating a channel but before starting encoding. This reduces the number of times of calling this MPI during encoding.



- Before calling this MPI, you are advised to call [HI_MPI_VENC_GetH265Timing](#) to obtain the timing configurations of the current VENC channel.

[Example]

None

[See Also]

None

HI_MPI_VENC_GetH265Timing

[Description]

Obtains the timing attribute of VPS packets for H.265 encoding.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetH265Timing(VENC_CHN VeChn, VENC\_PARAM\_H265\_TIMING\_S*pstH265Timing);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstH265Timing	Timing configuration of an H.265 VENC channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to obtain the timing attribute of an H.265 VENC channel.
- This MPI can be called after a VENC channel is created and before the channel is destroyed.
- You are advised to call this MPI after creating a channel but before starting encoding. This reduces the number of times of calling this MPI during encoding.

[Example]



None

[See Also]

None

HI_MPI_VENC_SetFrameLostStrategy

[Description]

Sets the frame discarding policies when the instantaneous bit rate is above the threshold.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetFrameLostStrategy(VENC_CHN VeChn, const  
VENC_PARAM_FRAMELOST_S *pstFrmLostParam);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstFrmLostParam	Frame discarding parameters for a VENC channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

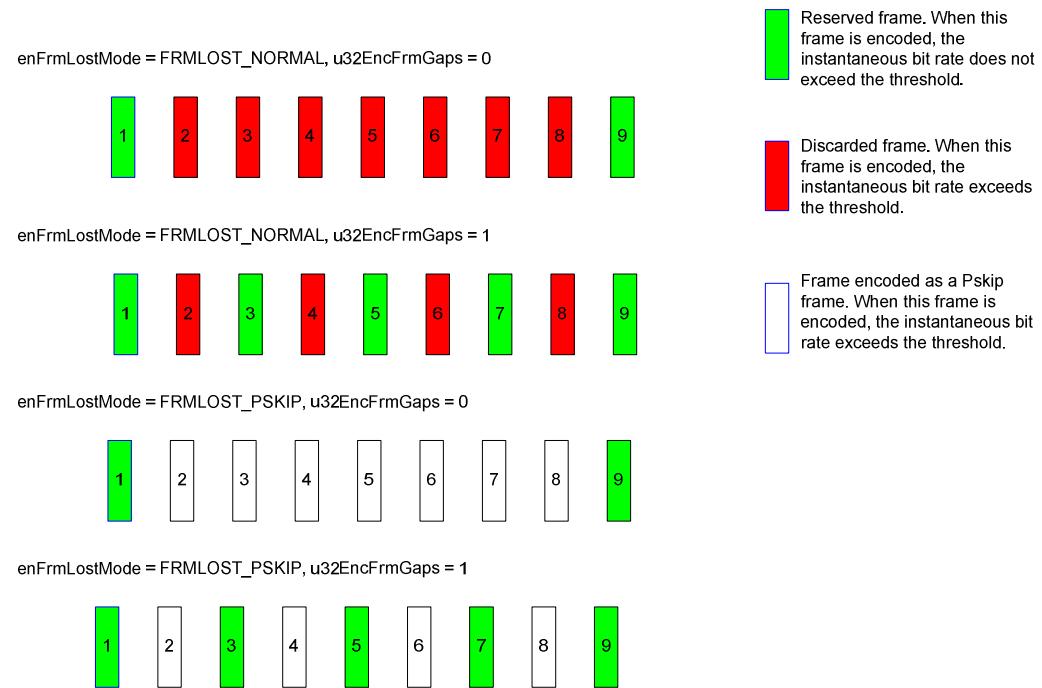
[Note]

- If **pstFrmLostParam** is null, an error code indicating failure is returned.
- **pstFrmLostParam** depends on the following four parameters:
 - **enFrmLostMode**: Indicates the frame discarding mode.
 - **u32EncFrmGaps**: Indicates the frame discarding interval.
 - **bFrmLostOpen**: Indicates the frame discarding enable.
 - **u32FrmLostBpsThr**: Indicates the frame discarding threshold.
- This MPI is an advanced interface. You can call it as required. However, the MPI parameters have default values. Frames are discarded when the instantaneous bit rate is above the threshold by default.



- When the instantaneous bit rate is above the threshold, two processing modes are supported: discarding frames and encoding Pskip frames. Only the processing mode of discarding frames is supported during MJPEG encoding.
- u32EncFrmGaps** determines whether to discard frames or encode Pskip frames evenly. See [Figure 6-12](#).

Figure 6-12 Frame processing modes



- This MPI can be called after a VENC channel is created and before the channel is destroyed. If the MPI is called during encoding, the operation takes effect when the next frame is encoded.

[Example]

None

[See Also]

None

HI_MPI_VENC_GetFrameLostStrategy

[Description]

Obtains the frame discarding polices when the instantaneous bit rate is above the threshold.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetFrameLostStrategy(VENC_CHN VeChn,  
VENC_PARAM_FRAMELOST_S *pstFrmLostParam);
```

[Parameter]



Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstFrmLostParam	Frame discarding parameters for a VENC channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

If **pstFrmLostParam** is null, an error code indicating failure is returned.

[Example]

None

[See Also]

None

HI_MPI_VENC_SetSuperFrameCfg

[Description]

Sets the processing mode of jumbo frames for the encoding channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetSuperFrameCfg(VENC_CHN VeChn, const  
VENC_SUPERFRAME_CFG_S *pstSuperFrmParam);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstSuperFrmParam	Configuration parameters of jumbo frames for the encoding channel	Input



[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_rc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- If no VENC channel is created, an error code indicating failure is returned.
- This MPI is an advanced interface. You can call it as required. The MPI parameters have the following default values: **enSuperFrmMode = SUPERFRM_NONE**, **u32SuperIFrmBitsThr = 500000**, **u32SuperPfrmBitsThr = 500000**, and **u32SuperBfrmBitsThr = 500000**.
- This MPI can be called after an encoding channel is created and before the channel is destroyed.

[Example]

None

[See Also]

None

HI_MPI_VENC_GetSuperFrameCfg

[Description]

Obtains the processing mode of jumbo frames for the encoding channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetSuperFrameCfg(VENC_CHN VeChn, VENC_SUPERFRAME_CFG_S  
*pstSuperFrmParam);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstSuperFrmParam	Configuration parameters of jumbo frames for the encoding channel	Output

[Return Value]



Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

If **pstSuperFrmParam** is null, an error code indicating failure is returned.

[Example]

None

[See Also]

None

HI_MPI_VENC_SetChnlPriority

[Description]

Sets the channel priority.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetChnlPriority(VENC_CHN VeChn, HI_U32 u32Priority);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
u32Priority	VENC channel priority Value range: [0, 2)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."



[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- If **u32Priority** is greater than 1, an error code indicating failure is returned.
- This MPI is an advanced interface. You can call it as required. The default channel priority is 0.
- This MPI can be called after a VENC channel is created and before the channel is destroyed.

[Example]

None

[See Also]

None

HI_MPI_VENC_GetChnlPriority

[Description]

Obtains the channel priority.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetChnlPriority(VENC_CHN VeChn, HI_U32 *pu32Priority);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pu32Priority	VENC channel priority	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]



If **pu32Priority** is null, an error code indicating failure is returned.

[Example]

None

[See Also]

None

HI_MPI_VENC_SetRcPriority

[Description]

Sets the RC priority.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetRcPriority(VENC_CHN VeChn, VENC_RC_PRIORITY_E enRcPriority);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID	Input
enRcPriority	Priority enumeration	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to set priority to the target bit rate or the jumbo frame threshold in bit rate control. If the target bit rate takes priority over the jumbo frame threshold, when the bit rate is too low, jumbo frames may be encoded to compensate the rate and they are not re-encoded. If the jumbo frame threshold takes priority over the target bit rate, the jumbo frames are re-encoded to decrease the number of bits. As a result, the bit rate may be too low.
- This MPI must be called after a VENC channel is created and before the channel is destroyed.
- This MPI supports only the bit rate control modes of the H.264 and H.265 protocols.



[Example]

None

[See Also]

None

HI_MPI_VENC_GetRcPriority

[Description]

Obtains the RC priority.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetRcPriority(VENC_CHN VeChn, VENC_RC_PRIORITY_E *penRcPriority);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID	Input
penRcPriority	Pointer to the RC priority	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

This MPI must be called after a VENC channel is created and before the channel is destroyed.

[Example]

None

[See Also]

None

HI_MPI_VENC_GetIntraRefresh

[Description]



Obtains the parameter for refreshing I macroblocks.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetIntraRefresh(VENC_CHN VeChn,  
VENC_PARAM_INTRA_REFRESH_S *pstIntraRefresh)
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID	Input
pstIntraRefresh	Pointer to the parameter for refreshing I macroblocks	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

This MPI must be called after a VENC channel is created and before the channel is destroyed.

[Example]

None

[See Also]

None

HI_MPI_VENC_SetIntraRefresh

[Description]

Sets the parameter for refreshing I macroblocks.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetIntraRefresh(VENC_CHN VeChn,  
VENC_PARAM_INTRA_REFRESH_S *pstIntraRefresh)
```

[Parameter]



Parameter	Description	Input/Output
VeChn	VENC channel ID	Input
pstIntraRefresh	Pointer to the parameter for refreshing I macroblocks	Input

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI needs to be called to reconfigure the parameter after GOP is changed.
- This MPI needs to be called to reconfigure the parameter after the advanced frame skipping reference parameter is configured. If **bEnablePred** is set to **0**, the advanced frame skipping reference mode is not supported.
- The number of rows to be refreshed must be set to an appropriate value to ensure that I macroblock refresh can be finished within one GOP cycle. Note that the refresh is implemented only in the BASE_PSLICE_REFBYBASE frame at the base layer in advanced frame skipping reference mode.
- **u32RefreshLineNum** needs to meet the formulas in [Table 6-12](#).

Table 6-12 Calculation of u32RefreshLineNum

-	Calculation Formula	Remarks
H.264	$u32RefreshLineNum * Gop \geq (u32PicHeight + 15)/16$	If advanced frame skipping reference is not used: $Gop = Gop$;
H.265	$u32RefreshLineNum * Gop \geq (u32PicHeight + 63)/64$	If advanced frame skipping reference is used: $Gop = (Gop + (u32Base * (u32Enhance + 1) - 1))/(u32Base * (u32Enhance+1))$

- For the MPI that takes effect when the next I frame is encoded, an I frame is generated after parameters are configured by calling this MPI to ensure that parameter settings take effect.
- This MPI supports only the H.264 protocol.

[Example]

None

HI_MPI_VENC_SetRefParamEx

[Description]

Sets the advanced frame skipping reference parameters for an H.264/H.265 VENC channel with virtual I frames.



[Syntax]

```
HI_S32 HI_MPI_VENC_SetRefParamEx(VENC_CHN VeChn, const  
VENC_PARAM_REF_EX_S * pstRefParam);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstRefParam	Advanced frame skipping reference parameters for an H.264/H.265 VENC channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- If **pstRefParam** is null, an error code indicating failure is returned.
- Compared with the advanced frame skipping reference MPI, this MPI contains one new feature that supports the insertion of the virtual I frame into the bottom layer. When the virtual I frame is not inserted, this MPI has the same effect as the advanced frame skipping reference MPI. When the virtual I frame is inserted, the fast forward mode at multiple rates is supported.
- If this MPI is called during encoding, the operation takes effect only when the next I frame is encoded.
- This MPI cannot be called when the profile of an H.264 VENC channel is svc-t.
- If the virtual I frame is enabled, this MPI does not support IntraRefresh.
- This MPI supports the configuration of the interval for virtual I frames.
- This MPI supports the adjustment of **QpDelta** for the virtual I frames relative to P frames. The quality of the virtual I frames is adjusted to optimize the encoding quality.

[Example]

None

[See Also]

None



HI_MPI_VENC_GetRefParamEx

[Description]

Obtains the advanced frame skipping reference parameters for an H.264/H.265 VENC channel with virtual I frames.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetRefParamEx(VENC_CHN VeChn, VENC_PARAM_REF_EX_S *  
pstRefParam);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstRefParam	Advanced frame skipping reference parameters for an H.264/H.265 VENC channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

If **pstRefParam** is null, an error code indicating failure is returned.

[Example]

None

[See Also]

None

HI_MPI_VENC_SetModParam

[Description]

Sets the module parameters related to encoding.

[Syntax]



```
HI_S32 HI_MPI_VENC_SetModParam(VENC\_PARAM\_MOD\_S *pstModParam)
```

[Parameter]

Parameter	Description	Input/Output
pstModParam	Pointer to the parameters of the encoding module	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI can be called only before a channel is created. If this MPI is called after a channel is created, the error code HI_ERR_VENC_NOT_PERM is returned.
- This MPI can be called to set the module parameters **hi35xx_venc.ko**, **hi35xx_h264e.ko**, **hi35xx_h265e.ko**, and **hi35xx_jpge.ko**.

[Example]

None

[See Also]

None

HI_MPI_VENC_GetModParam

[Description]

Obtains the module parameters related to encoding.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetModParam(VENC\_PARAM\_MOD\_S *pstModParam)
```

[Parameter]

Parameter	Description	Input/Output
pstModParam	Pointer to the parameters of the encoding module	Output



[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

This MPI can be called to obtain the module parameters **hi35xx_venc.ko**, **hi35xx_h264e.ko**, **hi35xx_h265e.ko**, and **hi35xx_jpge.ko**.

[Example]

None

[See Also]

None

6.4 Data Structures

VENC data structures are defined as follows:

- [VENC_MAX_CHN_NUM](#): Defines the maximum number of channels.
- [RC_TEXTURE_THR_SIZE](#): Defines the number of thresholds for bit rate control in texture.
- [H264E_NALU_TYPE_E](#): Defines the NALU type of an H.264 stream.
- [H264E_REFSLICE_TYPE_E](#): Defines the reference frame type in a specific frame skipping reference mode based on the obtained H.264 streams.
- [H264E_REF_TYPE_E](#): Defines the frame type and reference attributes of the H.264 frame skipping reference streams.
- [H265E_REF_TYPE_E](#): Defines the frame type and reference attributes of the H.265 frame skipping reference streams.
- [JPEGE_PACK_TYPE_E](#): Defines the Pack type of a JPEG stream.
- [MPEG4E_PACK_TYPE_E](#): Defines the Pack type of an MPEG-4 stream.
- [H265E_NALU_TYPE_E](#): Defines the NALU type of an H.265 stream.
- [VENC_DATA_TYPE_U](#): Defines the stream result type.
- [VENC_PACK_S](#): Defines a stream packet.
- [VENC_STREAM_INFO_H264_S](#): Defines the features of an H.264 stream.
- [VENC_STREAM_INFO_JPEG_S](#): Defines the features of a JPEG/MJPEG stream.
- [VENC_STREAM_INFO_MPEG4_S](#): Defines the features of an MPEG-4 stream
- [VENC_STREAM_INFO_H265_S](#): Defines the features of an H.265 stream.



- [VENC_STREAM_S](#): Defines the stream frame type.
- [VENC_STREAM_BUF_INFO_S](#): Defines the stream buffer information.
- [VENC_ATTR_H264_S](#): Defines the attribute of the H.264 encoder.
- [VENC_ATTR_MJPEG_S](#): Defines the attribute of the MJPEG encoder.
- [VENC_ATTR_JPEG_S](#): Defines the attribute of the JPEG snapshot encoder.
- [VENC_ATTR_MPEG4_S](#): Defines the attribute of the MPEG-4 encoder.
- [VENC_ATTR_H265_S](#): Defines the attribute of the H.265 encoder.
- [VENC_ATTR_S](#): Defines the encoder attributes.
- [VENC_CHN_ATTR_S](#): Defines the attribute of a VENC channel.
- [VENC_CHN_STAT_S](#): Defines the status of a VENC channel.
- [VENC_PARAM_H264_SLICE_SPLIT_S](#): Defines the slice attribute of an H.264 channel.
- [VENC_PARAM_H264_INTER_PRED_S](#): Defines the inter-prediction attribute of an H.264 channel.
- [VENC_PARAM_H264_INTRA_PRED_S](#): Defines the intra-prediction attribute of an H.264 channel.
- [VENC_PARAM_H264_TRANS_S](#): Defines the transformation and quantization attribute of an H.264 channel.
- [VENC_PARAM_H264_ENTROPY_S](#): Defines the entropy attribute of an H.264 channel.
- [VENC_PARAM_H264_POC_S](#): Defines the POC attribute of an H.264 channel.
- [VENC_PARAM_H264_DBLOCK_S](#): Defines the de-blocking attribute of an H.264 channel.
- [VENC_PARAM_H264_VUI_S](#): Defines the VUI attribute of an H.264 channel.
- [VENC_PARAM_H265_VUI_S](#): Defines the VUI structure of an H.265 channel.
- [VENC_PARAM_VUI_ASPECT_RATIO_S](#): Defines the AspectRatio information in the VUI of an H.264/H.265 encoding channel.
- [VENC_PARAM_VUI_H264_TIME_INFO_S](#): Defines the Time_Info information in the VUI of an H.264 encoding channel.
- [VENC_PARAM_VUI_H265_TIME_INFO_S](#): Defines the Time_Info information in the VUI of an H.265 encoding channel.
- [VENC_PARAM_VUI_VIDEO_SIGNAL_S](#): Defines the Video_Signal information in the VUI of an H.264/H.265 encoding channel.
- [VENC_PARAM_VUI_BITSTREAM_RESTRICT_S](#): Defines the Bitstream_Restriction structure in the VUI of an H.264/H.265 VENC channel.
- [VENC_PARAM_JPEG_S](#): Defines the parameters of the JPEG encoder.
- [VENC_PARAM_MJPEG_S](#): Defines the advanced parameters of the MJPEG encoder.
- [VENC_ROI_CFG_S](#): Defines an ROI for a VENC channel.
- [VENC_ROIBG_FRAME_RATE_S](#): Defines the frame rate of a non-ROI.
- [VENC_PARAM_REF_S](#): Defines the advanced frame skipping reference parameters for H.264/H.265 encoding.
- [VENC_PARAM_REF_EX_S](#): Defines the advanced frame skipping reference parameters for an H.264/H.265 VENC channel with virtual I frames.
- [VENC_RC_ATTR_S](#): Defines RC attributes for controlling the bit rate of a VENC channel.



- **VENC_RC_MODE_E:** Defines the mode of the RC for controlling the bit rate of a VENC channel.
- **VENC_ATTR_H264_CBR_S:** Defines the CBR attribute of an H.264 channel.
- **VENC_ATTR_H264_VBR_S:** Defines the VBR attribute of an H.264 channel.
- **VENC_ATTR_H264_FIXQP_S:** Defines the FixQP attribute of an H.264 channel.
- **VENC_ATTR_MPEG4_CBR_S:** Defines the CBR attribute of an MPEG-4 channel.
- **VENC_ATTR_MPEG4_VBR_S:** Defines the VBR attribute of an MPEG-4 channel.
- **VENC_ATTR_H264_AVBR_S:** Defines the AVBR attribute structure of an H.264 VENC channel.
- **VENC_ATTR_MPEG4_FIXQP_S:** Defines the FixQP attribute of an MPEG-4 channel.
- **VENC_ATTR_MJPEG_FIXQP_S:** Defines the FixQP attribute of an MJPEG channel.
- **VENC_ATTR_MJPEG_CBR_S:** Defines the CBR attribute of an MJPEG channel.
- **VENC_ATTR_MJPEG_VBR_S:** Defines the VBR attribute of an MJPEG channel.
- **VENC_SUPERFRM_MODE_E:** Defines the mode of processing jumbo frames when the bit rate is being controlled.
- **VENC_PARAM_H264_VBR_S:** Defines the configurations of the advanced parameters related to the VBR control mode of an H.264 channel.
- **VENC_ATTR_H265_CBR_S:** Defines the CBR attribute of an H.265 VENC channel.
- **VENC_ATTR_H265_VBR_S:** Defines the VBR attribute of an H.265 VENC channel.
- **VENC_ATTR_H265_AVBR_S:** Defines the AVBR attribute structure of an H.265 VENC channel.
- **VENC_ATTR_H265_FIXQP_S:** Defines the FixQP attribute of an H.265 VENC channel.
- **VENC_PARAM_H264_CBR_S:** Defines the configurations of the advanced parameters related to the new CBR control mode of an H.264 channel.
- **VENC_PARAM_MJPEG_CBR_S:** Defines the configurations of the advanced parameters related to the CBR control mode of an MJPEG channel.
- **VENC_PARAM_MJPEG_VBR_S:** Defines the configurations of the advanced parameters related to the VBR control mode of an MJPEG channel.
- **VENC_PARAM_MPEG4_CBR_S:** Defines the configurations of the advanced parameters related to the CBR control mode of an MPEG-4 channel.
- **VENC_PARAM_H264_AVBR_S:** Defines the configuration of the advanced parameters related to the AVBR control mode of an H.264 VENC channel.
- **VENC_PARAM_MPEG4_VBR_S:** Defines the configurations of the advanced parameters related to the VBR control mode of an MPEG-4 channel.
- **VENC_PARAM_H265_VBR_S:** Defines the configurations of the advanced parameters related to the VBR control mode of an H.265 encoding channel.
- **VENC_PARAM_H265_AVBR_S:** Defines the configurations of the advanced parameters related to the AVBR control mode of an H.265 VENC channel.
- **VENC_PARAM_H265_CBR_S:** Defines the configurations of the advanced parameters related to the CBR control mode of an H.265 encoding channel.
- **VENC_RC_PARAM_S:** Defines the advanced parameters for controlling the bit rate of a VENC channel.
- **VENC_CROP_CFG_S:** Defines the crop parameters of a VENC channel.
- **VENC_FRAME_RATE_S:** Defines the parameters for controlling the frame rate of a VENC channel.
- **VENC_COLOR2GREY_S:** Defines color-to-gray parameters.



- [VENC_JPEG_SNAP_MODE_E](#): Defines the snapshot mode of a JPEG VENC channel.
- [VENC_RECV_PIC_PARAM_S](#): Defines the number of received frames to be encoded.
- [H264E_IDR_PIC_ID_MODE_E](#): Defines the mode of setting the idr_pic_id of an IDR frame or I frame.
- [VENC_H264_IDRPICID_CFG_S](#): Defines the parameters for setting the idr_pic_id of an IDR frame or I frame.
- [VENC_PARAM_H265_SLICE_SPLIT_S](#): Defines the slice split attribute of an H.265 VENC channel.
- [VENC_PARAM_H265_PU_S](#): Defines the PU attribute of an H.265 VENC channel.
- [VENC_PARAM_H265_TRANS_S](#): Defines the transformation and quantization attribute of an H.265 VENC channel.
- [VENC_PARAM_H265_ENTROPY_S](#): Defines the entropy encoding attribute of an H.265 VENC channel.
- [VENC_PARAM_H265_DBLK_S](#): Defines the deblocking attribute of an H.265 VENC channel.
- [VENC_PARAM_H265_SAO_S](#): Defines the SAO attribute of an H.265 VENC channel.
- [VENC_PARAM_H265_TIMING_S](#): Defines the timing attribute of an H.265 VENC channel.
- [VENC_FRAMELOST_MODE_E](#): Defines the frame discarding mode when the instantaneous bit rate of a VENC channel is above the threshold.
- [VENC_PARAM_FRAMELOST_S](#): Defines the frame discarding policy when the instantaneous bit rate of a VENC channel is above the threshold.
- [VENC_SUPERFRAME_CFG_S](#): Defines the parameters for jumbo frame processing modes.
- [VENC_RC_PRIORITY_E](#): Defines the RC priority.
- [VENC_PARAM_INTRA_REFRESH_S](#): Defines the parameter for controlling I macroblock refresh in the P frame.
- [VENC_PARAM_MOD_S](#): Defines the module parameters related to encoding.
- [VENC_MODTYPE_E](#): Defines the type of the module parameters related to encoding.
- [VENC_PARAM_MOD_VENC_S](#): Defines the module parameter for **hi35xx_venc.ko**.
- [VENC_PARAM_MOD_H264E_S](#): Defines the module parameter for **hi35xx_h264e.ko**.
- [VENC_PARAM_MOD_H265E_S](#): Defines the module parameter for **hi35xx_h265.ko**.
- [VENC_PARAM_MOD_JPEGE_S](#): Defines the module parameter for **hi35xx_jpeg.ko**.

VENC_MAX_CHN_NUM

[Description]

Defines the maximum number of channels.

[Syntax]

```
#define VENC_MAX_CHN_NUM 16
```

[Note]

None

[See Also]



None

RC_TEXTURE_THR_SIZE

[Description]

Defines the number of thresholds for bit rate control in texture.

[Syntax]

```
#define RC_TEXTURE_THR_SIZE 12
```

[Note]

None

[See Also]

None

H264E_NALU_TYPE_E

[Description]

Defines the NALU type of an H.264 stream.

[Syntax]

```
typedef enum hiH264E_NALU_TYPE_E
{
    H264E_NALU_PSLICE = 1,
    H264E_NALU_ISLICE = 5,
    H264E_NALU_SEI = 6,
    H264E_NALU_SPS = 7,
    H264E_NALU_PPS = 8,
    H264E_NALU_IPSLICE = 9,
    H264E_NALU_BUTT
} H264E_NALU_TYPE_E;
```

[Member]

Member	Description
H264E_NALU_PSLICE	PSLICE
H264E_NALU_ISLICE	PSLICE
H264E_NALU_SEI	SEI
H264E_NALU_SPS	SPS
H264E_NALU_PPS	PPS

[Note]



None

[See Also]

None

H264E_REFSLICE_TYPE_E

[Description]

Defines the reference frame type in a specific frame skipping reference mode based on the obtained H.264 streams.

[Syntax]

```
typedef enum hiH264E_REFSLICE_TYPE_E
{
    H264E_REFSLICE_FOR_1X      = 1,
    H264E_REFSLICE_FOR_2X      = 2,
    H264E_REFSLICE_FOR_4X      = 5,
    H264E_REFSLICE_FOR_BUTT
} H264E_REFSLICE_TYPE_E;
```

[Member]

Member	Description
H264E_REFSLICE_FOR_1X	Reference frame in 1x frame skipping reference mode
H264E_REFSLICE_FOR_2X	Reference frame in 2x frame skipping reference mode or reference frame in 4x frame skipping reference mode for 2x frame skipping reference
H264E_REFSLICE_FOR_4X	Reference frame in 4x frame skipping reference mode
H264E_REFSLICE_BUTT	Non-reference mode

[Note]

None

[See Also]

None

H264E_REF_TYPE_E

[Description]

Defines the frame type and reference attributes of the H.264 frame skipping reference streams.

[Syntax]

```
typedef enum hiH264E_REF_TYPE_E
{
    BASE_IDRSLICE = 0,           //IDR frame at the base layer
```



```
BASE_REFTOIDR = 1,           //Virtual I frame
BASE_PSLICE_REFBYBASE,      //P frame at the base layer,
referenced by other frames at the base layer
BASE_PSLICE_REFBYENHANCE,    //P frame at the base layer,
referenced by frames at the enhance layer
ENHANCE_PSLICE_REFBYENHANCE, //P frame at the enhance layer,
referenced by other frames at the enhance layer
ENHANCE_PSLICE_NOTFORREF,    //P frame at the enhance layer, not
referenced
ENHANCE_PSLICE_BUTT
} H264E_REF_TYPE_E;
```

[Member]

Member	Description
BASE_IDRSLICE	IDR frame at the base layer
BASE_REFTOIDR	Virtual I frame
BASE_PSLICE_REFBYBASE	P frame at the base layer, referenced by other frames at the base layer
BASE_PSLICE_REFBYENHANCE	P frame at the base layer, referenced by frames at the enhance layer
ENHANCE_PSLICE_REFBYENHANCE	P frame at the enhance layer, referenced by other frames at the enhance layer
ENHANCE_PSLICE_NOTFORREF	P frame at the enhance layer, not referenced

[Note]

None

[See Also]

None

H265E_REF_TYPE_E

[Description]

Defines the frame type and reference attributes of the H.265 frame skipping reference streams.

[Syntax]

```
TypeDef enum hiH264E_REF_TYPE_E H265E_REF_TYPE_E;
```

[Member]

See [H264E_REF_TYPE_E](#).

[Note]



None

[See Also]

[H264E_REF_TYPE_E](#)

JPEGE_PACK_TYPE_E

[Description]

Defines the PACK type of a JPEG stream.

[Syntax]

```
typedef enum hiJPEGE_PACK_TYPE_E
{
    JPEGE_PACK_ECS = 5,
    JPEGE_PACK_APP = 6,
    JPEGE_PACK_VDO = 7,
    JPEGE_PACK_PIC = 8,
    JPEGE_PACK_BUTT
} JPEGE_PACK_TYPE_E;
```

[Member]

Member	Description
JPEGE_PACK_ECS	ECS
JPEGE_PACK_APP	APP
JPEGE_PACK_VDO	VDO
JPEGE_PACK_PIC	PIC

[Note]

None

[See Also]

None

MPEG4E_PACK_TYPE_E

[Description]

Defines the Pack type of an MPEG-4 stream.

[Syntax]

```
typedef enum hiMPEG4E_PACK_TYPE_E
{
    MPEG4E_PACK_VOP_P = 1,
    MPEG4E_PACK_VOP_I = 5,
```



```
MPEG4E_PACK_VOS = 6,  
MPEG4E_PACK_VO = 7,  
MPEG4E_PACK_VOL = 8,  
MPEG4E_PACK_GVOP = 9  
} MPEG4E_PACK_TYPE_E;
```

[Member]

Member	Description
MPEG4E_PACK_VOS	Video object sequence types
MPEG4E_PACK_VO	Video object types
MPEG4E_PACK_VOL	Video object layer types
MPEG4E_PACK_GVOP	Group of video object plane types
MPEG4E_PACK_VOP_P	Video object plane types for the P frame
MPEG4E_PACK_VOP_I	Video packet types for the I frame

[Note]

None

[See Also]

None

H265E_NALU_TYPE_E

[Description]

Defines the NALU type of an H.265 stream.

[Syntax]

```
typedef enum hiH265E_NALU_TYPE_E  
{  
    H265E_NALU_PSLICE = 1,  
    H265E_NALU_ISLICE = 19,  
    H265E_NALU_VPS = 32,  
    H265E_NALU_SPS = 33,  
    H265E_NALU_PPS = 34,  
    H265E_NALU_SEI = 39,  
    H265E_NALU_BUTT  
} H265E_NALU_TYPE_E;
```

[Member]



Member	Description
H265E_NALU_PSLICE	PSLICE type
H265E_NALU_ISLICE	ISLICE type
H265E_NALU_VPS	VPS type
H265E_NALU_SPS	SPS type
H265E_NALU_PPS	PPS type
H265E_NALU_SEI	SEI type

[Note]

None

[See Also]

None

VENC_DATA_TYPE_U

[Description]

Defines the stream result type.

[Syntax]

```
typedef union hivENC_DATA_TYPE_U
{
    H264E_NALU_TYPE_E      enH264EType;
    JPEG_E_PACK_TYPE_E     enJPEGEType;
    MPEG4_E_PACK_TYPE_E   enMPEG4EType;
    H265E_NALU_TYPE_E     enH265EType;
}VENC_DATA_TYPE_U;
```

[Member]

Member	Description
enH264EType	Type of an H.264 stream packet
enJPEGEType	Type of a JPEG stream packet
enMPEG4EType	Type of an MPEG-4 stream packet
enH265EType	Type of the H.265 stream packet

[Note]

None

[See Also]



- [H264E_NALU_TYPE_E](#)
- [JPEGE_PACK_TYPE_E](#)
- [MPEG4E_PACK_TYPE_E](#)
- [H265E_NALU_TYPE_E](#)

VENC_PACK_INFO_S

[Description]

Defines the structure of other types of stream packet data which is contained in the current stream packet data.

[Syntax]

```
typedef struct hiVENC_PACK_INFO_S
{
    VENC\_DATA\_TYPE\_U u32PackType;
    HI_U32 u32PackOffset;
    HI_U32 u32PackLength;
} VENC_PACK_INFO_S;
```

[Member]

Member	Description
u32PackType	Type of other stream packets in the current stream packet data
u32PackOffset	Offset of other types of stream packet data in the current stream packet data
u32PackLength	Size of other types of stream packet data in the current stream packet data

VENC_PACK_S

[Description]

Defines a stream packet.

[Syntax]

```
typedef struct hiVENC_PACK_S
{
    HI_U32             u32PhyAddr;
    HI_U8              *pu8Addr;
    HI_U32             u32Len;
    HI_U64             u64PTS;
    HI_BOOL            bFrameEnd;
    VENC\_DATA\_TYPE\_U   DataType;
    HI_U32             u32Offset;
    HI_U32             u32DataNum;
```



```
VENC_PACK_INFO_S     stPackInfo[8];  
}VENC_PACK_S;
```

[Member]

Member	Description
pu8Addr	Initial address of a stream packet
u32PhyAddr	Physical address of a stream packet
u32Len	Length of a stream packet
DataType	Type of a stream. H.264, JPEG, and MPEG-4 packets are supported.
u64PTS	Time stamp, in μ s
bFrameEnd	Flag of ending a frame Value range: HI_TRUE : the last stream packet of a frame HI_FALSE : the stream packet is not the last one of a field
u32Offset	Valid data in the stream packet and offset of the start address pu8Addr for stream packet
u32DataNum	Number of other stream packets contained in the current stream packet (the type of the current stream packet is specified by DataType) data
stPackInfo[8]	Information about other types of stream packet data in the current stream packet data

[Note]

None

[See Also]

[VENC_DATA_TYPE_U](#)

VENC_STREAM_INFO_H264_S

[Description]

Defines the features of an H.264 stream.

[Syntax]

```
typedef struct hivENC_STREAM_INFO_H264_S  
{  
    HI_U32 u32PicBytesNum;  
    HI_U32 u32PSkipMbNum;  
    HI_U32 u32IpcmMbNum;  
    HI_U32 u32Inter16x8MbNum;
```



```
HI_U32 u32Inter16x16MbNum;
HI_U32 u32Inter8x16MbNum;
HI_U32 u32Inter8x8MbNum;
HI_U32 u32Intra16MbNum;
HI_U32 u32Intra8MbNum;
HI_U32 u32Intra4MbNum;
H264E_REFSLICE_TYPE_E enRefSliceType;
H264E_REF_TYPE_E enRefType;
HI_U32 u32UpdateAttrCnt;
HI_U32 u32StartQp;
}VENC_STREAM_INFO_H264_S;
```

[Member]

Member	Description
u32PicBytesNum	Number of bytes to be encoded in the current frame
u32PSkipMbNum	Number of macroblocks in skip encoding mode to be encoded in the current frame
u32IpcmMbNum	Number of macroblocks in IPCM encoding mode to be encoded in the current frame
u32Inter16x8MbNum	Number of macroblocks in 16x8 inter-prediction encoding mode to be encoded in the current frame
u32Inter16x16MbNum	Number of macroblocks in 16x16 inter-prediction encoding mode to be encoded in the current frame
u32Inter8x16MbNum	Number of macroblocks in 8x16 inter-prediction encoding mode to be encoded in the current frame
u32Inter8x8MbNum	Number of macroblocks in 8x8 inter-prediction encoding mode to be encoded in the current frame
u32Intra16MbNum	Number of macroblocks in intra16 prediction encoding mode to be encoded in the current frame
u32Intra8MbNum	Number of macroblocks in intra8 prediction encoding mode to be encoded in the current frame
u32Intra4MbNum	Number of macroblocks in intra4 prediction encoding mode to be encoded in the current frame
enRefSliceType	Encode the frame whose reference frame is in a specific frame skipping reference mode
enRefType	Type of the encoded frame in advanced frame skipping reference mode
u32UpdateAttrCnt	Number of times that channel attributes or parameters (including RC parameter) are configured
u32StartQp	Start QP value of the frame that is being encoded



[Note]

You can store only the H.264 streams whose reference frames are in a specific frame skipping reference mode. The details are as follows:

- In 1x frame skipping reference mode, you can store only the streams whose **enRefSliceType** value is **H264E_REFSLICE_FOR_1X**.
- In 2x frame skipping reference mode, you can store only the streams whose **enRefSliceType** value is **H264E_REFSLICE_FOR_2X**.
- In 4x frame skipping reference mode, you can store only the streams whose **enRefSliceType** value is **H264E_REFSLICE_FOR_4X** or store the streams whose **enRefSliceType** value is **H264E_REFSLICE_FOR_2X** or **H264E_REFSLICE_FOR_4X**.

[See Also]

- [VENC_DATA_TYPE_U](#)
- [H264E_REFSLICE_TYPE_E](#)

VENC_STREAM_INFO_JPEG_S

[Description]

Defines the features of a JPEG/MJPEG stream.

[Syntax]

```
typedef struct hiVENC_STREAM_INFO_JPEG_S
{
    HI_U32 u32PicBytesNum;
    HI_U32 u32UpdateAttrCnt;
    HI_U32 u32Qfactor;
}VENC_STREAM_INFO_JPEG_S;
```

[Member]

Member	Description
u32PicBytesNum	Size of a JPEG stream, in byte
u32UpdateAttrCnt	Number of times that channel attributes or parameters (including RC parameter) are set
u32Qfactor	Qfactor value of the currently encoded frame

[Note]

None

[See Also]

[VENC_DATA_TYPE_U](#)



VENC_STREAM_INFO_MPEG4_S

[Description]

Defines the features of an MPEG-4 stream.

[Syntax]

```
typedef struct hiVENC_STREAM_INFO_MPEG4_S
{
    HI_U32 u32PicBytesNum;
    HI_U32 u32UpdateAttrCnt;
}VENC_STREAM_INFO_MPEG4_S;
```

[Member]

Member	Description
u32PicBytesNum	Size of an MPEG-4 stream, in byte
u32UpdateAttrCnt	Number of times that channel attributes or parameters (including RC parameter) are set

[Note]

None

[See Also]

[VENC_DATA_TYPE_U](#)

VENC_STREAM_INFO_H265_S

[Description]

Defines the features of an H.265 stream.

[Syntax]

```
typedef struct hiVENC_STREAM_INFO_H265_S
{
    HI_U32 u32PicBytesNum;
    HI_U32 u32Inter64x64CuNum;
    HI_U32 u32Inter32x32CuNum;
    HI_U32 u32Inter16x16CuNum;
    HI_U32 u32Inter8x8CuNum;
    HI_U32 u32Intra32x32CuNum;
    HI_U32 u32Intra16x16CuNum;
    HI_U32 u32Intra8x8CuNum;
    HI_U32 u32Intra4x4CuNum;
    H265E_REF_TYPE_E enRefType;
    HI_U32 u32UpdateAttrCnt;
```



```
    HI_U32 u32StartQp;  
}VENC_STREAM_INFO_H265_S;
```

[Member]

Member	Description
u32PicBytesNum	Number of bytes to be encoded in the current frame
u32Inter64x64CuNum	Number of coding units (CUs) to be encoded in inter64x64 prediction mode in the current frame
u32Inter32x32CuNum	Number of CUs to be encoded in inter32x32 prediction mode in the current frame
u32Inter16x16CuNum	Number of CUs to be encoded in inter16x16 prediction mode in the current frame
u32Inter8x8CuNum	Number of CUs to be encoded in inter8x8 prediction mode in the current frame
u32Intra32x32CuNum	Number of CUs to be encoded in intra32x32 prediction mode in the current frame
u32Intra16x16CuNum	Number of CUs to be encoded in intra16x16 prediction mode in the current frame
u32Intra8x8CuNum	Number of CUs to be encoded in intra8x8 prediction mode in the current frame
u32Intra4x4CuNum	Number of CUs to be encoded in intra4x4 prediction mode in the current frame
enRefType	Type of the encoded frame in advanced frame skipping reference mode
u32UpdateAttrCnt	Number of times that channel attributes or parameters (including RC parameter) are configured
u32StartQp	Start QP value of the frame that is being encoded

[Note]

For details about **enRefType**, see [VENC_STREAM_INFO_H264_S](#).

[See Also]

- [VENC_DATA_TYPE_U](#)
- [H265E_REF_TYPE_E](#)

VENC_STREAM_S

[Description]

Defines the stream frame type.

[Syntax]



```
typedef struct hiVENC_STREAM_S
{
    VENC_PACK_S *pstPack;
    HI_U32       u32PackCount;
    HI_U32       u32Seq;
    union
    {
        VENC_STREAM_INFO_H264_S stH264Info;
        VENC_STREAM_INFO_JPEG_S stJpegInfo;
        VENC_STREAM_INFO_MPEG4_S stMpeg4Info;
        VENC_STREAM_INFO_H265_S stH265Info;
    };
} VENC_STREAM_S
```

[Member]

Member	Description
pstPack	Structure of a stream frame
u32PackCount	Number of stream packets per frame
u32Seq	Sequence number of a stream The sequence number of a frame is obtained by frame and the sequence number of a packet is obtained by packet.
stH264Info/stJpegInfo/stMpeg4Info/stH265Info	Features of a stream

[Note]

None

[See Also]

- [VENC_PACK_S](#)
- [HI_MPI_VENC_GetStream](#)

VENC_STREAM_BUF_INFO_S

[Description]

Defines the stream buffer information.

[Syntax]

```
typedef struct hiVENC_STREAM_BUF_INFO_S
{
    HI_U32   u32PhyAddr;
    HI_VOID  *pUserAddr;
    HI_U32   u32BufSize;
```



```
} VENC_STREAM_BUF_INFO_S;
```

[Member]

Member	Description
u32PhyAddr	Start physical address for a stream buffer
pUserAddr	Virtual address for a stream buffer
u32BufSize	Stream buffer size

[Note]

None

[See Also]

[HI_MPI_VENC_GetStreamBufInfo](#)

VENC_ATTR_H264_S

[Description]

Defines the attribute structure for H.264 encoding.

[Syntax]

```
typedef struct hiVENC_ATTR_H264_S
{
    HI_U32    u32MaxPicWidth;
    HI_U32    u32MaxPicHeight;
    HI_U32    u32BufSize;
    HI_U32    u32Profile;
    HI_BOOL   bByFrame;
    HI_U32    u32PicWidth;
    HI_U32    u32PicHeight;
    HI_U32    u32BFrameNum;
    HI_U32    u32RefNum;
} VENC_ATTR_H264_S;
```

[Member]

Member	Description
u32MaxPicWidth	Maximum width of a picture to be encoded, in pixel Value range: [MIN_WIDTH, MAX_WIDTH] The value must be an integral multiple of MIN_ALIGN . Static attribute
u32PicWidth	Width of a picture to be encoded, in pixel Value range: [MIN_WIDTH, MAX_WIDTH]



Member	Description
	The value must be an integral multiple of MIN_ALIGN .
u32MaxPicHeight	Maximum height of a picture to be encoded, in pixel Value range: [MIN_WIDTH, MAX_HEIGHT] The value must be an integral multiple of MIN_ALIGN . Static attribute
u32PicHeight	Height of a picture to be encoded, in pixel Value range: [MIN_HEIGHT, MAX_HEIGHT] The value must be an integral multiple of MIN_ALIGN .
u32BufSize	Stream buffer size, in byte Value range: [Min, Max] The value must be an integral multiple of 64. You are advised to set the buffer size to the maximum size of a picture to be encoded. That is, the recommended value is as follows: u32MaxPicWidth x u32MaxPicHeight x 1.5 bytes Static attribute The module parameter H264eMiniBufMode can be configured when the .ko driver is loaded to select the mode for limiting the value range. <ul style="list-style-type: none">• H264eMiniBufMode = 0<ul style="list-style-type: none">Minimum value: 1/2 of the maximum size of a picture to be encodedMaximum value: There is no limitation but much memory is consumed if the buffer size is large.• H264eMiniBufMode = 1 (the user needs to ensure that the buffer size is appropriate)<ul style="list-style-type: none">Minimum value: 32 x 1024 bytesMaximum value: There is no limitation but much memory is consumed if the buffer size is large.
bByFrame	Obtaining streams by frame/packet Value range: [HI_TRUE, HI_FALSE] <ul style="list-style-type: none">• HI_TRUE: obtain streams by frame• HI_FALSE: obtain streams by packet Static attribute
u32Profile	Encoding level Value range: [0, 3] 0: baseline 1: main profile 2: high profile 3: svc-t
u32BFrameNum	Number of supported B frames to be encoded



Member	Description
	<p>Value range: [0, Max] Max: no limit u32BFrameNum is reserved for the Hi3516A/Hi3518EV200 and is not supported currently.</p>
u32RefNum	<p>Number of reference frames supported during H.264 encoding Value range: [1, 2] u32RefNum is reserved for the Hi3516A/Hi3518EV200 and is not supported currently.</p>

[Note]

None

[See Also]

None

VENC_ATTR_MJPEG_S

[Description]

Defines the attribute of the MJPEG encoding.

[Syntax]

```
typedef struct hivENC_ATTR_MJPEG_S
{
    HI_U32    u32MaxPicWidth;
    HI_U32    u32MaxPicHeight;
    HI_U32    u32BufSize;
    HI_BOOL   bByFrame;
    HI_U32    u32PicWidth;
    HI_U32    u32PicHeight;
}VENC_ATTR_MJPEG_S;
```

[Member]

Member	Description
u32MaxPicWidth	<p>Maximum width of a picture to be encoded, in pixel Value range: [MIN_WIDTH, MAX_WIDTH]. The value must be an integral multiple of MIN_ALIGN. Static attribute</p>
u32PicWidth	<p>Width of a picture to be encoded, in pixel Value range: [MIN_WIDTH, MAX_WIDTH] The value must be an integral multiple of MIN_ALIGN.</p>



Member	Description
u32MaxPicHeight	Maximum height of a picture to be encoded, in pixel Value range: [MIN_HEIGHT, MAX_HEIGHT]. The value must be an integral multiple of MIN_ALIGN. Static attribute
u32PicHeight	Height of a picture to be encoded, in pixel Value range: [MIN_HEIGHT, MAX_HEIGHT] The value must be an integral multiple of MIN_ALIGN.
u32BufSize	Buffer size The value must be greater than or equal to the maximum picture width multiplied by the maximum picture height after 16-pixel alignment, and must be an integral multiple of 64. Static attribute The module parameter JpegeMiniBufMode can be configured when the .ko driver is loaded to select the mode for limiting the value range. <ul style="list-style-type: none">• JpegeMiniBufMode = 0<ul style="list-style-type: none">Value range: greater than or equal to the product of the 16-pixel-aligned maximum picture width and maximum picture height• JpegeMiniBufMode = 1 (the user needs to ensure that the buffer size is appropriate)<ul style="list-style-type: none">Value range: greater than or equal to 32 x 1024 bytesMaximum value: There is no limitation but much memory is consumed if the buffer size is large.
bByFrame	Stream acquisition mode Value range: [HI_TRUE, HI_FALSE] <ul style="list-style-type: none">• HI_TRUE: obtains streams by frame.• HI_FALSE: obtains streams by packet. Static attribute

[Note]

None

[See Also]

None

VENC_ATTR_JPEG_S

[Description]

Defines the attribute of the JPEG snapshot.

[Syntax]



```
typedef struct hiVENC_ATTR_JPEG_S
{
    HI_U32    u32MaxPicWidth;
    HI_U32    u32MaxPicHeight;
    HI_U32    u32BufSize;
    HI_BOOL   bByFrame;
    HI_U32    u32PicWidth;
    HI_U32    u32PicHeight;
    HI_BOOL   bSupportDCF;
}VENC_ATTR_JPEG_S;
```

[Member]

Member	Description
u32MaxPicWidth	Maximum width of a picture to be encoded, in pixel Value range: [MIN_WIDTH, MAX_WIDTH] The value must be an integral multiple of MIN_ALIGN. Static attribute
u32PicWidth	Width of a picture to be encoded Value range: [MIN_WIDTH, MAX_WIDTH] The value must be an integral multiple of MIN_ALIGN.
u32MaxPicHeight	Maximum height of a picture to be encoded, in pixel Value range: [MIN_HEIGHT, MAX_HEIGHT] The value must be an integral multiple of MIN_ALIGN. Static attribute
u32PicHeight	Height of a picture to be encoded Value range: [MIN_HEIGHT, MAX_HEIGHT] The value must be an integral multiple of MIN_ALIGN.
u32BufSize	Buffer size. The value must be an integral multiple of 64. Static attribute The module parameter JpegeMiniBufMode can be configured when the .ko driver is loaded to select the mode for limiting the value range. <ul style="list-style-type: none">• JpegeMiniBufMode = 0<ul style="list-style-type: none">Value range: greater than or equal to the product of the 16-pixel-aligned maximum picture width and maximum picture height• JpegeMiniBufMode = 1 (the user needs to ensure that the buffer size is appropriate)<ul style="list-style-type: none">Value range: greater than or equal to 32 x 1024 bytesMaximum value: There is no limitation but much memory is consumed if the buffer size is large.



Member	Description
bByFrame	Obtaining streams by frame/packet Value range: [HI_TRUE, HI_FALSE] • HI_TRUE : obtains streams by frame. • HI_FALSE : obtains streams by packet. Static attribute
bSupportDCF	Whether the JPEG picture has a thumbnail Static attribute

[Note]

None

[See Also]

None

VENC_ATTR_MPEG4_S

[Description]

Defines the attribute of the MPEG-4 channel.

[Syntax]

```
typedef struct hivENC_ATTR_MPEG4_S
{
    HI_U32 u32MaxPicWidth;
    HI_U32 u32MaxPicHeight;
    HI_U32 u32BufSize;
    HI_BOOL bByFrame;
    HI_U32 u32PicWidth;
    HI_U32 u32PicHeight;
} VENC_ATTR_MPEG4_S;
```

[Member]

Member	Description
u32MaxPicWidth	Maximum width of a picture to be encoded, in pixel Value range: [MIN_WIDTH, MAX_WIDTH] The value must be an integral multiple of MIN_ALIGN. Static attribute
u32PicWidth	Width of a picture to be encoded Value range: [MIN_WIDTH, MAX_WIDTH] The value must be an integral multiple of MIN_ALIGN. Static attribute



Member	Description
u32MaxPicHeight	Maximum height of a picture to be encoded, in pixel Value range: [MIN_HEIGHT, MAX_HEIGHT] The value must be an integral multiple of MIN_ALIGN. Static attribute
u32PicHeight	Height of a picture to be encoded Value range: [MIN_HEIGHT, MAX_HEIGHT] The value must be an integral multiple of MIN_ALIGN. Static attribute
u32BufSize	Stream buffer size, in byte Value range: [Min, Max] The value must be an integral multiple of 64. You are advised to set the buffer size to the maximum size of a picture to be encoded. That is, the recommended value is as follows: u32MaxPicWidth x u32MaxPicHeight x1.5 bytes Minimum value: 1/2 of the maximum size of a picture to be encoded Maximum value: There is no limitation but much memory is consumed if the buffer size is large. Static attribute
bByFrame	Mode for obtaining streams Values: {HI_TRUE, HI_FALSE} <ul style="list-style-type: none">• HI_TRUE: obtain streams by frame• HI_FALSE: obtain streams by packet Static attribute

[Note]

None

[See Also]

None

VENC_ATTR_H265_S

[Description]

Defines the attribute of the H.265 encoder.

[Syntax]

```
typedef struct hiVENC_ATTR_H265_S
{
    HI_U32    u32MaxPicWidth;
    HI_U32    u32MaxPicHeight;
    HI_U32    u32BufSize;
```



```
    HI_U32 u32Profile;
    HI_BOOL bByFrame;
    HI_U32 u32PicWidth;
    HI_U32 u32PicHeight;
    HI_U32 u32BFrameNum;
    HI_U32 u32RefNum;
}VENC_ATTR_H265_S;
```

[Member]

Member	Description
u32MaxPicWidth	Maximum width of a picture to be encoded, in pixel Value range: [MIN_WIDTH, MAX_WIDTH] The value must be an integral multiple of MIN_ALIGN . Static attribute
u32PicWidth	Width of a picture to be encoded Value range: [MIN_WIDTH, MAX_WIDTH] The value must be an integral multiple of MIN_ALIGN .
u32MaxPicHeight	Maximum height of a picture to be encoded, in pixel Value range: [MIN_HEIGHT, MAX_HEIGHT] The value must be an integral multiple of MIN_ALIGN . Static attribute
u32PicHeight	Height of a picture to be encoded Value range: [MIN_HEIGHT, MAX_HEIGHT] The value must be an integral multiple of MIN_ALIGN .
u32BufSize	Stream buffer size, in byte Value range: [Min, Max] The value must be an integral multiple of 64. Static attribute You are advised to set the buffer size to the maximum size of a picture to be encoded. That is, the recommended value is as follows: u32MaxPicWidth x u32MaxPicHeight x1.5 bytes The module parameter H265eMiniBufMode can be configured when the .ko driver is loaded to select the mode for limiting the value range. <ul style="list-style-type: none">• H265eMiniBufMode = 0<ul style="list-style-type: none">Minimum value: 1/2 of the maximum size of a picture to be encodedMaximum value: There is no limitation but much memory is consumed if the buffer size is large.• H265eMiniBufMode = 1 (the user needs to ensure that the buffer size is appropriate)<ul style="list-style-type: none">Minimum value: 32 x 1024 bytes



Member	Description
	Maximum value: There is no limitation but much memory is consumed if the buffer size is large.
bByFrame	Mode for obtaining streams Value range: {HI_TRUE, HI_FALSE} HI_TRUE: obtain streams by frame. HI_FALSE: obtain streams by packet. Static attribute
u32Profile	Encoding level Value: 0 (main profile) Static attribute
u32BFrameNum	Number of supported B frames to be encoded Value range: [0, Max] Max: no limit u32BFrameNum is reserved for the Hi3516A and is not supported currently.
u32RefNum	Number of reference frames supported during H.265 encoding Value range: [1, 2] u32RefNum is reserved for the Hi3516A and is not supported currently.

[Note]

None

[See Also]

None

VENC_ATTR_S

[Description]

Defines the encoder attributes.

[Syntax]

```
typedef struct hivENC_ATTR_S
{
    PAYLOAD_TYPE_E enType;
    union
    {
        VENC_ATTR_H264_S stAttrH264;
        VENC_ATTR_MJPEG_S stAttrMjpeg;
        VENC_ATTR_JPEG_S stAttrJpeg;
        VENC_ATTR_MPEG4_S stAttrMpeg4;
    }
}
```



```
VENC_ATTR_H265_S stAttrH265;  
};  
}VENC_ATTR_S;
```

[Member]

Member	Description
enType	Protocol type
stAttrH264/stAttrMjpeg/stAttrJpeg/stAttrMppeg4/stAttrH265	Attribute of a protocol-compliant encoder

[Note]

None

[See Also]

[HI_MPI_VENC_CreateChn](#)

VENC_CHN_ATTR_S

[Description]

Defines the attribute of a VENC channel.

[Syntax]

```
typedef struct hIVENC_CHN_ATTR_S  
{  
    VENC_ATTR_S     stVeAttr;  
    VENC_RC_ATTR_S stRcAttr;  
}VENC_CHN_ATTR_S;
```

[Member]

Member	Description
stVeAttr	Encoder attributes
stRcAttr	RC attributes

[Note]

None

[See Also]

[HI_MPI_VENC_CreateChn](#)

VENC_CHN_STAT_S

[Description]



Defines the status of a VENC channel.

[Syntax]

```
typedef struct hiVENC_CHN_STAT_S
{
    HI_U32    u32LeftPics;
    HI_U32    u32LeftStreamBytes;
    HI_U32    u32LeftStreamFrames;
    HI_U32    u32CurPacks;
    HI_U32    u32LeftRecvPics;
    HI_U32    u32LeftEncPics;
} VENC_CHN_STAT_S;
```

[Member]

Member	Description
u32LeftPics	Number of pictures to be encoded
u32LeftStreamBytes	Number of remaining bytes in a stream buffer
u32LeftStreamFrames	Number of remaining frames in a stream buffer
u32CurPacks	Number of stream packets in the current frame
u32LeftRecvPics	Number of frames to be received. This member is valid after HI_MPI_VENC_StartRecvPicEx is called.
u32LeftEncPics	Number of frames to be encoded. This member is valid after HI_MPI_VENC_StartRecvPicEx is called.

[Note]

None

[See Also]

[HI_MPI_VENC_Query](#)

VENC_PARAM_H264_SLICE_SPLIT_S

[Description]

Defines the slice structure of an H.264 channel.

[Syntax]

```
typedef struct hiVENC_PARAM_H264_SLICE_SPLIT_S
{
    HI_BOOL bSplitEnable;
    HI_U32   u32SplitMode;
    HI_U32   u32SliceSize;
} VENC_PARAM_H264_SLICE_SPLIT_S;
```



[Member]

Member	Description
bSplitEnable	Whether the slice function is enabled
u32SplitMode	Slice mode <ul style="list-style-type: none">• 0: slice a frame by byte.• 1: slice a frame by rows of macroblocks. This member must be set to be less than 2 .
u32SliceSize	When u32SplitMode is set to 0 , it indicates the number of byte per slice Minimum value: 128 Maximum value: min((0xFFFF), u32PicSize/2) where, u32PicSize = picwidth x picheight x 3/2 When u32SplitMode is set to 1 , it indicates the number of rows of macroblocks occupied by each slice. Minimum value: 1 Maximum value: (Picture height + 15)/16

[Note]

None

[See Also]

- [HI_MPI_VENC_SetH264SliceSplit](#)
- [HI_MPI_VENC_GetH264SliceSplit](#)

VENC_PARAM_H264_INTER_PRED_S

[Description]

Defines the inter-prediction structure of an H.264 channel.

[Syntax]

```
typedef struct hivENC_PARAM_H264_INTER_PRED_S
{
    /* search window */
    HI_U32 u32HWSIZE;
    HI_U32 u32VWSIZE;
    HI_BOOL bInter16x16PredEn;
    HI_BOOL bInter16x8PredEn;
    HI_BOOL bInter8x16PredEn;
    HI_BOOL bInter8x8PredEn;
    HI_BOOL bExtedgeEn;
} VENC_PARAM_H264_INTER_PRED_S;
```



[Member]

Member	Description
u32HWSIZE	Size of a horizontal search window
u32VWSIZE	Size of a vertical search window
bInter16x16PredEn	16x16 inter-prediction. By default, this function is enabled.
bInter16x8PredEn	16x8 inter-prediction enable. By default, this function is enabled.
bInter8x16PredEn	8x16 inter-prediction enable. By default, this function is enabled.
bInter8x8PredEn	8x8 inter-prediction. By default, this function is enabled.
bExtedgeEn	When a search window is larger than a picture, this member indicates whether to enable edge extension of the picture. By default, edge extension is enabled.

[Note]

None

[See Also]

- [HI_MPI_VENC_SetH264InterPred](#)
- [HI_MPI_VENC_GetH264InterPred](#)

VENC_PARAM_H264_INTRA_PRED_S

[Description]

Defines the intra-prediction structure of an H.264 channel.

[Syntax]

```
typedef struct hivenc_param_h264_intra_pred_s
{
    HI_BOOL bIntra16x16PredEn;
    HI_BOOL bIntraNxNPredEn;
    HI_U32 constrained_intra_pred_flag;
    HI_BOOL bIpcmEn;
} VENC_PARAM_H264_INTRA_PRED_S;
```

[Member]

Member	Description
bIntra16x16PredEn	16x16 intra-prediction enable. By default, this function is enabled. Value:



Member	Description
	<ul style="list-style-type: none">• 0: disabled• 1: enabled
bIntraNxNPredEn	<p><i>NxN</i> intra-prediction enable. By default, this function is enabled.</p> <p>Value:</p> <ul style="list-style-type: none">• 0: disabled• 1: enabled
constrained_intra_pred_flag;	<p>Default value: 0</p> <p>Value: 0 or 1</p>
bIpcmEn	<p>IP camera prediction enable. The default value varies depending on chip type.</p> <p>Value: 0 or 1</p>

[Note]

For the specific meanings of the members, see the H.264 protocol.

[See Also]

- [HI_MPI_VENC_SetH264IntraPred](#)
- [HI_MPI_VENC_GetH264IntraPred](#)

VENC_PARAM_H264_TRANS_S

[Description]

Defines the transformation and quantization structure of an H.264 channel.

[Syntax]

```
typedef struct hiVENC_PARAM_H264_TRANS_S
{
    HI_U32 u32IntraTransMode;
    HI_U32 u32InterTransMode;
    HI_BOOL bScalingListValid;
    HI_U8  InterScalingList8X8[64];
    HI_U8  IntraScalingList8X8[64];
    HI_S32 chroma_qp_index_offset;
}VENC_PARAM_H264_TRANS_S;
```

[Member]

Member	Description
u32IntraTransMode	<p>Conversion mode for intra-prediction</p> <ul style="list-style-type: none">• 0: supports 4x4 and 8x8 transformation and high profile.



Member	Description
	<ul style="list-style-type: none">• 1: supports 4x4 transformation and baseline, main, and high profiles.• 2: supports 8x8 transformation and high profile. <p>The system sets this member based on the channel protocol type.</p>
u32InterTransMode	Transformation mode for inter-prediction <ul style="list-style-type: none">• 0: supports 4x4 and 8x8 transformation and high profile.• 1: supports 4x4 transformation and baseline, main, and high profiles.• 2: supports 8x8 transformation and high profile. <p>The system sets this member based on the channel protocol type.</p>
bScalingListValid	InterScalingList8x8 and IntraScalingList8x8 are valid only in the H.264 high profile. Value: <ul style="list-style-type: none">• 0: invalid• 1: valid
InterScalingList8x8	A quantization table for 8x8 inter-prediction. You can use your own quantization table in the high profile. Value range: [1, 255]
IntraScalingList8x8	A quantization table for 8x8 intra-prediction. You can use your own quantization table in the high profile. Value range: [1, 255]
chroma_qp_index_offset	For details, see the H.264 protocol. Default value: 0 Value range: [-12, +12]

[Note]

For the specific meanings of the members, see the H.264 protocol.

[See Also]

- [HI_MPI_VENC_SetH264Trans](#)
- [HI_MPI_VENC_GetH264Trans](#)

VENC_PARAM_H264_ENTROPY_S

[Description]

Defines the entropy structure of an H.264 channel.

[Syntax]

```
typedef struct hivENC_PARAM_H264_ENTROPY_S
```



```
{  
    HI_U32 u32EntropyEncModeI;  
    HI_U32 u32EntropyEncModeP;  
    HI_U32 cabac_stuff_en;  
    HI_U32 Cabac_init_idc;  
}VENC_PARAM_H264_ENTROPY_S;
```

[Member]

Member	Description
u32EntropyEncModeI	Entropy mode for I frame <ul style="list-style-type: none">• 0: CAVLC• 1: CABAC Other values: undefined The baseline profile does not support CABAC.
u32EntropyEncModeP	Entropy mode for P frame <ul style="list-style-type: none">• 0: CAVLC• 1: CABAC Other values: null The baseline profile does not support CABAC.
cabac_stuff_en	For details, see the H.264 protocol. By default, this member is set to 0. Value: 0 or 1
Cabac_init_idc	For details, see the H.264 protocol. The value range is 0–2. The default value is 0.

[Note]

None

[See Also]

- [HI_MPI_VENC_SetH264Entropy](#)
- [HI_MPI_VENC_GetH264Entropy](#)

VENC_PARAM_H264_POC_S

[Description]

Defines the POC structure of an H.264 channel.

[Syntax]

```
typedef struct hivENC_PARAM_H264_POC_S  
{  
    HI_U32 pic_order_cnt_type;  
}VENC_PARAM_H264_POC_S;
```



[Member]

Member	Description
pic_order_cnt_type	The value range is 0–2, and the default value is 2. For details, see the H.264 protocol.

[Note]

None

[See Also]

- [HI_MPI_VENC_SetH264Poc](#)
- [HI_MPI_VENC_GetH264Poc](#)

VENC_PARAM_H264_DBLK_S

[Description]

Defines the de-blocking structure of an H.264 channel.

[Syntax]

```
typedef struct hiVENC_PARAM_H264_DBLK_S
{
    HI_U32 disable_deblocking_filter_idc;
    HI_S32 slice_alpha_c0_offset_div2;
    HI_S32 slice_beta_offset_div2;
}VENC_PARAM_H264_DBLK_S;
```

[Member]

Member	Description
disable_deblocking_filter_idc	The value range is [0, 2], and the default value is 0. For details, see the H.264 protocol.
slice_alpha_c0_offset_div2	The value range is [-6, +6], and the default value is 5. For details, see the H.264 protocol.
slice_beta_offset_div2	The value range is [-6, +6], and the default value is 5. For details, see the H.264 protocol.

[Note]

None

[See Also]

- [HI_MPI_VENC_SetH264Dbblk](#)
- [HI_MPI_VENC_GetH264Dbblk](#)



VENC_PARAM_H264_VUI_S

[Description]

Defines the VUI structure of an H.264 channel.

[Syntax]

```
typedef struct hiVENC_PARAM_H264_VUI_S
{
    VENC_PARAM_VUI_ASPECT_RATIO_S      stVuiAspectRatio;
    VENC_PARAM_VUI_H264_TIME_INFO_S    stVuiTimeInfo;
    VENC_PARAM_VUI_VIDEO_SIGNAL_S     stVuiVideoSignal;
    VENC_PARAM_VUI_BITSTREAM_RESTRIC_S stVuiBitstreamRestric;
} VENC_PARAM_H264_VUI_S;
```

[Member]

Member	Description
stVuiAspectRatio	For details, see the H.264 protocol.
stVuiTimeInfo	For details, see the H.264 protocol.
stVuiVideoSignal	For details, see the H.264 protocol.
stVuiBitstreamRestric	For details, see the H.264 protocol.

[Note]

None

[See Also]

- [HI_MPI_VENC_SetH264Vui](#)
- [HI_MPI_VENC_GetH264Vui](#)

VENC_PARAM_H265_VUI_S

[Description]

Defines the VUI structure of an H.265 channel.

[Syntax]

```
typedef struct hiVENC_PARAM_H265_VUI_S
{
    VENC_PARAM_VUI_ASPECT_RATIO_S      stVuiAspectRatio;
    VENC_PARAM_VUI_VIDEO_SIGNAL_S     stVuiVideoSignal;
    VENC_PARAM_VUI_H265_TIME_INFO_S   stVuiTimeInfo;
    VENC_PARAM_VUI_BITSTREAM_RESTRIC_S stVuiBitstreamRestric;
} VENC_PARAM_H265_VUI_S;
```

[Member]



Member	Description
stVuiAspectRatio	For details, see the H.265 protocol.
stVuiTimeInfo	For details, see the H.265 protocol.
stVuiVideoSignal	For details, see the H.265 protocol.
stVuiBitstreamRestric	For details, see the H.265 protocol.

[Note]

None

[See Also]

- [HI_MPI_VENC_SetH265Vui](#)
- [HI_MPI_VENC_GetH265Vui](#)

VENC_PARAM_VUI_ASPECT_RATIO_S

[Description]

Defines the AspectRatio information in the VUI of an H.264/H.265 encoding channel.

[Syntax]

```
typedef struct hivenc_param_vui_aspect_ratio_s
{
    HI_U8          aspect_ratio_info_present_flag;
    HI_U8          aspect_ratio_idc;
    HI_U8          overscan_info_present_flag;
    HI_U8          overscan_appropriate_flag;
    HI_U16         sar_width;
    HI_U16         sar_height ;
}VENC_PARAM_VUI_ASPECT_RATIO_S;
```

[Member]

Member	Description
aspect_ratio_info_present_flag	For details, see the H.264/H.265 protocol. The default value is 0. Value: 0 or 1
aspect_ratio_idc	For details, see the H.264/H.265 protocol. The default value is 1. Value range: 0–255 excluding 17–254
overscan_info_present_flag	For details, see the H.264/H.265 protocol. The default value is 0. Value: 0 or 1



Member	Description
overscan_appropriate_flag	For details, see the H.264/H.265 protocol. The default value is 0. Value: 0 or 1
sar_width	For details, see the H.264/H.265 protocol. The default value is 1. Value range: (0, 65535]. sar_width and sar_height are relatively prime.
sar_height	For details, see the H.264/H.265 protocol. The default value is 1. Value range: (0, 65535]. sar_height and sar_width are relatively prime.

[Note]

Note

[See Also]

- [VENC_PARAM_H264_VUI_S](#)
- [VENC_PARAM_H265_VUI_S](#)

VENC_PARAM_VUI_H264_TIME_INFO_S

[Description]

Defines the TIME_INFO information in the VUI of an H.264 encoding channel.

[Syntax]

```
typedef struct hivENC_PARAM_H264_VUI_TIME_INFO_S
{
    HI_U8          timing_info_present_flag;
    HI_U8          fixed_frame_rate_flag;
    HI_U32         num_units_in_tick;
    HI_U32         time_scale;
} VENC_PARAM_VUI_H264_TIME_INFO_S;
```

[Member]

Member	Description
timing_info_present_flag	For details, see the H.264 protocol. The default value is 0. Value: 0 or 1
num_units_in_tick	For details, see the H.264 protocol. The default value is 1. Value range: > 0



Member	Description
time_scale	For details, see the H.264 protocol. The default value is 60. Value range: > 0
fixed_frame_rate_flag;	For details, see the H.264 protocol. The default value is 1. Value: 0 or 1

[Note]

Note

[See Also]

[VENC_PARAM_H264_VUI_S](#)

VENC_PARAM_VUI_H265_TIME_INFO_S

[Description]

Defines the TIME_INFO information in the VUI of an H.265 encoding channel.

[Syntax]

```
typedef struct hiVENC_PARAM_H265_TIME_INFO_S
{
    HI_U32 timing_info_present_flag;
    HI_U32 num_units_in_tick;
    HI_U32 time_scale;
    HI_U32 num_ticks_poc_diff_one_minus1;
}VENC_PARAM_VUI_H265_TIME_INFO_S;
```

[Member]

Member	Description
timing_info_present_flag	For details, see the H.265 protocol. The default value is 0. Value: 0 or 1
num_units_in_tick	For details, see the H.265 protocol. The default value is 1. Value range: > 0
time_scale	For details, see the H.265 protocol. The default value is 60. Value range: > 0
num_ticks_poc_diff_one_minus1	For details, see the H.265 protocol. The default value is 1. Value range: [0, 2^32-2]



[Note]

Note

[See Also]

[VENC_PARAM_H265_VUI_S](#)

VENC_PARAM_VUI_VIDEO_SIGNAL_S

[Description]

Defines the VIDEO_SIGNAL information in the VUI of an H.264/H.265 encoding channel.

[Syntax]

```
typedef struct hiVENC_PARAM_VUI_VIDEO_SIGNAL_S
{
    HI_U8     video_signal_type_present_flag;
    HI_U8     video_format;
    HI_U8     video_full_range_flag;
    HI_U8     colour_description_present_flag;
    HI_U8     colour_primaries;
    HI_U8     transfer_characteristics;
    HI_U8     matrix_coefficients;
}VENC_PARAM_VUI_VIDEO_SIGNAL_S;
```

[Member]

Member	Description
video_signal_type_present_flag	For details, see the H.264/H.265 protocol. The default value is 1. Value: 0 or 1
video_format	For details, see the H.264/H.265 protocol. The default value is 5. Value range: H.264: [0, 7] H.265: [0, 5]
video_full_range_flag	For details, see the H.264/H.265 protocol. The default value is 1. Value: 0 or 1
colour_description_present_flag	For details, see the H.264/H.265 protocol. The default value is 1. Value: 0 or 1
colour_primaries	For details, see the H.264/H.265 protocol. The default value is 1. Value range: [0, 255]



Member	Description
transfer_characteristics	For details, see the H.264/H.265 protocol. The default value is 1. Value range: [0, 255]
matrix_coefficients	For details, see the H.264/H.265 protocol. The default value is 1. Value range: [0, 255]

[Note]

Note

[See Also]

- [VENC_PARAM_H264_VUI_S](#)
- [VENC_PARAM_H265_VUI_S](#)

VENC_PARAM_VUI_BITSTREAM_RESTRIC_S

[Description]

Defines the Bitstream_Restriction structure in the VUI of an H.264/H.265 VENC channel.

[Syntax]

```
typedef struct hiVENC_PARAM_VUI_BITSTREAM_RESTRIC_S
{
    HI_U8 bitstream_restriction_flag ;
} VENC_PARAM_VUI_BITSTREAM_RESTRIC_S;
```

[Member]

Member	Description
bitstream_restriction_flag	For details, see the H.264/H.265 protocol. The default value is 0. Value: 0 or 1

[Note]

Note

[See Also]

- [VENC_PARAM_H264_VUI_S](#)
- [VENC_PARAM_H265_VUI_S](#)

VENC_PARAM_JPEG_S

[Description]



Defines the parameters of the JPEG encoder.

[Syntax]

```
typedef struct hiVENC_PARAM_JPEG_S
{
    HI_U32 u32Qfactor;
    HI_U8 u8YQt[64];
    HI_U8 u8CbQt[64];
    HI_U8 u8CrQt[64];
    HI_U32 u32MCUPerECS;
} VENC_PARAM_JPEG_S;
```

[Member]

Member	Description
u32Qfactor	The value range is [1, 99], and the default value is 90. For details, see the RFC2435 protocol.
u8YQt	Y quantization table Value range: [1, 255]
u8CbQt	Cb quantization table Value range: [1, 255]
u8CrQt	Cr quantization table Value range: [1, 255]
u32MCUPerECS	This member indicates the number of MCUs per ECS. By default, the value is set to 0 , indicating no ECS division. u32MCUPerECS: [0, (picwidth + 15) >> 4 x (picheight + 15) >> 4 x 2]

[Note]

None

[See Also]

- [HI_MPI_VENC_SetJpegParam](#)
- [HI_MPI_VENC_GetJpegParam](#)

VENC_PARAM_MJPEG_S

[Description]

Defines the advanced parameters of the MJPEG encoder.

[Syntax]

```
typedef struct hiVENC_PARAM_MJPEG_S
{
```



```
    HI_U8   u8YQt[64];  
    HI_U8   u8CbQt[64];  
    HI_U8   u8CrQt[64];  
    HI_U32  u32MCUPerECS;  
} VENC_PARAM_MJPEG_S;
```

[Member]

Member	Description
u8YQt	Y quantization table Value range: [1, 255]
u8CbQt	Cb quantization table Value range: [1, 255]
u8CrQt	Cr quantization table Value range: [1, 255]
u32MCUPerECS	Number of MCUs in each ECS. The default value 0 indicates that there are no ECSs. u32MCUPerECS: [0, (picwidth + 15) >> 4 x (picheight + 15) >> 4 x 2]

[Note]

None

[See Also]

- [HI_MPI_VENC_SetMjpegParam](#)
- [HI_MPI_VENC_GetMjpegParam](#)

VENC_ROI_CFG_S

[Description]

Defines an ROI.

[Syntax]

```
typedef struct hivENC_ROI_CFG_S  
{  
    HI_U32   u32Index;  
    HI_BOOL  bEnable;  
    HI_BOOL  bAbsQp;  
    HI_S32   s32Qp;  
    RECT_S   stRect;  
}VENC_ROI_CFG_S;
```

[Member]



Member	Description
u32Index	Index of an ROI. The system supports indexes ranging from 0 to 7.
bEnable	Whether to enable this ROI
bAbsQp	QP mode of an ROI <ul style="list-style-type: none">• HI_FALSE: relative QP• HI_TRUE: absolute QP
s32Qp	QP value. When the QP mode is set to HI_FALSE , this member indicates the QP offset, and the value range is [−51, +51]. When the QP mode is set to HI_TRUE , this member indicates the macroblock QP value, and the value range is [0, 51].
stRect	ROI The value of s32X , s32Y , u32Width , or u32Height must be 16-pixel-aligned. The start coordinates of an ROI and the ROI must both be within the picture range.

[Note]

None

[See Also]

- [HI_MPI_VENC_SetRoiCfg](#)
- [HI_MPI_VENC_GetRoiCfg](#)

VENC_ROIBG_FRAME_RATE_S

[Description]

Defines the frame rate of a non-ROI.

[Syntax]

```
typedef struct hiVENC_ROIBG_FRAME_RATE_S
{
    HI_S32 s32SrcFrmRate;
    HI_S32 s32DstFrmRate;
}VENC_ROIBG_FRAME_RATE_S;
```

[Member]

Member	Description
s32SrcFrmRate	Source frame rate of a non-ROI
s32DstFrmRate	Target frame rate of a non-ROI



[Note]

None

[See Also]

- [HI_MPI_VENC_SetRoiBgFrameRate](#)
- [HI_MPI_VENC_GetRoiBgFrameRate](#)

VENC_PARAM_REF_S

[Description]

Defines the advanced frame skipping reference parameters for H.264/H.265 encoding.

[Syntax]

```
typedef struct hivenc_param_ref_s
{
    HI_U32    u32Base;
    HI_U32    u32Enhance;
    HI_BOOL   bEnablePred;
}VENC_PARAM_REF_S;
```

[Member]

Member	Description
u32Base	Base layer period Value range: (0, +∞)
u32Enhance	Enhance layer period Value range: [0, 255]
bEnablePred	Whether some frames at the base layer are referenced by other frames at the base layer. When bEnablePred is HI_FALSE , all frames at the base layer reference IDR frames.

[Note]

None

[See Also]

None

VENC_PARAM_REF_EX_S

[Description]

Defines the advanced frame skipping reference parameters for an H.264/H.265 VENC channel with virtual I frames.

[Syntax]



```
typedef struct hivENC_PARAM_REF_EX_S
{
    HI_U32    u32Base;
    HI_U32    u32Enhance;
    HI_BOOL   bEnablePred;
    HI_BOOL   bVirtualIEnable;
    HI_U32    u32VirtualIInterval;
    HI_S32    s32VirtualIQpDelta;
}VENC_PARAM_REF_EX_S;
```

[Member]

Member	Description
u32Base	Base layer period Value range: [0, +∞)
u32Enhance	Enhance layer period Value range: [0, 255]
bEnablePred	When the virtual I frames are enabled, bEnablePred must be HI_TRUE .
bVirtualIEnable	Whether to inset the virtual I frames. The virtual I frame is a P frame referring to the IDR frame. It is used to revolve the error code resuming and random playback issues.
u32VirtualIInterval	Virtual I frame interval. Value range: [1, +∞)
s32VirtualIQpDelta	QpDelta of the virtual I frames relative to the common P frames. A positive number indicates that the QP value of the virtual I frame is reduced. Value range: [-10, +10]

[Note]

None

[See Also]

None

VENC_RC_ATTR_S

[Description]

Defines RC attributes for controlling the bit rate of a VENC channel.

[Syntax]

```
typedef struct hivENC_RC_ATTR_S
```



```
{  
    VENC_RC_MODE_E      enRcMode;  
    union  
    {  
        VENC_ATTR_H264_CBR_S   stAttrH264Cbr;  
        VENC_ATTR_H264_VBR_S   stAttrH264Vbr;  
        VENC_ATTR_H264_FIXQP_S stAttrH264FixQp;  
        VENC_ATTR_H264_AVBR_S  stAttrH264AVbr;  
        VENC_ATTR_H264_ABR_S   stAttrH264Abr;  
        VENC_ATTR_MPEG4_CBR_S  stAttrMpeg4Cbr;  
        VENC_ATTR_MPEG4_FIXQP_S stAttrMpeg4FixQp;  
        VENC_ATTR_MPEG4_VBR_S  stAttrMpeg4Vbr;  
        VENC_ATTR_MJPEG_CBR_S  stAttrMjpegeCbr;  
        VENC_ATTR_MJPEG_FIXQP_S stAttrMjpegeFixQp;  
        VENC_ATTR_MJPEG_VBR_S  stAttrMjpegeVbr;  
        VENC_ATTR_H265_CBR_S   stAttrH265Cbr;  
        VENC_ATTR_H265_VBR_S   stAttrH265Vbr;  
        VENC_ATTR_H265_AVBR_S  stAttrH265AVbr;  
        VENC_ATTR_H265_FIXQP_S stAttrH265FixQp;  
    };  
    HI_VOID*      pRcAttr ;  
}VENC_RC_ATTR_S;
```

[Member]

Member	Description
enRcMode	RC mode
stAttrH264Cbr	CBR mode attribute of an H.264 channel
stAttrH264Vbr	VBR mode attribute of an H.264 channel
stAttrH264FixQp	FixQp mode attribute of an H.264 channel
stAttrH264AVbr	AVBR mode attribute of an H.264 channel
stAttrH264Abr	Average bit rate (ABR) mode attribute of an H.264 channel (not supported)
stAttrMpeg4Vbr	VBR mode attribute of an MPEG4 encoding channel
stAttrMpeg4FixQp	FixQp mode attribute of an MPEG-4 channel
stAttrMpeg4Cbr	CBR mode attribute of an MPEG-4 channel.
stAttrMjpegeFixQp	FixQp mode attribute of an MJPEG channel
stAttrMjpegeCbr	CBR mode attribute of an MJPEG channel
stAttrMjpegeVbr	VBR mode attribute of an MJPEG channel
stAttrH265Cbr	CBR mode attribute of an H.265 encoding channel



Member	Description
stAttrH265Vbr	VBR mode attribute of an H.265 encoding channel
stAttrH265AVbr	AVBR mode attribute of an H.265 encoding channel
stAttrH265FixQp	FixQp mode attribute of an H.265 encoding channel

[Note]

None

[See Also]

[HI_MPI_VENC_CreateChn](#)

VENC_RC_MODE_E

[Description]

Defines the mode of the RC for controlling the bit rate of a VENC channel.

[Syntax]

```
typedef enum hivenc_rc_mode_e
{
    VENC_RC_MODE_H264CBR = 1,
    VENC_RC_MODE_H264VBR,
    VENC_RC_MODE_H264AVBR,
    VENC_RC_MODE_H264ABR,
    VENC_RC_MODE_H264FIXQP,
    VENC_RC_MODE_MJPEGCBR,
    VENC_RC_MODE_MJPEGVBR,
    VENC_RC_MODE_MJPEGABR,
    VENC_RC_MODE_MJPEGFIXQP,
    VENC_RC_MODE_MPEG4CBR,
    VENC_RC_MODE_MPEG4VBR,
    VENC_RC_MODE_MPEG4ABR,
    VENC_RC_MODE_MPEG4FIXQP,
    VENC_RC_MODE_H265CBR,
    VENC_RC_MODE_H265VBR,
    VENC_RC_MODE_H265AVBR,
    VENC_RC_MODE_H265FIXQP,
    VENC_RC_MODE_BUTT,
} VENC_RC_MODE_E;
```

[Member]



Member	Description
VENC_RC_MODE_H264CBR	H264 CBR mode
VENC_RC_MODE_H264VBR	H264 VBR mode
VENC_RC_MODE_H264AVBR	H.264 AVBR mode
VENC_RC_MODE_H264ABR	H264 ABR mode (not supported)
VENC_RC_MODE_H264FIXQP	H264 FixQp mode
VENC_RC_MODE_MJPEGCBR	MJPEG CBR mode
VENC_RC_MODE_MJPEGVBR	MJPEG VBR mode
VENC_RC_MODE_MJPEGABR	MJPEG ABR mode (not supported)
VENC_RC_MODE_MJPEGFIXQP	MJPEG FixQp mode
VENC_RC_MODE_MPEG4CBR	MPEG-4 CBR mode
VENC_RC_MODE_MPEG4VBR	MPEG-4 VBR mode
VENC_RC_MODE_MPEG4ABR	MPEG-4 ABR mode (not supported)
VENC_RC_MODE_MPEG4FIXQP	MPEG-4 FixQp mode
VENC_RC_MODE_H265CBR	H.265 CBR mode
VENC_RC_MODE_H265VBR	H.265 VBR mode
VENC_RC_MODE_H265AVBR	H.265 AVBR mode
VENC_RC_MODE_H265FIXQP	H.265 FixQp mode

[Note]

None

[See Also]

[HI_MPI_VENC_CreateChn](#)

VENC_ATTR_H264_CBR_S

[Description]

Defines the CBR attribute of an H.264 channel.

[Syntax]

```
typedef struct hivENC_ATTR_H264_CBR_S
{
    HI_U32      u32Gop;
    HI_U32      u32StatTime;
    HI_U32      u32SrcFrmRate;
    HI_F32     fr32DstFrmRate;
```



```
    HI_U32      u32BitRate;
    HI_U32      u32FluctuateLevel;
} VENC_ATTR_H264_CBR_S;
```

[Member]

Member	Description
u32Gop	H264 GOP value Value range: [1, 65536]
u32StatTime	CBR statistics time, in second Value range: [1, 60]
u32SrcFrmRate	VI frame rate, in frame/s Value range: [1, 240]
fr32DstFrmRate	Output frame rate of the encoder, in frame/s Value range: (0, u32SrcFrmRate]
u32BitRate	Average bit rate, in kbit/s Value range: [2, 102400]
u32FluctuateLevel	Fluctuation level of the maximum bit rate relative to the average bit rate. u32FluctuateLevel is reserved and not used currently. Value range: [1, 5] The fluctuation level 1 is recommended

[Note]

- **SrcFrmRate** must be set to the actual frame rate at which TimeRef is generated. The RC calculates the actual frame rate and controls the bit rate based on the value of **SrcFrmRate**. If the source of the picture to be encoded is VI, **SrcFrmRate** is set to the actual output frame rate of the VI because TimeRef is generated when the VI outputs frames. If the VI and VPSS are in online mode, **SrcFrmRate** is set to the actual output frame rate of the VPSS because TimeRef is generated when the VPSS outputs frames. If the VI and VPSS are in offline mode, the frame rate of the VPSS is controlled by group, and the source frame rate of the VPSS group must be set to the same value as **SrcFrmRate**. In this case, **SrcFrmRate** is set to the actual output frame rate of the VI because TimeRef is generated when the VI outputs frames. If the output frame rate of the VI channel is 30 and the VENC does not control the frame rate, **SrcFrmRate** is set to **30**. If the output frame rate of the VI channel is 30 and the VENC controls the frame rate, **SrcFrmRate** and the input frame rate controlled by the VENC must be set to **30**. If both the VENC and RC control the frame rate, the final frame rate is the smaller value between the frame rates controlled by the VENC and RC. For example, if you set the input frame rate and output frame rate controlled by the VENC to **30** and **6** respectively and set the input frame rate and output frame rate controlled by the RC to **30** and **10** respectively, the final output frame rate is **6**. When you set **DstFrmRate**, its data structure is defined as the fractional type **HI_FR32** that is the same as **HI_U32**. That is, the upper 16 bits indicate the denominator and the lower 16 bits indicate the numerator. To set **DstFrmRate** to an integer, set the upper 16 bits to zeros. For example, if you set **SrcFrmRate** to **25** and **DstFrmRate** to **12**, 12 frames will be selected from 25 frames



for encoding, and the other 13 frames are discarded. If you set **SrcFrmRate** to **25** and **DstFrmRate** to **15/2**, the encoder will encode 15 frames in two seconds.

- **DstFrmRate** can be set as follows:
 - If you want the integral frame rate 25, set **DstFrmRate** to **25**.
 - If you want the fractional frame rate 15/2, set **DstFrmRate** to [**15 + (2 << 16)**].
- When both the VENC and RC are used to control the frame rate, the target frame rate of the VENC is less than that of the RC. For example, if the source frame rate and target frame rate of the VENC are set to 30 fps and 6 fps respectively and the source frame rate and target frame rate of the RC are set to 30 fps and 10 fps respectively, the frame rate cannot be controlled accurately. It is recommended that the target frame rate of the VENC be greater than or equal to that of the RC. In this way, the expected target frame rate is obtained.
- When the VPSS is bound only to the VENC and the automatic mode is used, the required VPSS resources are reduced if the frame rate is controlled by the using the VENC.

[See Also]

[HI_MPI_VENC_CreateChn](#)

VENC_ATTR_H264_VBR_S

[Description]

Defines the VBR attribute of an H.264 channel.

[Syntax]

```
typedef struct hivENC_ATTR_H264_VBR_S
{
    HI_U32      u32Gop;
    HI_U32      u32StatTime;
    HI_U32      u32SrcFrmRate;
    HI_FR32     fr32DstFrmRate;
    HI_U32      u32MaxBitRate;
    HI_U32      u32MaxQp;
    HI_U32      u32MinQp;
} VENC_ATTR_H264_VBR_S;
```

[Member]

Member	Description
u32Gop	H264 GOP value Value range: [1, 65536]
u32StatTime	VBR statistics time, in second Value range: [1, 60]
u32SrcFrmRate	VI frame rate, in frame/s Value range: [1, 240]
fr32DstFrmRate	Output frame rate of the encoder, in frame/s



Member	Description
	Value range: (0, u32SrcFrmRate]
u32MaxBitRate	Maximum output bit rate of the encoder, in frame/s Value range: [2, 102400]
u32MaxQp	Maximum QP supported by the encoder Value range: (u32MinQp, 51]
u32MinQp	Minimum QP supported by the encoder Value range: [0, 51]

[Note]

For details, see the descriptions of **u32SrcFrmRate** and **fr32DstFrmRate** in [VENC_ATTR_H264_CBR_S](#).

[See Also]

[HI_MPI_VENC_CreateChn](#)

VENC_ATTR_H264_AVBR_S

[Description]

Defines the AVBR attribute structure of an H.264 VENC channel.

[Syntax]

```
typedef struct hiVENC_ATTR_H264_AVBR_S
{
    HI_U32      u32Gop;
    HI_U32      u32StatTime;
    HI_U32      u32SrcFrmRate;
    HI_FR32     fr32DstFrmRate;
    HI_U32      u32MaxBitRate;
} VENC_ATTR_H264_AVBR_S;
```

[Member]

Member	Description
u32Gop	H.264 GOP value Value range: [1, 65536]
u32StatTime	AVBR statistical time, in second Value range: [1, 60]
u32SrcFrmRate	VI frame rate, in fps Value range: [1, 240]



Member	Description
fr32DstFrmRate	Output frame rate of the encoder, in fps Value range: (0, u32SrcFrmRate]
u32MaxBitRate	Maximum output bit rate of the encoder, in kbit/s Value range: [2, 102400]

[Note]

For details, see the descriptions of **u32SrcFrmRate** and **fr32DstFrmRate** in [VENC_ATTR_H264_CBR_S](#).

[See Also]

[HI_MPI_VENC_CreateChn](#)

VENC_ATTR_H264_FIXQP_S

[Description]

Defines the FixQP attribute of an H.264 channel.

[Syntax]

```
typedef struct hivENC_ATTR_H264_FIXQP_S
{
    HI_U32      u32Gop;
    HI_U32      u32SrcFrmRate;
    HI_FR32     fr32DstFrmRate;
    HI_U32      u32IQp;
    HI_U32      u32PQp;
} VENC_ATTR_H264_FIXQP_S;
```

[Member]

Member	Description
u32Gop	H264 GOP value Value range: [1, 65536]
u32SrcFrmRate	VI frame rate, in frame/s Value range: [1, 240]
fr32DstFrmRate	Output frame rate of the encoder, in frame/s Value range: (0, u32SrcFrmRate]
u32IQp	QP values of all the macroblocks of I frames Value range: [0, 51]
u32PQp	QP values of all the macroblocks of P frames Value range: [0, 51]



[Note]

For details, see the descriptions of **u32SrcFrmRate** and **fr32DstFrmRate** in [VENC_ATTR_H264_CBR_S](#).

[See Also]

[HI_MPI_VENC_CreateChn](#)

VENC_ATTR_MPEG4_CBR_S

[Description]

Defines the CBR attribute of an MPEG-4 channel.

[Syntax]

```
typedef struct hIVENC_ATTR_MPEG4_CBR_S
{
    HI_U32      u32Gop;
    HI_U32      u32StatTime;
    HI_U32      u32SrcFrmRate;
    HI_FR32     fr32DstFrmRate;
    HI_U32      u32BitRate;
    HI_U32      u32FluctuateLevel;
} VENC_ATTR_MPEG4_CBR_S;
```

[Member]

Member	Description
u32Gop	MPEG-4 GOP value Value range: [1, 65536]
u32SrcFrmRate	VI frame rate, in frame/s Value range: [1, 240]
fr32DstFrmRate	Output frame rate of the encoder, in frame/s Value range: (0, u32SrcFrmRate]
u32StatTime	CBR statistics time, in second Value range: [1, 60]
u32BitRate	Average bit rate, in kbit/s Value range: [2, 102400]
u32FluctuateLevel	Fluctuation level of the maximum bit rate relative to the average bit rate, u32FluctuateLevel is reserved and not used currently. Value range: [1, 5] The fluctuation level 1 is recommended



[Note]

For details, see the descriptions of **u32SrcFrmRate** and **fr32DstFrmRate** in [VENC_ATTR_H264_CBR_S](#).

[See Also]

[HI_MPI_VENC_CreateChn](#)

VENC_ATTR_MPEG4_VBR_S

[Description]

Defines the VBR attribute of an MPEG-4 channel.

[Syntax]

```
typedef struct hivENC_ATTR_MPEG4_VBR_S
{
    HI_U32      u32Gop;
    HI_U32      u32StatTime;
    HI_U32      u32SrcFrmRate;
    HI_FR32     fr32DstFrmRate;
    HI_U32      u32MaxBitRate;
    HI_U32      u32MaxQp;
    HI_U32      u32MinQp;
} VENC_ATTR_MPEG4_VBR_S;
```

[Member]

Member	Description
u32Gop	MPEG-4 GOP value Value range: [1, 65536]
u32StatTime	VBR statistical time, in second Value range: [1, 60]
u32SrcFrmRate	VI frame rate, in frame/s Value range: [1, 240]
fr32DstFrmRate	Output frame rate of the encoder, in frame/s Value range: (0, u32SrcFrmRate]
u32MaxBitRate	Maximum output bit rate of the encoder, in kbit/s Value range: [2, 102400]
u32MaxQp	Maximum QP supported by the encoder Value range: (u32MinQp, 31]
u32MinQp	Minimum QP supported by the encoder Value range: [1, 31]



[Note]

For details, see the descriptions of **u32SrcFrmRate** and **fr32DstFrmRate** in [VENC_ATTR_H264_CBR_S](#).

[See Also]

[HI_MPI_VENC_CreateChn](#)

VENC_ATTR_MPEG4_FIXQP_S

[Description]

Defines the FixQp attribute of an MPEG-4 channel.

[Syntax]

```
typedef struct hivenc_attr_mpeg4_fixqp_s
{
    HI_U32      u32Gop;
    HI_U32      u32SrcFrmRate;
    HI_FR32     fr32DstFrmRate;
    HI_U32      u32IQp;
    HI_U32      u32PQp;
} VENC_ATTR_MPEG4_FIXQP_S;
```

[Member]

Member	Description
u32Gop	MPEG-4 GOP value Value range: [1, 65536]
u32SrcFrmRate	VI frame rate, in frame/s Value range: [1, 240]
fr32DstFrmRate	Output frame rate of the encoder, in frame/s Value range: (0, u32SrcFrmRate]
u32IQp	QP values of all the macroblocks of I frames Value range: [1, 31]
u32PQp	QP values of all the macroblocks of P frames Value range: [1, 31]

[Note]

For details, see the descriptions of **u32SrcFrmRate** and **fr32DstFrmRate** in [VENC_ATTR_H264_CBR_S](#).

[See Also]

[HI_MPI_VENC_CreateChn](#)



VENC_ATTR_MJPEG_FIXQP_S

[Description]

Defines the FixQp attribute of an MJPEG channel.

[Syntax]

```
typedef struct hiVENC_ATTR_MJPEG_FIXQP_S
{
    HI_U32      u32SrcFrmRate;
    HI_FR32     fr32DstFrmRate;
    HI_U32      u32Qfactor;
} VENC_ATTR_MJPEG_FIXQP_S;
```

[Member]

Member	Description
u32SrcFrmRate	VI frame rate, in frame/s Value range: [1, 240]
fr32DstFrmRate	Output frame rate of the encoder, in frame/s Value range: (0, u32ViFrmRate]
u32Qfactor	Qfactor for MJPEG encoding Value range: [1, 99]

[Note]

For details, see the descriptions of **u32SrcFrmRate** and **fr32DstFrmRate** in [VENC_ATTR_H264_CBR_S](#).

[See Also]

[HI_MPI_VENC_CreateChn](#)

VENC_ATTR_MJPEG_CBR_S

[Description]

Defines the CBR attribute of an MJPEG channel.

[Syntax]

```
typedef struct hiVENC_ATTR_MJPEG_CBR_S
{
    HI_U32      u32StatTime;
    HI_U32      u32SrcFrmRate;
    HI_FR32     fr32DstFrmRate;
    HI_U32      u32BitRate;
    HI_U32      u32FluctuateLevel;
} VENC_ATTR_MJPEG_CBR_S;
```



[Member]

Member	Description
u32StatTime	CBR statistics time, in second Value range: [1, 60]
u32SrcFrmRate	Input frame rate of the encoder, in frame/s Value range: [1, 240]
fr32DstFrmRate	Output frame rate of the encoder, in frame/s Value range: (0, u32SrcFrmRate]
u32BitRate	Average bit rate, in kbit/s Value range: [2, 102400]
u32FluctuateLevel	Fluctuation level of the maximum bit rate relative to the average bit rate. u32FluctuateLevel is reserved and not used currently. Value range: [1, 5] The fluctuation level 1 is recommended.

[Note]

For details, see the descriptions of **u32SrcFrmRate** and **fr32DstFrmRate** in [VENC_ATTR_H264_CBR_S](#).

[See Also]

[HI_MPI_VENC_CreateChn](#)

VENC_ATTR_MJPEG_VBR_S

[Description]

Defines the VBR attribute of an MJPEG channel.

[Syntax]

```
typedef struct hiVENC_ATTR_MJPEG_VBR_S
{
    HI_U32      u32StatTime;
    HI_U32      u32SrcFrmRate;
    HI_FR32     fr32DstFrmRate;
    HI_U32      u32MaxBitRate;
    HI_U32      u32MaxQfactor;
    HI_U32      u32MinQfactor;
}VENC_ATTR_MJPEG_VBR_S;
```

[Member]



Member	Description
u32StatTime	VBR statistics time, in second Value range: [1, 60]
u32SrcFrmRate	Input frame rate of the encoder, in frame/s Value range: [1, 240]
fr32DstFrmRate	Output frame rate of the encoder, in frame/s Value range: (0, u32SrcFrmRate]
u32MaxBitRate	Maximum bit rate, in kbit/s Value range: [2, 102400]
u32MaxQfactor	Maximum quantization factor Value range: [1, 99]
u32MinQfactor	Minimum quantization factor Value range: [1, u32MaxQfactor)

[Note]

For details, see the descriptions of **u32SrcFrmRate** and **fr32DstFrmRate** in [VENC_ATTR_H264_CBR_S](#).

[See Also]

[HI_MPI_VENC_CreateChn](#)

VENC_ATTR_H265_CBR_S

[Description]

Defines the CBR attribute of an H.265 VENC channel.

[Syntax]

```
typedef struct hIVENC_ATTR_H265_CBR_S
{
    HI_U32      u32Gop;
    HI_U32      u32StatTime;
    HI_U32      u32SrcFrmRate;
    HI_FR32     fr32DstFrmRate;
    HI_U32      u32BitRate;
    HI_U32      u32FluctuateLevel;
} VENC_ATTR_H265_CBR_S;
```

[Member]



Member	Description
u32Gop	H.265 GOP value Value range: [1, 65536]
u32StatTime	CBR statistics time, in second Value range: [1, 60]
u32SrcFrmRate	VI frame rate, in frame/s Value range: [1, 240]
fr32DstFrmRate	Output frame rate of the encoder, in frame/s Value range: (0, u32SrcFrmRate]
u32BitRate	Average bit rate, in kbit/s Value range: [2, 102400]
u32FluctuateLevel	Fluctuation level of the maximum bit rate relative to the average bit rate. It is reserved for future use. Value range: [0, 5] The fluctuation level 0 is recommended.

[Note]

For details, see the descriptions of **u32SrcFrmRate** and **fr32DstFrmRate** in [VENC_ATTR_H264_CBR_S](#).

[See Also]

[HI_MPI_VENC_CreateChn](#)

VENC_ATTR_H265_VBR_S

[Description]

Defines the VBR attribute of an H.265 VENC channel.

[Syntax]

```
typedef struct hivENC_ATTR_H265_VBR_S
{
    HI_U32      u32Gop;
    HI_U32      u32StatTime;
    HI_U32      u32SrcFrmRate;
    HI_FR32     fr32DstFrmRate;
    HI_U32      u32MaxBitRate;
    HI_U32      u32MaxQp;
    HI_U32      u32MinQp;
} VENC_ATTR_H265_VBR_S;
```

[Member]



Member	Description
u32Gop	H.265 GOP value Value range: [1, 65536]
u32StatTime	VBR statistical time, in second Value range: [1, 60]
u32SrcFrmRate	VI frame rate, in frame/s Value range: [1, 240]
fr32DstFrmRate	Output frame rate of the encoder, in frame/s Value range: (0, u32SrcFrmRate]
u32MaxBitRate	Maximum output bit rate of the encoder, in kbit/s Value range: [2, 102400]
u32MaxQp	Maximum QP supported by the encoder Value range: (u32MinQp, 51]
u32MinQp	Minimum QP supported by the encoder Value range: [0, 51]

[Note]

For details, see the descriptions of **u32SrcFrmRate** and **fr32DstFrmRate** in [VENC_ATTR_H264_CBR_S](#).

[See Also]

[HI_MPI_VENC_CreateChn](#)

VENC_ATTR_H265_AVBR_S

[Description]

Defines the AVBR attribute structure of an H.265 VENC channel.

[Syntax]

```
typedef struct hivENC_ATTR_H265_AVBR_S
{
    HI_U32      u32Gop;
    HI_U32      u32StatTime;
    HI_U32      u32SrcFrmRate;
    HI_FR32     fr32DstFrmRate;
    HI_U32      u32MaxBitRate;
}VENC_ATTR_H265_AVBR_S;
```

[Member]



Member	Description
u32Gop	H.265 GOP value Value range: [1, 65536]
u32StatTime	AVBR statistical time, in second Value range: [1, 60]
u32SrcFrmRate	VI frame rate, in fps Value range: [1, 240]
fr32DstFrmRate	Output frame rate of the encoder, in fps Value range: (0, u32SrcFrmRate]
u32MaxBitRate	Maximum output bit rate of the encoder, in kbit/s Value range: [2, 102400]

[Note]

For details, see the descriptions of **u32SrcFrmRate** and **fr32DstFrmRate** in [VENC_ATTR_H264_CBR_S](#).

[See Also]

[HI_MPI_VENC_CreateChn](#)

VENC_ATTR_H265_FIXQP_S

[Description]

Defines the FixQp attribute of an H.265 VENC channel.

[Syntax]

```
typedef struct hivENC_ATTR_H265_FIXQP_S
{
    HI_U32      u32Gop;
    HI_U32      u32SrcFrmRate;
    HI_FR32     fr32DstFrmRate;
    HI_U32      u32IQp;
    HI_U32      u32PQp;
} VENC_ATTR_H265_FIXQP_S;
```

[Member]

Member	Description
u32Gop	H.265 GOP value Value range: [1, 65536]
u32SrcFrmRate	VI frame rate, in frame/s Value range: [1, 240]



Member	Description
fr32DstFrmRate	Output frame rate of the encoder, in frame/s Value range: (0, u32SrcFrmRate]
u32IQp	QP values of all the macroblocks of I frames Value range: [0, 51]
u32PQp	QP values of all the macroblocks of P frames Value range: [0, 51]

[Note]

For details, see the descriptions of **u32SrcFrmRate** and **fr32DstFrmRate** in [VENC_ATTR_H264_CBR_S](#).

[See Also]

[HI_MPI_VENC_CreateChn](#)

VENC_SUPERFRM_MODE_E

[Description]

Defines the mode of processing jumbo frames when the bit rate is being controlled.

[Syntax]

```
typedef enum hiRC_SUPERFRM_MODE_E
{
    SUPERFRM_NONE=0,
    SUPERFRM_DISCARD,
    SUPERFRM_REENCODE,
    SUPERFRM_BUTT
}VENC_SUPERFRM_MODE_E;
```

[Member]

Member	Description
SUPERFRM_NONE	No special measure is taken.
SUPERFRM_DISCARD	Jumbo frames are discarded.
SUPERFRM_REENCODE	Jumbo frames are re-encoded.

[Note]

None

[See Also]

[HI_MPI_VENC_SetSuperFrameCfg](#)



VENC_PARAM_H264_VBR_S

[Description]

Defines the configurations of the advanced parameters related to the VBR control mode of an H.264 channel.

[Syntax]

```
typedef struct hiVENC_PARAM_H264_VBR_S
{
    HI_S32    s32IPQPDelta;
    HI_S32    s32ChangePos;
    HI_U32    u32MinIprop;
    HI_U32    u32MaxIprop;
    HI_U32    u32MinIQP;
} VENC_PARAM_H264_VBR_S;
```

[Member]

Member	Description
s32IPQPDelta	IP QP variance Value range: [0, 30]
s32ChangePos	Percentage of the bit rate when the QP starts to be adjusted in VBR mode relative to the maximum bit rate Value range: [50, 100]
u32MinIprop	Minimum ratio of the I/P frame bit rates Value range: [1, 100]
u32MaxIprop	Maximum ratio of the I/P frame bit rate Value range: [u32MinIprop, 100]
u32MinIQP	Minimum QP of an I frame for controlling the maximum number of bits of an I frame Value range: [MinQp, MaxQp)

[Note]

None

[See Also]

- [HI_MPI_VENC_GetRcParam](#)
- [HI_MPI_VENC_SetRcParam](#)

VENC_PARAM_H264_AVBR_S

[Description]



Defines the configuration of the advanced parameters related to the AVBR control mode of an H.264 VENC channel.

[Syntax]

```
typedef struct hiVENC_PARAM_H264_AVBR_S
{
    HI_S32    s32IPQDelta;
    HI_S32    s32ChangePos;
    HI_U32    u32MinIprop;
    HI_U32    u32MaxIprop;
    HI_U32    u32MaxQp;
    HI_U32    u32MinQp;
    HI_U32    u32MaxIQp;
    HI_U32    u32MinIQp;
    HI_S32    s32MaxReEncodeTimes;
    HI_S32    s32MinStillPercent;
    HI_U32    u32MaxStillQP;
    HI_U32    u32MinStillPSNR;
}VENC_PARAM_H264_AVBR_S;
```

[Member]

Member	Description
s32IPQDelta	IP QP variance Value range: [0, 30]
s32ChangePos	Ratio of the bit rate when the QP starts to be adjusted in AVBR mode relative to the maximum bit rate Value range: [50, 100] Default value: 90
u32MinIprop	Minimum ratio of the I frame bit rate to the P frame bit rate Value range: [1, 100] Default value: 1
u32MaxIprop	Maximum ratio of the I frame bit rate to the P frame bit rate Value range: [u32MinIprop, 100] Default value: 100
u32MaxQp	Maximum QP value of the P frame and B frame Value range: (MinQp, 51) Default value: 51
u32MinQp	Minimum QP value of the P frame and B frame Value range: [0, 51] Default value: 16



Member	Description
u32MaxIQp	Maximum QP value of an I frame Value range: (MinIQp, 51] Default value: 51
u32MinIQP	Minimum QP value of an I frame Value range: [0, 51] Default value: 16
s32MinStillPercent	Minimum percentage of the target bit rate in the static scenario. If this member is set to 100 , the target bit rate will not be proactively reduced in AVBR mode when the scenario is judged to be a static one, and the effect of the AVBR mode is consistent with that of the VBR mode. Value range: [20, 100] Default value: 25
u32MaxStillQP	Maximum QP value of the I frame in the static scenario Value range: [MinIQp, MaxIQp] Default value: 35
u32MinStillPSNR	Invalid member currently
s32MaxReEncodeTimes	Number of times that each frame is re-encoded. The value 0 indicates that the frame is not re-encoded. Value range: [0, 3] Default value: 2

[Note]

None

[See Also]

- [HI_MPI_VENC_GetRcParam](#)
- [HI_MPI_VENC_SetRcParam](#)

VENC_PARAM_H264_CBR_S

[Description]

Defines the configurations of the advanced parameters related to the new CBR control mode of an H.264 channel.

[Syntax]

```
typedef struct hiVENC_PARAM_H264_CBR_S
{
    HI_U32    u32MinIprop;
    HI_U32    u32MaxIprop;
    HI_U32    u32MaxQp;
```



```
    HI_U32  u32MinQp;
    HI_S32  s32IPQPDelta;
    HI_S32  s32QualityLevel;
    HI_S32  s32MaxReEncodeTimes;
    HI_U32  u32MinIQp;
    HI_U32  u32MaxIQp;
}VENC_PARAM_H264_CBR_S;
```

[Member]

Member	Description
u32MinIprop	Minimum ratio of I frames to P frames Value range: [1, 100]
u32MaxIprop	Maximum ratio of I frames to P frames Value range: [u32MinIprop, 100]
u32MaxQp	Maximum frame QP value for controlling the picture quality Value range: (u32MinQp, 51]
u32MinQp	Minimum frame QP value for controlling bit rate fluctuation Value range: [0, u32MaxQp)
s32IPQPDelta	IP QP variance Value range: [-10, 30]
s32QualityLevel	Quality level A smaller value indicates better quality. Value range: [1, 5]
s32MaxReEncodeTimes	Number of times that each frame is re-encoded The value 0 indicates that the frame is not re-encoded. Value range: [0, 3]
u32MinIQp	Minimum QP of an I frame for controlling the maximum number of bits of an I frame Value range: [0, 51]
u32MaxIQp	Maximum QP of an I frame for controlling the maximum number of bits of an I frame Value range: (u32MinIQp, 51] or -1

[Note]

None

[See Also]

- [HI_MPI_VENC_GetRcParam](#)
- [HI_MPI_VENC_SetRcParam](#)



VENC_PARAM_MJPEG_CBR_S

[Description]

Defines the configurations of the advanced parameters related to the CBR control mode of an MJPEG channel.

[Syntax]

```
typedef struct hivENC_PARAM_MJPEG_CBR_S
{
    HI_U32 u32MaxQfactor;
    HI_U32 u32MinQfactor;
    HI_U32 u32RQRatio[RC_RQRATIO_SIZE];
}VENC_PARAM_MJPEG_CBR_S;
```

[Member]

Member	Description
u32MaxQfactor	Maximum frame Qfactor value for controlling the picture quality Value range: (u32MinQfactor, 99]
u32MinQfactor	Minimum frame Qfactor for controlling the picture quality Value range: [1, u32MaxQfactor)
u32RQRatio	R-Q ratio. u32RQRatio is reserved and not used currently. RC_RQRATIO_SIZE is defined to 8.

[Note]

None

[See Also]

- [HI_MPI_VENC_GetRcParam](#)
- [HI_MPI_VENC_SetRcParam](#)

VENC_PARAM_MJPEG_VBR_S

[Description]

Defines the configurations of the advanced parameters related to the VBR control mode of an MJPEG channel.

[Syntax]

```
typedef struct hivENC_PARAM_MJPEG_VBR_S
{
    HI_S32 s32DeltaQfactor;
    HI_S32 s32ChangePos;
}VENC_PARAM_MJPEG_VBR_S;
```



[Member]

Member	Description
s32DeltaQfactor	Maximum Qfactor variance value between frames when the Qfactor value is adjusted in VBR mode Value range: [0, 30]
s32ChangePos	Percentage of the bit rate when the Qfactor starts to be adjusted in VBR mode relative to the maximum bit rate Value range: [50, 100]

[Note]

None

[See Also]

- [HI_MPI_VENC_GetRcParam](#)
- [HI_MPI_VENC_SetRcParam](#)

VENC_PARAM_MPEG4_CBR_S

[Description]

Defines the configurations of the advanced parameters related to the CBR control mode of an MPEG-4 channel.

[Syntax]

```
typedef struct hivenc_param_mpeg4_cbr_s
{
    HI_U32    u32MinIprop;
    HI_U32    u32MaxIprop;
    HI_U32    u32MaxQp;
    HI_U32    u32MinQp;
    HI_U32    u32MaxPPDeltaQp;
    HI_U32    u32MaxIPDeltaQp;
    HI_S32    s32IPQPDelta;
    HI_U32    u32RQRatio[RC_RQRATIO_SIZE];
}VENC_PARAM_MPEG4_CBR_S;
```

[Member]

Member	Description
u32MaxQp	Maximum frame QP value for controlling the picture quality Value range: (u32MinQp, 31]
u32MinQp	Minimum frame QP value for controlling bit rate fluctuation Value range: [1, u32MaxQp)



Member	Description
u32MaxPPDeltaQp	Maximum QP value between P frames Value range: [0, 30]
u32MaxIPDeltaQp	Maximum QP value between I frames and P frames Value range: [0, 30]
u32MinIprop	Minimum ratio of the I/P frame bit rate Value range: [1, 100]
u32MaxIprop	Maximum ratio of the I/P bit rate Value range: [u32MinIprop, 100]
s32IPQPDelta	IP QP variance Value range: [-10, 30]
u32RQRatio	R-Q ratio. u32RQRatio is reserved and not used currently. Value range: [0, 100]

[Note]

None

[See Also]

- [HI_MPI_VENC_GetRcParam](#)
- [HI_MPI_VENC_SetRcParam](#)

VENC_PARAM_MPEG4_VBR_S

[Description]

Defines the configurations of the advanced parameters related to the VBR control mode of an MPEG-4 channel.

[Syntax]

```
typedef struct hivENC_PARAM_MPEG4_VBR_S
{
    HI_S32 s32IPQPDelta;
    HI_S32 s32ChangePos;
    HI_U32 u32MinIprop;
    HI_U32 u32MaxIprop;
}VENC_PARAM_MPEG4_VBR_S;
```

[Member]

Member	Description
s32IPQPDelta	IP QP variance Value range: [0, 30]



Member	Description
s32ChangePos	Percentage of the bit rate when the QP starts to be adjusted in VBR mode relative to the maximum bit rate Value range: [50, 100]
u32MinIprop	Minimum ratio of I frames to P frames Value range: [1, 100]
u32MaxIprop	Maximum ratio of I frames to P frames Value range: [u32MinIprop, 100]

[Note]

None

[See Also]

- [HI_MPI_VENC_GetRcParam](#)
- [HI_MPI_VENC_SetRcParam](#)

VENC_PARAM_H265_VBR_S

[Description]

Defines the configurations of the advanced parameters related to the VBR control mode of an H.265 encoding channel.

[Syntax]

```
typedef struct hiVENC_PARAM_H265_VBR_S
{
    HI_S32    s32IPQPDelta;
    HI_S32    s32ChangePos;
    HI_U32   u32MinIprop;
    HI_U32   u32MaxIprop;
    HI_U32   u32MinIQP;
} VENC_PARAM_H265_VBR_S;
```

[Member]

Member	Description
s32IPQPDelta	IP QP variance Value range: [0, 30]
s32ChangePos	Percentage of the bit rate when the QP starts to be adjusted in VBR mode relative to the maximum bit rate Value range: [50, 100]
u32MinIprop	Minimum ratio of the I frame bit rate to the P frame bit rate Value range: [1, 100]



Member	Description
u32MaxIprop	Maximum ratio of the I frame bit rate to the P frame bit rate Value range: [u32MinIprop, 100]
u32MinIQP	Minimum QP of an I frame for controlling the maximum number of bits of an I frame Value range: [MinQp, MaxQp)

[Note]

None

[See Also]

- [HI_MPI_VENC_GetRcParam](#)
- [HI_MPI_VENC_SetRcParam](#)

VENC_PARAM_H265_AVBR_S

[Description]

Defines the configurations of the advanced parameters related to the AVBR control mode of an H.265 VENC channel.

[Syntax]

```
typedef struct hiVENC_PARAM_H265_AVBR_S
{
    HI_S32    s32IPQPDelta;
    HI_S32    s32ChangePos;
    HI_U32    u32MinIprop;
    HI_U32    u32MaxIprop;
    HI_U32    u32MaxQp;
    HI_U32    u32MinQp;
    HI_U32    u32MaxIQp;
    HI_U32    u32MinIQp;
    HI_S32    s32MaxReEncodeTimes;
    HI_S32    s32MinStillPercent;
    HI_U32    u32MaxStillQP;
    HI_U32    u32MinStillPSNR;
} VENC_PARAM_H265_AVBR_S;
```

[Member]

Member	Description
s32IPQPDelta	IP QP variance Value range: [0, 30]



Member	Description
s32ChangePos	Ratio of the bit rate when the QP starts to be adjusted in AVBR mode relative to the maximum bit rate Value range: [50, 100] Default value: 90
u32MinIprop	Minimum ratio of the I frame bit rate to the P frame bit rate Value range: [1, 100] Default value: 1
u32MaxIprop	Maximum ratio of the I frame bit rate to the P frame bit rate Value range: [u32MinIprop, 100] Default value: 100
u32MaxQp	Maximum QP value of the P frame and B frame Value range: (MinQp, 51] Default value: 51
u32MinQp	Minimum QP value of the P frame and B frame Value range: [0, 51] Default value: 16
u32MaxIQp	Maximum QP value of an I frame Value range: (MinIQp, 51] Default value: 51
u32MinIQP	Minimum QP value of an I frame Value range: [0, 51] Default value: 16
s32MinStillPercent	Minimum percentage of the target bit rate in the static scenario. If this member is set to 100 , the target bit rate will not be proactively reduced in AVBR mode when the scenario is judged to be a static one, and the effect of the AVBR mode is consistent with that of the VBR mode. Value range: [20, 100] Default value: 25
u32MaxStillQP	Maximum QP value of the I frame in the static scenario Value range: [MinIQp, MaxIQp] Default value: 35
u32MinStillPSNR	Invalid member currently
s32MaxReEncodeTimes	Number of times that each frame is re-encoded. The value 0 indicates that the frame is not re-encoded. Value range: [0, 3] Default value: 2



[Note]

None

[See Also]

- [HI_MPI_VENC_GetRcParam](#)
- [HI_MPI_VENC_SetRcParam](#)

VENC_PARAM_H265_CBR_S

[Description]

Defines the configurations of the advanced parameters related to the CBR control mode of an H.265 encoding channel.

[Syntax]

```
typedef struct hivenc_param_h265_cbr_s
{
    HI_U32    u32MinIprop;
    HI_U32    u32MaxIprop;
    HI_U32    u32MaxQp;
    HI_U32    u32MinQp;
    HI_S32    s32IPQPDelta;
    HI_S32    s32QualityLevel;
    HI_S32    s32MaxReEncodeTimes;
    HI_U32    u32MinIQp;
    HI_U32    u32MaxIQp;
}VENC_PARAM_H265_CBR_S;
```

[Member]

Member	Description
u32MinIprop	Minimum ratio of I frames to P frames Value range: [1, 100]
u32MaxIprop	Maximum ratio of I frames to P frames Value range: [u32MinIprop, 100]
u32MaxQp	Maximum frame QP value for controlling the picture quality Value range: (u32MinQp, 51]
u32MinQp	Minimum frame QP value for controlling bit rate fluctuation Value range: [0, u32MaxQp)
s32IPQPDelta	IP QP variance Value range: [-10, 30]
s32QualityLevel	Quality level. A smaller value indicates better quality Value range: [1, 5]



Member	Description
s32MaxReEncodeTimes	Number of times that each frame is re-encoded. The value 0 indicates that the frame is not re-encoded. Value range: [0, 3]
u32MinIqp	Minimum QP of an I frame for controlling the maximum number of bits of an I frame Value range: [0, 51]
u32MaxIqp	Maximum QP of an I frame for controlling the maximum number of bits of an I frame Value range: (u32MinIqp, 51] or -1

[Note]

None

[See Also]

- [HI_MPI_VENC_GetRcParam](#)
- [HI_MPI_VENC_SetRcParam](#)

VENC_RC_PARAM_S

[Description]

Defines the advanced parameters for controlling the bit rate of a VENC channel.

[Syntax]

```
typedef struct hiVENC_RC_PARAM_S
{
    HI_U32 u32ThrdI[RC_TEXTURE_THR_SIZE];
    HI_U32 u32ThrdP[RC_TEXTURE_THR_SIZE];
    HI_U32 u32RowQpDelta;
    HI_S32 s32FirstFrameStartQp;
    union
    {
        VENC_PARAM_H264_CBR_S stParamH264Cbr;
        VENC_PARAM_H264_VBR_S stParamH264VBR;
        VENC_PARAM_H264_AVBR_S stParamH264AVbr;
        VENC_PARAM_MJPEG_CBR_S stParamMjpegCbr;
        VENC_PARAM_MJPEG_VBR_S stParamMjpegVbr;
        VENC_PARAM_MPEG4_CBR_S stParamMpeg4Cbr;
        VENC_PARAM_MPEG4_VBR_S stParamMpeg4Vbr;
        VENC_PARAM_H265_CBR_S stParamH265Cbr;
        VENC_PARAM_H265_VBR_S stParamH265Vbr;
        VENC_PARAM_H265_AVBR_S stParamH265AVbr;
    };
}
```



```
    HI_VOID* pRcParam;  
}VENC_RC_PARAM_S;
```

[Member]

Member	Description
u32ThrdI	Mad threshold for controlling the macroblock-level bit rate of I frames Value range: [0, 255] The value of RC_TEXTURE_THR_SIZE is 12.
u32ThrdP	Mad threshold for controlling the macroblock-level bit rate of P frames Value range: [0, 255] The value of RC_TEXTURE_THR_SIZE is 12.
u32RowQpDelta	Fluctuation amplitude of the start QP value of each macroblock row relative to the start QP value of the frame during macroblock-level bit rate control Value range: [0, 10]
s32FirstFrameStart Qp	Start QP value of the first frame, valid in CBR, VBR, or AVBR mode Value range: <ul style="list-style-type: none">• [MinIQP, MaxIQP] and -1 for the CBR mode• [MinIQP, MaxQp] and -1 for the VBR mode• [MinIQP, MaxIqp] and -1 for the AVBR mode If s32FirstFrameStartQp is -1, the start QP value of the first frame is internally calculated by the encoder. If s32FirstFrameStartQp is set to any other valid value, the user specifies this valid value as the start QP value of the first frame.
stParamH264Cbr	Advanced parameter of the CBR control mode for an H.264 channel
stParamH264Vbr	Advanced parameter of the VBR control mode for an H.264 channel
stParamH264AVbr	Advanced parameter of the AVBR control mode of an H.264 channel
stParamMjpegCbr	Advanced parameter of the CBR control mode for an MJPEG channel
stParamMjpegVbr	Advanced parameter of the VBR control mode of an MJPEG channel
stParamMpeg4Cbr	Advanced parameter of the CBR control mode for an MPEG-4 channel
stParamMpeg4Vbr	Advanced parameter of the VBR control mode of an MPEG-4 channel
stParamH265Cbr	Advanced parameter of the CBR control mode of an H.265 channel
stParamH265Vbr	Advanced parameter of the VBR control mode of an H.265 channel
stParamH265AVbr	Advanced parameter of the AVBR control mode of an H.265 channel



Member	Description
pRcParam	Reserved

[Note]

None

[See Also]

- [HI_MPI_VENC_SetRcParam](#)
- [HI_MPI_VENC_GetRcParam](#)

VENC_CROP_CFG_S

[Description]

Defines the crop parameters of a VENC channel.

[Syntax]

```
typedef struct hiVENC_CROP_CFG_S
{
    HI_BOOL bEnable;      /* Crop region enable */
    RECT_S stRect;        /* Crop region, note: s32X must be multi of 16 */
}VENC_CROP_CFG_S;
```

[Member]

Member	Description
bEnable	Crop enable Value range: [HI_FALSE, HI_TRUE] HI_TRUE: enabled HI_FALSE: disabled
stRect	Crop region stRect.s32X : The parameter value must be 16-pixel-aligned. stRect.s32Y : The parameter value must be 2-pixel-aligned. stRect.u32Width and s32Rect.u32Height meet the width and height requirements of a VENC channel.

[Note]

None

[See Also]

- [HI_MPI_VENC_SetCrop](#)
- [HI_MPI_VENC_GetCrop](#)



VENC_FRAME_RATE_S

[Description]

Defines the parameters for controlling the frame rate of a VENC channel.

[Syntax]

```
typedef struct hiVENC_FRAME_RATE_S
{
    HI_S32 s32SrcFrmRate;
    HI_S32 s32DstFrmRate;
} VENC_FRAME_RATE_S;
```

[Member]

Member	Description
s32SrcFrmRate	Input frame rate of a VENC channel, in frame/s Value range: -1 or [1, 240]
s32DstFrmRate	Output frame rate of a VENC channel, in frame/s Value range: -1 or (0, s32SrcFrmRate]

[Note]

None

[See Also]

- [HI_MPI_VENC_SetFrameRate](#)
- [HI_MPI_VENC_GetFrameRate](#)

VENC_COLOR2GREY_S

[Description]

Defines the color-to-gray function of a VENC channel.

[Syntax]

```
typedef struct hiVENC_COLOR2GREY_S
{
    HI_BOOL bColor2Grey;           /* Whether to enable Color2Grey.*/
} VENC_COLOR2GREY_S;
```

[Member]

Member	Description
bColor2Grey	Color-to-gray enable for a VENC channel



[Note]

None

[See Also]

- [HI_MPI_VENC_SetColor2Grey](#)
- [HI_MPI_VENC_GetColor2Grey](#)

VENC_JPEG_SNAP_MODE_E

[Description]

Defines the snapshot mode of a JPEG VENC channel.

[Syntax]

```
typedef enum hiVENC_JPEG_SNAP_MODE_E
{
    JPEG_SNAP_ALL    = 0,
    JPEG_SNAP_FLASH  = 1,
    JPEG_SNAP_BUTT,
} VENC_JPEG_SNAP_MODE_E;
```

[Member]

Member	Description
JPEG_SNAP_ALL	Snapshot mode in which all captured pictures are encoded. This mode is the default snapshot mode of a JPEG channel.
JPEG_SNAP_FLASH	Snapshot mode in which only the pictures are captured when the camera flash is on are encoded. This mode is available only when the front-end camera flash works.

[Note]

None

[See Also]

- [HI_MPI_VENC_SetJpegSnapMode](#)
- [HI_MPI_VENC_GetJpegSnapMode](#)

VENC_RECV_PIC_PARAM_S

[Description]

Defines the number of received frames to be encoded.

[Syntax]

```
typedef struct hiVENC_RECV_PIC_PARAM_S
{
    HI_S32 s32RecvPicNum;
```



```
} VENC_RECV_PIC_PARAM_S;
```

[Member]

Member	Description
s32RecvPicNum	Number of received frames to be encoded by the encoding channel

[Note]

None

[See Also]

[HI_MPI_VENC_StartRecvPicEx](#)

H264E_IDR_PIC_ID_MODE_E

[Description]

Defines the mode of setting the IDR frame or I frame.

[Syntax]

```
typedef enum hiH264E_IDR_PIC_ID_MODE_E
{
    H264E_IDR_PIC_ID_MODE_AUTO = 0,
    H264E_IDR_PIC_ID_MODE_USR,
    H264E_IDR_PIC_ID_MODE_BUTT,
} H264E_IDR_PIC_ID_MODE_E;
```

[Member]

Member	Description
H264E_IDR_PIC_ID_MODE_AUTO	Automatic mode
H264E_IDR_PIC_ID_MODE_USR	User-defined mode

[Note]

None

[See Also]

[HI_MPI_VENC_SetH264IdrPicId](#)

VENC_H264_IDRPICID_CFG_S

[Description]

Defines the parameters for setting the IDR frame or I frame.

[Syntax]



```
typedef struct hiVENC_H264_IDRPICID_CFG_S
{
    H264E_IDR_PIC_ID_MODE_E enH264eIdrPicIdMode;
    HI_U32 u32H264eIdrPicId;
} VENC_H264_IDRPICID_CFG_S;
```

[Member]

Member	Description
enH264eIdrPicIdMode	Mode of setting the idr_pic_id
u32H264IdrPicId	idr_pic_id value Value range: [0, 65535]

[Note]

None

[See Also]

[HI_MPI_VENC_SetH264IdrPicId](#)

VENC_PARAM_H265_SLICE_SPLIT_S

[Description]

Defines the slice split attribute of an H.265 VENC channel.

[Syntax]

```
typedef struct hiVENC_PARAM_H265_SLICE_SPLIT_S
{
    HI_BOOL bSplitEnable;
    HI_U32 u32SplitMode;
    HI_U32 u32SliceSize;
    HI_U32 loop_filter_across_slices_enabled_flag;
} VENC_PARAM_H265_SLICE_SPLIT_S;
```

[Member]

Member	Description
bSplitEnable	Slice split enable
u32SplitMode	Slice split mode 0: by byte 1: by LCU line The value that is greater than or equal to 2 is not supported.
u32SliceSize	Number of bytes in each slice when u32SplitMode is 0 Minimum value: 128



Member	Description
	Maximum value: min((0xFFFF), u32PicSize/2) The picture size is calculated as follows: u32PicSize = picwidth x picheight x 3/2 Number of LCU lines occupied by each slice when u32SplitMode is 1 Minimum value: 1 Maximum value: (Picture height + 63)/64
loop_filter_across_slices_enabled_flag	Whether filtering is performed on the left and top borders of slices

[Note]

None

[See Also]

- [HI_MPI_VENC_SetH265SliceSplit](#)
- [HI_MPI_VENC_GetH265SliceSplit](#)

VENC_PARAM_H265_PU_S

[Description]

Defines the PU attribute of an H.265 VENC channel.

[Syntax]

```
typedef struct hiVENC_PARAM_H265_PU_S
{
    HI_BOOL bPu32x32En;
    HI_BOOL bPu16x16En;
    HI_BOOL bPu8x8En;
    HI_BOOL bPu4x4En;
    HI_U32 constrained_intra_pred_flag;
    HI_U32 strong_intra_smoothing_enabled_flag;
    HI_U32 pcm_enabled_flag;
    HI_U32 pcm_loop_filter_disabled_flag;
    HI_U32 u32MaxNumMergeCand;
} VENC_PARAM_H265_PU_S;
```

[Member]



Member	Description
bPu32x32En	PU32x32 enable Value range: 0 or 1 0: disabled 1: enabled (default)
bPu16x16En	PU16x16 enable Value range: 0 or 1 0: disabled 1: enabled (default)
bPu8x8En	PU8x8 enable Value range: 0 or 1 0: disabled 1: enabled (default)
bPu4x4En	PU4x4 enable Value range: 0 or 1 0: disabled 1: enabled (default)
constrained_intra_pred_flag	Value range: 0 (default) or 1
strong_intra_smoothing_enabled_flag	Value range: 0 or 1 (default)
pcm_enabled_flag	Value range: 0 (default) or 1 The default value varies according to the chip type.
pcm_loop_filter_disabled_flag	Value range: 0 or 1 (default) The default value varies according to the chip type. This member is valid only when pcm_enabled_flag is 1. pcm_loop_filter_disabled_flag must be 1 for the Hi3516A.
u32MaxNumMergeCand	Value range: [2, 5] Default value: 2 The default value varies according to the chip type. u32MaxNumMergeCand must be 2 for the Hi3516A.

[Note]

None

[See Also]

- [HI_MPI_VENC_SetH265PredUnit](#)
- [HI_MPI_VENC_GetH265PredUnit](#)



VENC_PARAM_H265_TRANS_S

[Description]

Defines the transformation and quantization attribute of an H.265 VENC channel.

[Syntax]

```
typedef struct hiVENC_PARAM_H265_TRANS_S
{
    HI_U32 transquant_bypass_enabled_flag;
    HI_U32 transform_skip_enabled_flag;
    HI_S32 cb_qp_offset;
    HI_S32 cr_qp_offset;
} VENC_PARAM_H265_TRANS_S;
```

[Member]

Member	Description
transquant_bypass_enabled_flag	Value range: 0 (default) or 1 The default value varies according to the chip type. The value must be 0 for the Hi3516A.
transform_skip_enabled_flag	Value range: 0 (default) or 1 The default value varies according to the chip type. The value must be 0 for the Hi3516A.
cb_qp_offset	Value range: [-12, +12] Default value: 0 The default value varies according to the chip type.
cr_qp_offset	Value range: [-12, +12] Default value: 0 The default value varies according to the chip type.

[Note]

None

[See Also]

- [HI_MPI_VENC_SetH265Trans](#)
- [HI_MPI_VENC_GetH265Trans](#)

VENC_PARAM_H265_ENTROPY_S

[Description]

Defines the entropy encoding attribute of an H.265 VENC channel.

[Syntax]



```
typedef struct hiVENC_PARAM_H265_ENTROPY_S
{
    HI_U32 cabac_init_flag;
} VENC_PARAM_H265_ENTROPY_S;
```

[Member]

Member	Description
cabac_init_flag	Value range: 0 or 1 (default)

[Note]

None

[See Also]

- [HI_MPI_VENC_SetH265Entropy](#)
- [HI_MPI_VENC_GetH265Entropy](#)

VENC_PARAM_H265_DBLK_S

[Description]

Defines the deblocking attribute of an H.265 VENC channel.

[Syntax]

```
typedef struct hiVENC_PARAM_H265_DBLK_S
{
    HI_U32 slice_deblocking_filter_disabled_flag;
    HI_S32 slice_beta_offset_div2;
    HI_S32 slice_tc_offset_div2;
} VENC_PARAM_H265_DBLK_S;
```

[Member]

Member	Description
slice_deblocking_filter_disabled_flag	Value range: 0 (default) or 1
slice_tc_offset_div2	Value range: [-12, +12] Default value: 0
slice_beta_offset_div2	Value range: [-12, +12] Default value: 0

[Note]

None

[See Also]



- [HI_MPI_VENC_SetH265Dbblk](#)
- [HI_MPI_VENC_GetH265Dbblk](#)

VENC_PARAM_H265_SAO_S

[Description]

Defines the SAO attribute of an H.265 VENC channel.

[Syntax]

```
typedef struct hiVENC_PARAM_H265_SAO_S
{
    HI_U32 slice_sao_luma_flag;
    HI_U32 slice_sao_chroma_flag;
} VENC_PARAM_H265_SAO_S;
```

[Member]

Member	Description
slice_sao_luma_flag	Value range: 0 or 1 (default)
slice_sao_chroma_flag	Value range: 0 or 1 (default)

[Note]

None

[See Also]

- [HI_MPI_VENC_SetH265Sao](#)
- [HI_MPI_VENC_GetH265Sao](#)

VENC_PARAM_H265_TIMING_S

[Description]

Defines the timing attribute of an H.265 VENC channel.

[Syntax]

```
typedef struct hiVENC_PARAM_H265_TIMING_S
{
    HI_S32 timing_info_present_flag;
    HI_S32 num_units_in_tick;
    HI_S32 time_scale;
    HI_U32 num_ticks_poc_diff_one;
} VENC_PARAM_H265_TIMING_S;
```

[Member]



Member	Description
timing_info_present_flag	For details, see the H.265 protocol. Value range: 0 (default) or 1
num_units_in_tick	For details, see the H.265 protocol. Value range: greater than 0 Default value: 1
time_scale	For details, see the H.265 protocol. Value range: greater than 0 Default value: 60
num_ticks_poc_diff_one	For details, see the H.265 protocol. Value range: [0, 4294967294] Default value: 1

[Note]

None

[See Also]

- [HI_MPI_VENC_SetH265Timing](#)
- [HI_MPI_VENC_GetH265Timing](#)

VENC_FRAMELOST_MODE_E

[Description]

Defines the frame discarding mode when the instantaneous bit rate of a VENC channel is above the threshold.

[Syntax]

```
typedef enum hiVENC_FRAMELOST_MODE_E
{
    FRMLOST_NORMAL=0,
    FRMLOST_PSKIP,
    FRMLOST_BUTT,
} VENC_FRAMELOST_MODE_E;
```

[Member]

Member	Description
FRMLOST_NORMAL	When the instantaneous bit rate is above the threshold, frames are discarded.
FRMLOST_PSKIP	When the instantaneous bit rate is above the threshold, Pskip frames are encoded.



[Note]

None

[See Also]

- [HI_MPI_VENC_SetFrameLostStrategy](#)
- [HI_MPI_VENC_GetFrameLostStrategy](#)

VENC_PARAM_FRAMELOST_S

[Description]

Defines the frame discarding policy when the instantaneous bit rate of a VENC channel is above the threshold.

[Syntax]

```
typedef struct hivENC_PARAM_FRAMELOST_S
{
    HI_BOOL bFrmLostOpen;
    HI_U32 u32FrmLostBpsThr;
    VENC_FRAMELOST_MODE_E enFrmLostMode;
    HI_U32 u32EncFrmGaps;
} VENC_PARAM_FRAMELOST_S;
```

[Member]

Member	Description
bFrmLostOpen	Frame discarding enable when the instantaneous bit rate exceeds the threshold
u32FrmLostBpsThr	Frame discarding threshold (in bit/s) Value range: [64k, 163840k]
enFrmLostMode	Frame discarding mode when the instantaneous bit rate is above the threshold. The default value is FRMLOST_NORMAL.
u32EncFrmGaps	Interval of discarding frames. The default value is 0. Value range: [0,65535]

[Note]

None

[See Also]

[HI_MPI_VENC_SetFrameLostStrategy](#)



VENC_SUPERFRAME_CFG_S

[Description]

Defines the parameters for jumbo frame processing modes.

[Syntax]

```
typedef struct hiVENC_SUPERFRAME_CFG_S
{
    VENC_SUPERFRM_MODE_E enSuperFrmMode;
    HI_U32 u32SuperIFrmBitsThr;
    HI_U32 u32SuperPfrmBitsThr;
    HI_U32 u32SuperBfrmBitsThr;
} VENC_SUPERFRAME_CFG_S;
```

[Member]

Member	Description
enSuperFrmMode	Jumbo frame processing mode. The default value is SUPERFRM_NONE.
u32SuperIFrmBitsThr	Jumbo I frame threshold. The default value is 500000. Value range: greater than or equal to 0
u32SuperPfrmBitsThr	Jumbo P frame threshold. The default value is 500000. Value range: greater than or equal to 0
u32SuperBfrmBitsThr	Jumbo B frame threshold. The default value is 500000. Value range: greater than or equal to 0

[Note]

None

[See Also]

[HI_MPI_VENC_SetSuperFrameCfg](#)

VENC_RC_PRIORITY_E

[Description]

Defines the RC priority.

[Syntax]

```
typedef enum hiVENC_RC_PRIORITY_E
{
    VENC_RC_PRIORITY_BITRATE_FIRST = 1,
    VENC_RC_PRIORITY_FRAMEBITS_FIRST,
    VENC_RC_PRIORITY_BUTT,
} VENC_RC_PRIORITY_E;
```



[Member]

Member	Description
VENC_RC_PRIORITY_BITRATE_FIRST	The target bit rate takes priority.
VENC_RC_PRIORITY_FRAMEBITS_FIRST	The jumbo frame threshold takes priority.

[Note]

None

[See Also]

None

VENC_PARAM_INTRA_REFRESH_S

[Description]

Defines the parameter for controlling I macroblock refresh in the P frame.

[Syntax]

```
typedef struct hiVENC_PARAM_INTRA_REFRESH_S
{
    HI_BOOL bRefreshEnable;
    HI_BOOL bISliceEnable;
    HI_U32 u32RefreshLineNum;
    HI_U32 u32ReqIQp;
} VENC_PARAM_INTRA_REFRESH_S;
```

[Member]

Member	Description
bRefreshEnable	I macroblock refresh enable 0 (default): disabled 1: enabled
bISliceEnable	Enable for converting the first refreshed slice into the Islice, valid only when bRefreshEnable is enabled 0 (default): disabled 1: enabled
u32RefreshLineNum	Number of rows to be refreshed during each I macroblock refresh. A larger value indicates faster refresh and lower stream smooth degree. A smaller value indicates slower refresh and higher stream smooth degree.



Member	Description
u32ReqIQp	QP value of the I frame. In intra-frame refresh mode, IDR frames may be required. The QP value of the I frame can be configured to control the quality of the inserted IDR frame. The higher the quality of the IDR frame, the larger the frame size; the lower the quality of the IDR frame, the smaller the frame size. Value range: [0, 51]

[Note]

u32RefreshLineNum is in the unit of macroblock line.

[See Also]

- [HI_MPI_VENC_GetIntraRefresh](#)
- [HI_MPI_VENC_SetIntraRefresh](#)

VENC_PARAM_MOD_S

[Description]

Defines the module parameters related to encoding.

[Syntax]

```
typedef struct hiVENC_MODPARAM_S
{
    VENC_MODTYPE_E enVencModType;
    union
    {
        VENC_PARAM_MOD_VENC_S stVencModParam;
        VENC_PARAM_MOD_H264E_S stH264eModParam;
        VENC_PARAM_MOD_H265E_S stH265eModParam;
        VENC_PARAM_MOD_JPEG_S stJpegeModParam;
    };
} VENC_PARAM_MOD_S;
```

[Member]

Member	Description
enVencModType	Type of the module parameters that are set or obtained
stVencModParam/stH264eModParam/stH265eModParam/stJpegeModParam	Structure of the module parameter for hi35xx_venc.ko , hi35xx_h264e.ko , hi35xx_h265.ko , or hi35xx_jpege.ko

[Note]



None

[See Also]

None

VENC_MODTYPE_E

[Description]

Defines the type of the module parameters related to encoding.

[Syntax]

```
typedef enum hiVENC_MODTYPE_E
{
    MODTYPE_VENC = 1,
    MODTYPE_H264E,
    MODTYPE_H265E,
    MODTYPE_JPEGE,
    MODTYPE_BUTT
} VENC_MODTYPE_E;
```

[Member]

Member	Description
MODTYPE_VENC	Type of the module parameter for hi35xx_venc.ko
MODTYPE_H264E	Type of the module parameter for hi35xx_h264e.ko
MODTYPE_H265E	Type of the module parameter for hi35xx_h265e.ko
MODTYPE_JPEGE	Type of the module parameter for hi35xx_jpege.ko

[Note]

None

[See Also]

[VENC_PARAM_MOD_S](#)

VENC_PARAM_MOD_VENC_S

[Description]

Defines the module parameter for **hi35xx_venc.ko**.

[Syntax]

```
typedef struct hiVENC_PARAM_MOD_VENC_S
{
    HI_U32 u32VencBufferCache;
} VENC_PARAM_MOD_VENC_S;
```



[Member]

Member	Description
U32VencBufferCache	Module parameter u32VencBufferCache for hi35xx_venc.ko 0: The stream buffer cache is disabled. 1: The stream buffer cache is enabled.

[Note]

None

[See Also]

[VENC_PARAM_MOD_S](#)

VENC_PARAM_MOD_H264E_S

[Description]

Defines the module parameter for **hi35xx_h264e.ko**.

[Syntax]

```
typedef struct hiVENC_PARAM_MOD_H264E_S
{
    HI_U32    u32OneStreamBuffer;
    HI_U32    u32H264eVBSource;
    HI_U32    u32H264eRcnEqualRef;
    HI_U32    u32H264eMiniBufMode;
} VENC_PARAM_MOD_H264E_S;
```

[Member]

Member	Description
u32OneStreamBuffer	Module parameter u32OneStreamBuffer for hi35xx_h264e.ko 0: multi-packet mode 1: single-packet mode
u32H264eVBSource	Module parameter u32H264eVBSource for hi35xx_h264e.ko 1: private VB 2: user VB
u32H264eRcnEqualRef	Module parameter u32H264eRcnEqualRef for hi35xx_h264e.ko 0: The reference frame does not share the space for storing the luminance data with the reconstruction frame. 1: The reference frame shares the space for storing the luminance data with the reconstruction frame.



Member	Description
u32H264eMiniBufMode	Module parameter u32H264eMiniBufMode for hi35xx_h264e.ko 0: The stream buffers are allocated based on the resolution. 1: The lower limit for the stream buffer size is 32 KB. Ensure that the size of the allocated stream buffer is appropriate.

[Note]

None

[See Also]

[VENC_PARAM_MOD_S](#)

VENC_PARAM_MOD_H265E_S

[Description]

Defines the module parameter for **hi35xx_h265.ko**.

[Syntax]

```
typedef struct hiVENC_PARAM_MOD_H265E_S
{
    HI_U32 u32OneStreamBuffer;
    HI_U32 u32H265eMiniBufMode;
} VENC_PARAM_MOD_H265E_S;
```

[Member]

Member	Description
u32OneStreamBuffer	Module parameter u32OneStreamBuffer for hi35xx_h265.ko 0: multi-packet mode 1: single-packet mode
u32H265eMiniBufMode	Module parameter u32H265eMiniBufMode for hi35xx_h265.ko 0: The stream buffers are allocated based on the resolution. 1: The lower limit for the stream buffer size is 32 KB. Ensure that the size of the allocated stream buffer is appropriate.

[Note]

None

[See Also]



VENC_PARAM_MOD_S

VENC_PARAM_MOD_JPEGE_S

[Description]

Defines the module parameter for **hi35xx_jpeg.ko**.

[Syntax]

```
typedef struct hiVENC_PARAM_MOD_JPEGE_S
{
    HI_U32 u32OneStreamBuffer;
    HI_U32 u32JpegeMiniBufMode;
} VENC_PARAM_MOD_JPEGE_S;
```

[Member]

Member	Description
u32OneStreamBuffer	Module parameter u32OneStreamBuffer for hi35xx_jpeg.ko 0: multi-packet mode 1: single-packet mode
u32JpegeMiniBufMode	Module parameter u32JpegeMiniBufMode for hi35xx_jpeg.ko 0: The stream buffers are allocated based on the resolution. 1: The lower limit for the stream buffer size is 32 KB. Ensure that the size of the allocated stream buffer is appropriate.

[Note]

None

[See Also]

[VENC_PARAM_MOD_S](#)

6.5 Error Codes

[Table 6-13](#) describes the error codes for the VENC APIs.

Table 6-13 Error codes for the VENC APIs

Error Code	Macro Definition	Description
0xA0088002	HI_ERR_VENC_INVALID_CHNID	The channel ID is invalid.
0xA0088003	HI_ERR_VENC_ILLEGAL_PARAM	The parameter is invalid.
0xA0088004	HI_ERR_VENC_EXIST	The device, channel or resource to be created or applied for exists.



Error Code	Macro Definition	Description
0xA0088005	HI_ERR_VENC_UNEXIST	The device, channel or resource to be used or destroyed does not exist.
0xA0088006	HI_ERR_VENC_NULL_PTR	The parameter pointer is null.
0xA0088007	HI_ERR_VENC_NOT_CONFIG	No parameter is set before use.
0xA0088008	HI_ERR_VENC_NOT_SUPPORT	The parameter or function is not supported.
0xA0088009	HI_ERR_VENC_NOT_PERM	The operation, for example, modifying static parameters, is forbidden.
0xA008800C	HI_ERR_VENC_NOMEM	The memory fails to be allocated due to some causes such as insufficient system memory.
0xA008800D	HI_ERR_VENC_NOBUF	The buffer fails to be allocated due to some causes such as oversize of the data buffer applied for.
0xA008800E	HI_ERR_VENC_BUF_EMPTY	The buffer is empty.
0xA008800F	HI_ERR_VENC_BUF_FULL	The buffer is full.
0xA0088010	HI_ERR_VENC_SYS_NOTREADY	The system is not initialized or the corresponding module is not loaded.
0xA0088012	HI_ERR_VENC_BUSY	The VENC system is busy.



Contents

6 VENC2	6-1
6.1 Overview	6-1
6.2 Functions	6-2
6.2.1 Data Encoding Flowchart	6-2
6.2.2 VENC Channels	6-3
6.2.3 Bit Rate Control	6-3
6.2.4 GOP Structure	6-6
6.2.5 Advanced Frame Skipping Reference Modes	6-9
6.2.6 Color-to-Gray	6-12
6.2.7 Cropping Encoding	6-12
6.2.8 ROI	6-13
6.2.9 Encoding Non-ROIs at a Low Frame Rate	6-14
6.2.10 Low Frame Rates of the OSD, ROI, and Non-ROI in QpMap Mode	6-14
6.2.11 JPEG Snapshot Modes	6-14
6.2.12 Intra-P Frame Refresh	6-15
6.2.13 Configuration Modes of Encoding Stream Frames	6-15
6.2.14 Configuration Modes of Encoding Stream Buffers	6-17
6.2.15 Recycle of the VBs for Encoding Frames	6-18
6.2.16 Calculation of the VB for Encoding Frames	6-18
6.3 API Reference	6-19
6.4 Data Structures	6-149
6.5 Error Codes	6-251



Figures

Figure 6-1 Data encoding of the VENC	6-2
Figure 6-2 Function division of a VENC channel	6-3
Figure 6-3 Position of H.265 QpMap LCU (64x64)	6-5
Figure 6-4 Position of H.264 QpMap MB.....	6-6
Figure 6-5 Frame structure in VENC_GOPMODE_DUALP mode.....	6-7
Figure 6-6 Frame structure in VENC_GOPMODE_SMARTP mode	6-8
Figure 6-7 Frame structure in AdvSmartP mode	6-8
Figure 6-8 Frame structure in VENC_GOPMODE_BIPREDB mode	6-9
Figure 6-9 Diagram of NormalP advanced frame skipping reference mode	6-10
Figure 6-10 Diagram of SmartP advanced frame skipping reference mode	6-10
Figure 6-11 Diagram of AdvSmartP advanced frame skipping reference mode.....	6-11
Figure 6-12 Diagram of DualP (u32SPinterval != 0) advanced frame skipping reference mode	6-11
Figure 6-13 Diagram of DualP (u32SPinterval = 0) advanced frame skipping reference mode	6-11
Figure 6-14 Diagram of BIPRED advanced frame skipping reference mode	6-12
Figure 6-15 Cropping encoding.....	6-13
Figure 6-16 ROI	6-14
Figure 6-17 Configuration modes of encoding stream frames	6-17
Figure 6-18 Stream packet structure	6-42
Figure 6-19 Checking whether the stream buffer is wrapped.....	6-50
Figure 6-20 Frame discarding diagram	6-133



Tables

Table 6-1 Encoding formats.....	6-1
Table 6-2 Five GOP modes supported by the GOP structure attribute	6-6
Table 6-3 Methods of calculating the value of each memory item	6-18
Table 6-4 Widths and heights of VENC channels.....	6-23
Table 6-5 Limitations on encoder attributes	6-24
Table 6-6 Public attributes of three RC modes	6-25
Table 6-7 Typical configuration of the average bit rate	6-26
Table 6-8 Mapping between enGopMode and the maximum delay time	6-38
Table 6-9 Default trans parameter values for different profile types	6-79
Table 6-10 Default entropy encoding parameter values for different profile types	6-82
Table 6-11 Requirements that u32RefreshLineNum needs to meet.....	6-140
Table 6-12 Error codes for the VENC APIs.....	6-251



6 VENC2

6.1 Overview

The video encoder (VENC) supports real-time encoding on multiple channels. Each VENC channel is independent. The encoding protocols and encoding profiles of VENC channels may be different. The VENC can also schedule the region module to overlay and cover the contents of the encoded pictures. The input sources of the VENC include:



CAUTION

The VENC2 module is supported only by Hi3519 V100, Hi3519 V101, and Hi3516C V300.

- Pictures that are read in user mode.
- Pictures that are captured by the VIU and then processed by the VPSS.
- Pictures that are captured by the VIU.

[Table 6-1](#) lists the encoding formats supported by the Hi3519 V100.

Table 6-1 Encoding formats

Chip	H.264			JPEG	MOTION JPEG	H.265	MPEG-4
	Baseline Profile	Main Profile	High Profile			Main Profile	
Hi3519 V100	Supported	Supported	Supported	Supported	Supported	Supported	Not supported
Hi3519 V101	Supported	Supported	Supported	Supported	Supported	Supported	Not supported
Hi3516C V300	Supported	Supported	Supported	Supported	Supported	Supported	Not supported

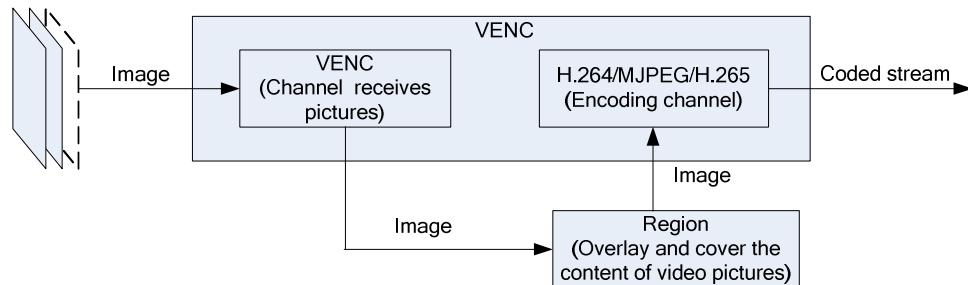


6.2 Functions

6.2.1 Data Encoding Flowchart

Figure 6-1 shows data encoding of the VENC.

Figure 6-1 Data encoding of the VENC



A typical encoding procedure includes receiving input pictures, covering and overlaying pictures, encoding pictures, and outputting streams.

The VENC consists of an encoding channel submodule and an encoding protocol submodule (H.264/H.265, JPEG, and MJPEG are supported).

The channel submodule can receive YUV pictures in the format of semi-planar YUV4:2:0 or semi-planar YUV4:2:2. For H.264 and H.265 encoding, only the semi-planar YUV4:2:0 format is supported; for JPEG and MJPEG encoding, the semi-planar YUV4:2:0 or semi-planar YUV 4:2:2 format is supported. In addition, Hi3519 V100/Hi3519 V101/Hi3516C V300 supports the single-component input (PIXEL_FORMAT_YUV_400). The channel submodule receives data of the external source picture without considering the external module from which the picture data comes from.

After receiving a picture, the channel compares the picture size with the VENC channel size. The VENC performs operations in the following cases:

- If the input picture size is greater than the VENC channel size, the VENC zooms out on the picture by using the VGS to fit into the VENC channel, and then encodes the picture.
- If the input picture size is smaller than the VENC channel size, the VENC discards the picture. The VENC does not support zoom in.
- If the input picture size is the same as the VENC channel size, the VENC encodes the picture directly.

NOTE

- The frame rate control function of the channel is disabled by default. You can enable this function by calling related MPIs. The rate controller (RC) also supports frame rate control. The frame rate control function of the RC is recommended, which ensures that the bit rate does not change severely.
- For the H.264/ H.265 encoding of Hi3519 V100/Hi3519 V101/Hi3516CV300, when the format of the input picture is changed from non-single-component format to single-component format, chrominance residue occurs on the picture before the next I frame is encoded due to inter-frame prediction quantization error. You are advised to call [HI_MPI_VENC_Resetchn](#) to reset the channel before switching the format of the input picture from non-single-component format to single-component format.

A REGION module can cover and overlay pictures.

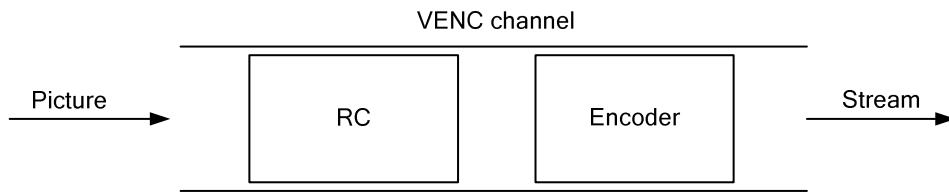


After video pre-processing and region management, the picture is sent to channel complying with a specific protocol, completing video encoding and stream output.

6.2.2 VENC Channels

A VENC channel is a basic container that stores various user configuration of VENC channels and manages the internal resources on VENC channels. A VENC channel converts pictures into streams and the conversion is completed by an RC and an encoder. The encoder completes only the encoding function; therefore, it is an encoder in narrow sense. The RC controls and adjusts the encoding parameters, controlling the output bit rate.

Figure 6-2 Function division of a VENC channel



6.2.3 Bit Rate Control

An RC mainly controls the encoding bit rate.

From the aspect of informatics, lower compression rate of a picture obtains higher picture quality. In actual scenarios, when the picture quality is stable, the encoding bit rate may fluctuate; when the encoding bit rate is stable, the picture quality may change. The following is an example using H.264 encoding. The protocol evaluates the picture quality based on the quantizer parameter (QP). Typically, when the QP value becomes small, the picture quality is improved and the encoding bit rate increases; when the QP value becomes large, the picture quality deteriorates and the encoding bit rate decreases.

Rate control must be specific to consecutive encoding streams. Therefore, the JPEG channel does not support bit rate control.

To adjust the picture quality and bit rate, the RC provides three modes for the MJPEG encoding channel: constant bit rate (CBR) mode, variable bit rate (VBR) mode, and FixQP mode; the RC provides five modes for the H.264/H.265 encoding channel: CBR, VBR, adaptive variable bit rate (AVBR), FixQP, and QpMap. The QpMap mode is provided for the H.264 and H.265 protocol channels of Hi3519 V101 and Hi3516C V300, and the bit rate control policy is determined by the user.

CBR

CBR means that a stable encoding bit rate is ensured within the bit rate statistical time. A stable bit rate is evaluated by the following two indexes.

- Bit rate statistical time (**u32StatTime**)

The statistical time unit is second. A longer statistical time indicates that the impact on bit rate adjustment exerted by the bit rate change of each frame is less, the bit rate is adjusted more slowly, and the picture quality changes more slightly.

- Adjustment amplitude of row-level bit rate control (**u32RowQpDelta**)



u32RowQpDelta indicates the maximum row-level adjustment range within one frame. The unit is the macroblock row. A larger adjustment amplitude indicates that the allowed QP row-level adjustment range is larger and the bit rate is more stable. In the scenarios where the picture complexity is unevenly distributed, the picture quality may be different in different regions if **u32RowQpDelta** is too large.

VBR

VBR means that the encoding bit rate can fluctuate within the bit rate statistical period. In this way, stable encoding picture quality can be ensured. Taking the H.264 encoding as an example, the VENC module provides four configurable parameters (**MaxQp**, **MinQp**, **MaxBitrate**, and **ChangePos**). **MaxQp** and **MinQp** are used to control the picture quality range, **MaxBitrate** is used to limit the maximum encoding bit rate within the bit rate statistical period, and **ChangePos** is used to control the bit rate base line for starting QP adjustment. When the encoding bit rate is greater than (**MaxBitrate** x **ChangePos**), the picture QP value is adjusted towards **MaxQp**. If the QP value reaches **MaxQp**, the QP value is fixed at the maximum value and **MaxBitrate** does not take effect. In this case, the encoding bit rate may exceed **MaxBitrate**. When the encoding bit rate is less than (**MaxBitrate** x **ChangePos**), the picture QP value is adjusted towards **MinQp**. If the QP value reaches **MinQp**, the encoding bit rate reaches the maximum value and the picture quality is the best.

AVBR

AVBR means that the encoding bit rate can fluctuate within the bit rate statistical period. In this way, stable encoding picture quality can be ensured. The RC detects the motion status of the current scenario, and improves the encoding bit rate in the motion scenario and reduces the target bit rate in the static scenario. Taking the H.264 encoding as an example, the VENC module provides three configurable parameters: **MaxBitrate**, **ChangePos**, and **MinStillPercent**). **MaxBitrate** indicates the maximum bit rate in the motion scenario. **MaxBitrate * ChangePos * MinStillPercent** indicates the minimum bit rate in the static scenario. The target bit rate is adjusted between the maximum bit rate and the minimum bit rate based on the motion degree. **MaxQp** and **MinQp** are used to control the picture quality range. The bit rate control takes the QP clamping as the highest priority, and bit rate control becomes invalid if the range of [MinQp, MaxQp] is exceeded.

FixQp

FixQp is a fixed QP value. Within the bit rate statistics, FixQp indicates that the QP values of all the macroblocks of encoded pictures are the same, the user-defined QP value is used, and the QP values of I frame and P frame can be set separately.

QpMap

QpMap is a QP value table. In the bit rate control mode, the strategy of bit rate control is decided by users. Each frame of the encoding picture is in the unit of 16x16 blocks. The QP value of 16x16 blocks is the QP value of the corresponding blocks set by users. The QP values of all blocks comprise the QP table. [Figure 6-3](#) shows the organization of the QP values in the table.



Figure 6-3 Position of H.265 QpMap LCU (64x64)

Position of the H.265 QpMap LCU (64x64, 128 Bits):
In the following example, the picture width is 9x64, the picture
height is 4x64, and the unit is pixel.

0	1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16	17
18	19	20	21	22	23	24	25	26
27	28	29	30	31	32	33	34	35

LCU

Value of each LCU QpMap in H.265 encoding:

16x16_0 [7:0]	16x16_1 [15:8]	16x16_2 [23:16]	16x16_3 [31:24]
16x16_4	16x16_5	16x16_6	16x16_7
16x16_8	16x16_9	16x16_10	16x16_11
16x16_12	16x16_13	16x16_14	16x16_15

LCU

Each LCU is 128 bits. Each 16x16 block occupies
8 bits.

1. Bit[7]: skip disable flag of the 16x16 block
2. Bit[6:0]: QP value. The value range is [-51, +51].
The negative values mainly apply to the relative
QP value.



Figure 6-4 Position of H.264 QpMap MB

Position of the H.264 QpMap MB (16x16, 8 bits):
In the following example, the picture width is 9x16, the picture
height is 4x16, and the unit is pixel.

MB

0	1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16	17
18	19	20	21	22	23	24	25	26
27	28	29	30	31	32	33	34	35

Value of each MB QpMap in H.264 encoding:

16x16
[7:0]

Each MB is 8 bits.

1. Bit[7]: skip disable flag of the 16x16 block
2. Bit[6:0]: QP value. The value range is [-51, +51]. The negative values mainly apply to the relative QP value.

NOTE

- In non-NORMALLP mode, the skip disable flag of the 16x16 block should be set to 1. If error check is performed, the amount of data to be processed is large and the performance is affected. Therefore, ensure that the configuration is correct when this function is used.
- In the QpMap table, the QP values must be configured in the format of complement. For example, when the relative QP is set in QpMap mode and the QP value is -20, the binary code of bit[6:0] is 110 1100 (complement of -20). The setting of bit[7] is shown in [Figure 6-3](#) and [Figure 6-4](#).
- When the QpMap table is used to implement Pskip frame encoding in the H.265 channel or when the frame rate of the non-ROI is lower than that of the ROI, the customer needs to call `HI_MPI_VENC_SetH265Sao` to disable the SAO function.

6.2.4 GOP Structure

For the H.264 and H.265 encoding protocols, the GOP structure attribute supports five GOP modes, as described in [Table 6-2](#).

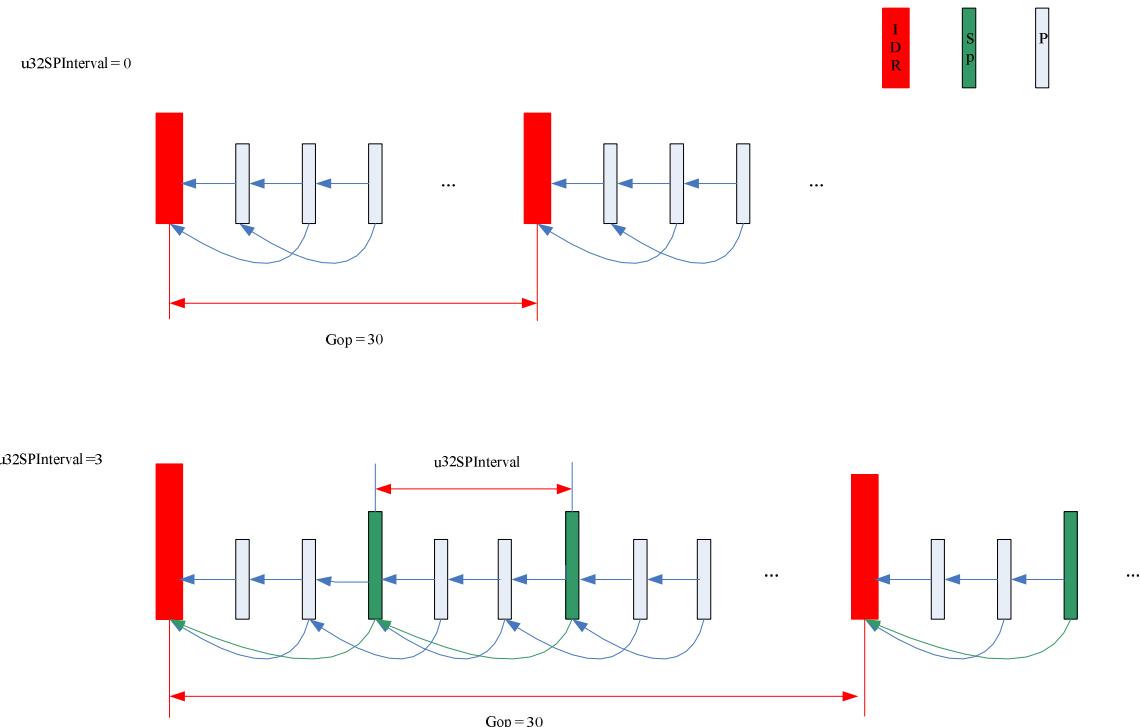
Table 6-2 Five GOP modes supported by the GOP structure attribute

GOP Mode	Number of Reference Frames	Hi3519 V100		Hi3519 V101		Hi3516C V300	
		H.264	H.265	H.264	H.265	H.264	H.265
VENC_GOPM ODE_NORMA LP	1	Supported	Supported	Supported	Supported	Supported	Supported

GOP Mode	Number of Reference Frames	Hi3519 V100		Hi3519 V101		Hi3516C V300	
		H.264	H.265	H.264	H.265	H.264	H.265
VENC_GOPM ODE_DUALP	2	Not supported	Supported	Supported	Supported	Supported	Supported
VENC_GOPM ODE_SMARTP	2	Supported	Supported	Supported	Supported	Supported	Supported
VENC_GOPM ODE_BIPREDB	2	Not supported	Not supported	Supported	Supported	Not supported	Not supported
VENC_GOPM ODE_LOWDEL AYB	2	Not supported					

- Figure 6-5 shows the frame structure in **VENC_GOPMODE_DUALP** mode.
In Figure 6-5, SP indicates a special P frame (also called an SP frame). The QP value of the SP frame is less than that of any other P frame. If **u32SPInterval** is **0**, the SP frame is not supported.

Figure 6-5 Frame structure in VENC_GOPMODE_DUALP mode



- Figure 6-6 shows the frame structure in **VENC_GOPMODE_SMARTP** mode.

In [Figure 6-6](#), Bg indicates the IDR frame, which a long-term reference frame. VI indicates the P frame, which refers only to the Bg (IDR frame). The QP value of the VI (P frame) is less than that of any other P frame.

[Figure 6-7](#) shows the frame structure in AdvSmartP mode when the AdvSmartP mode is enabled by calling [HI_MPI_VENC_EnableAdvSmartP](#).

Figure 6-6 Frame structure in VENC_GOPMODE_SMARTP mode

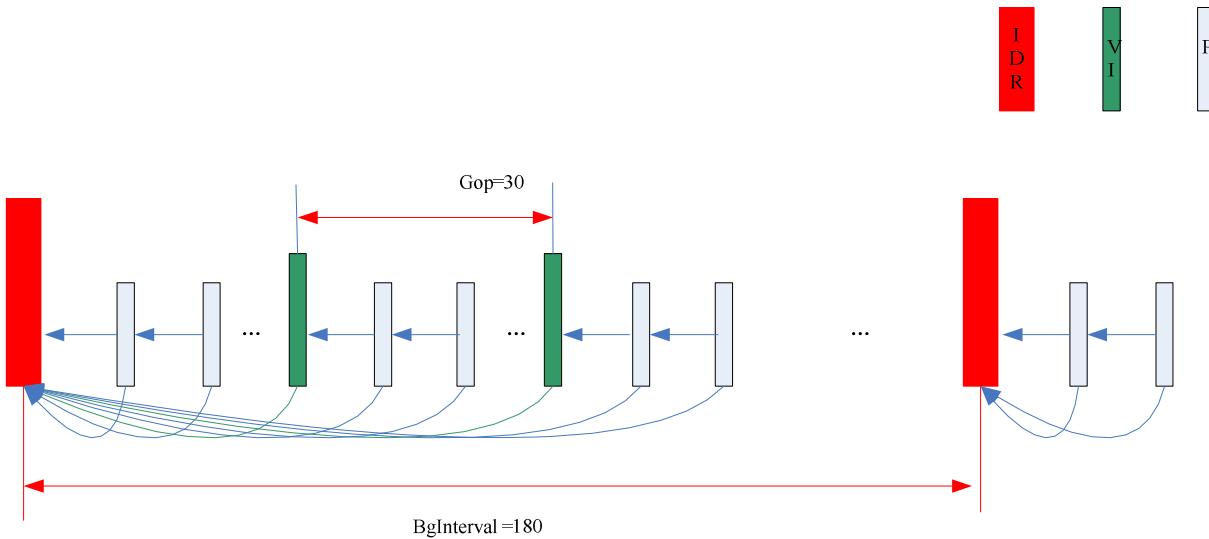
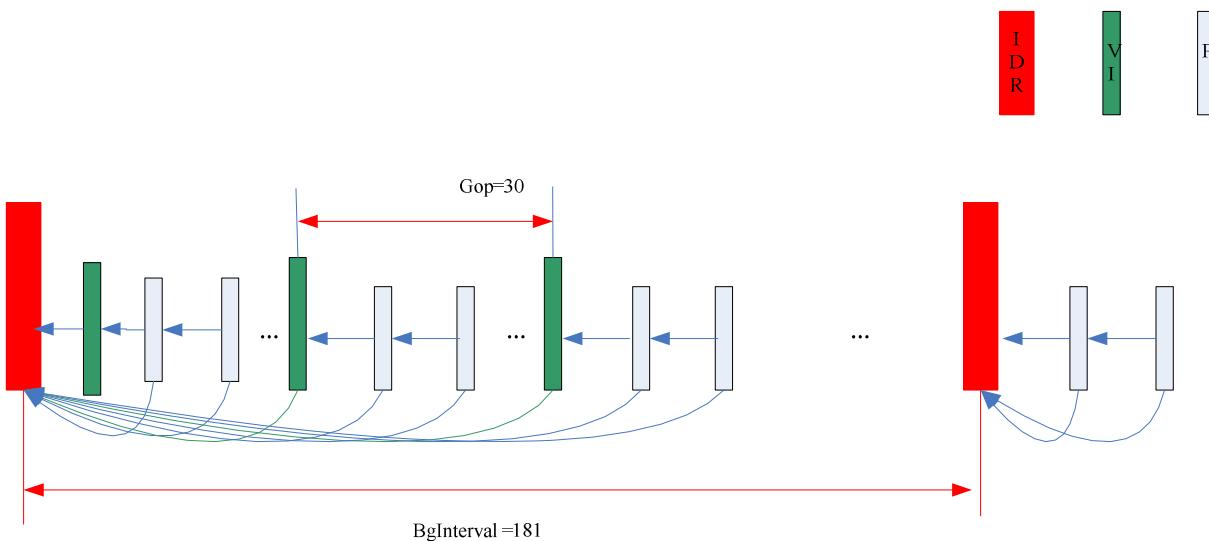


Figure 6-7 Frame structure in AdvSmartP mode

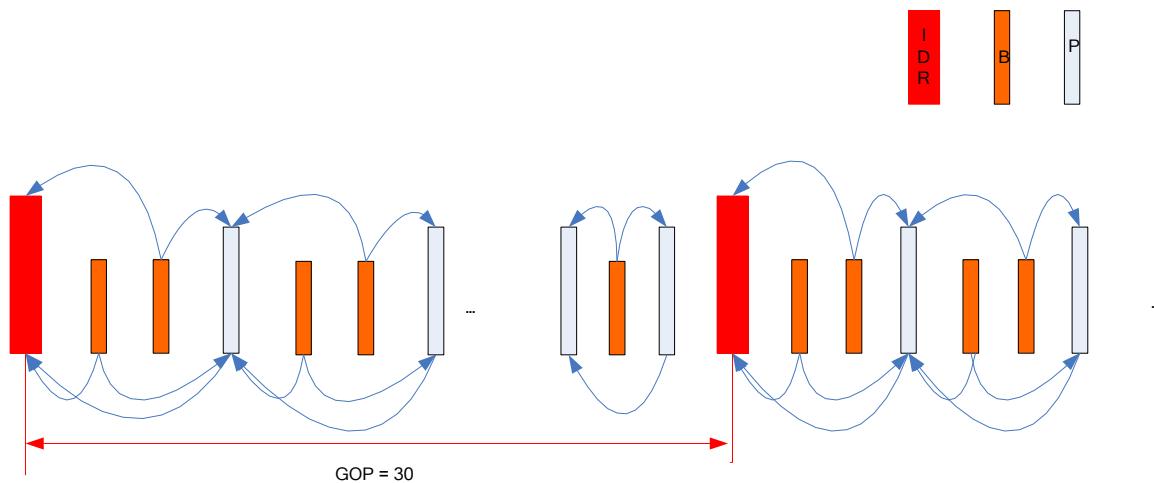


- [Figure 6-8](#) shows the frame structure in **VENC_GOPMODE_BIPREDB** mode.

In [Figure 6-8](#), **u32BFrmNum** (which is 2) indicates the number of B frames between an IDR frame and a P frame or between two P frames, and the B frame is not used as the reference frame. Note that **u32BFrmNum** may not be 2 at the end of a GOP structure, and the last frame of a GOP structure is a P frame.



Figure 6-8 Frame structure in VENC_GOPMODE_BIPREDB mode



6.2.5 Advanced Frame Skipping Reference Modes

The **u32Base**, **u32Enhance**, and **bEnablePred** parameters are involved in advanced frame skipping reference modes. For details about these parameters, see the description of [VENC_PARAM_REF_S](#). [Figure 6-9](#) to [Figure 6-14](#) shows the schematic diagram of advanced frame skipping reference modes.

NOTE

To configure the advanced frame skipping reference in DualP mode when **u32SpInterval** is not **0**, the following condition must be met: $u32Base * (u32Enhance + 1) = u32SpInterval$. For example, for a 4x advanced frame skipping reference that contains two base layers and one enhance layer, the value of **u32SpInterval** for DualP is $2 \times (1 + 1) = 4$.

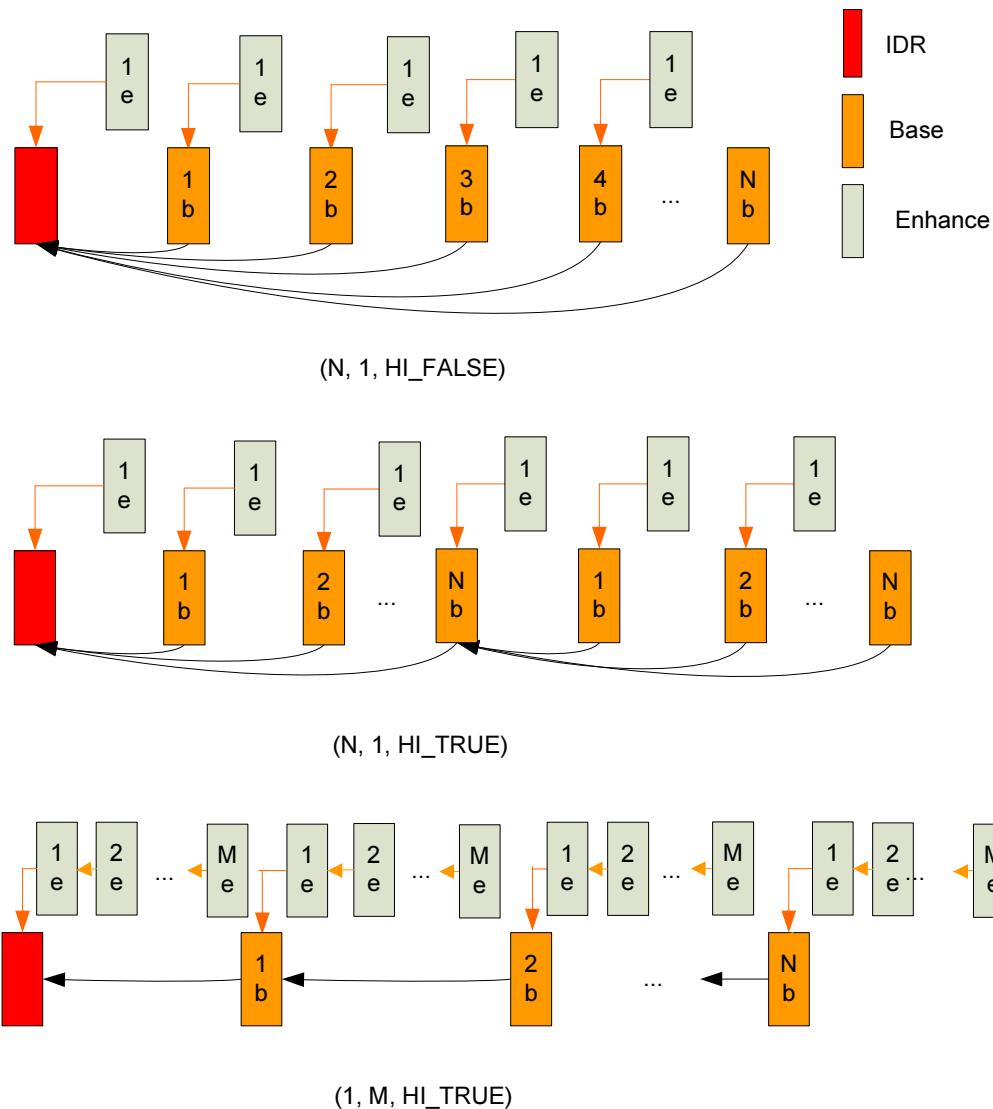
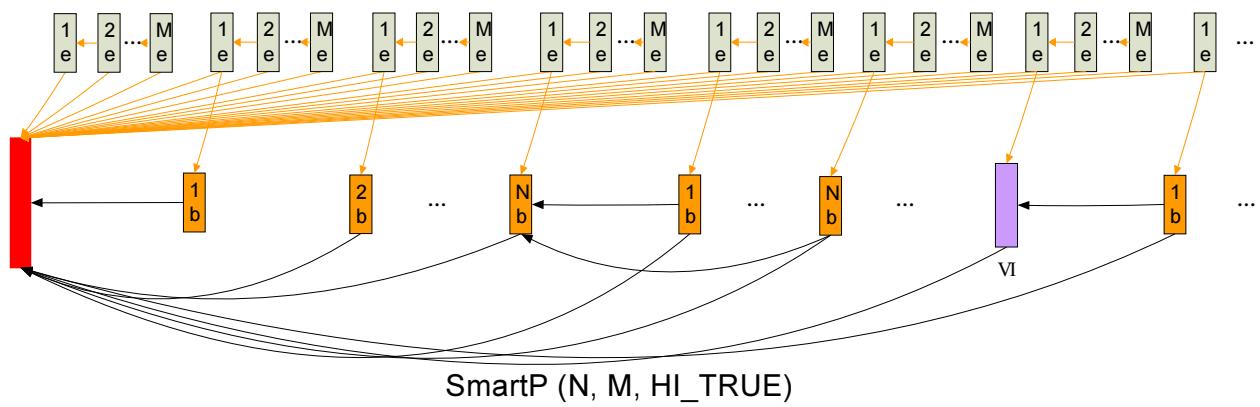
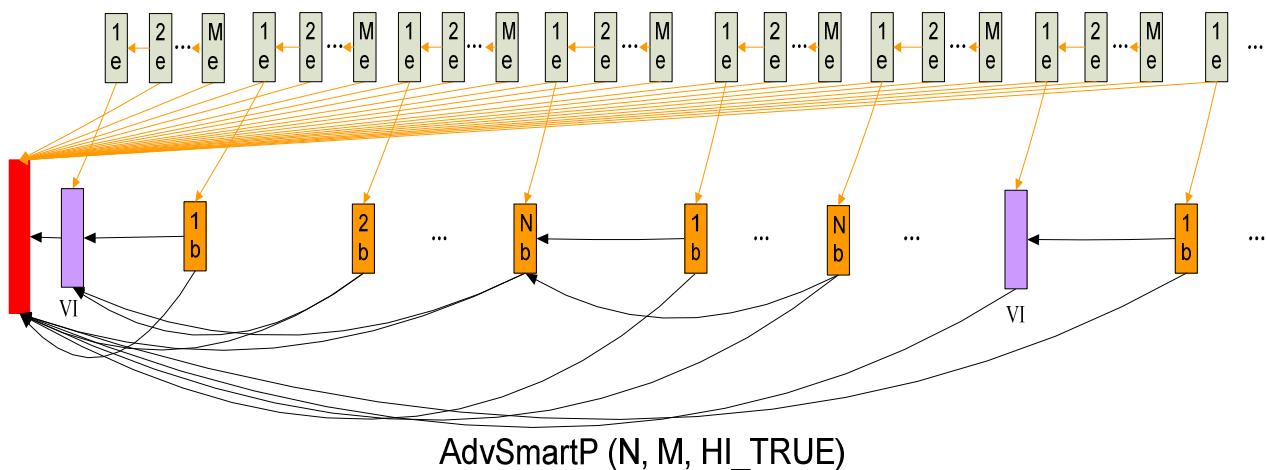
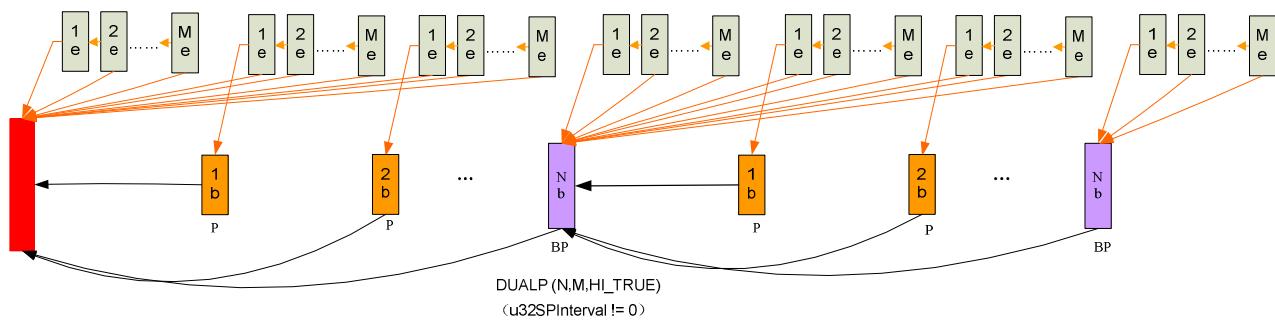
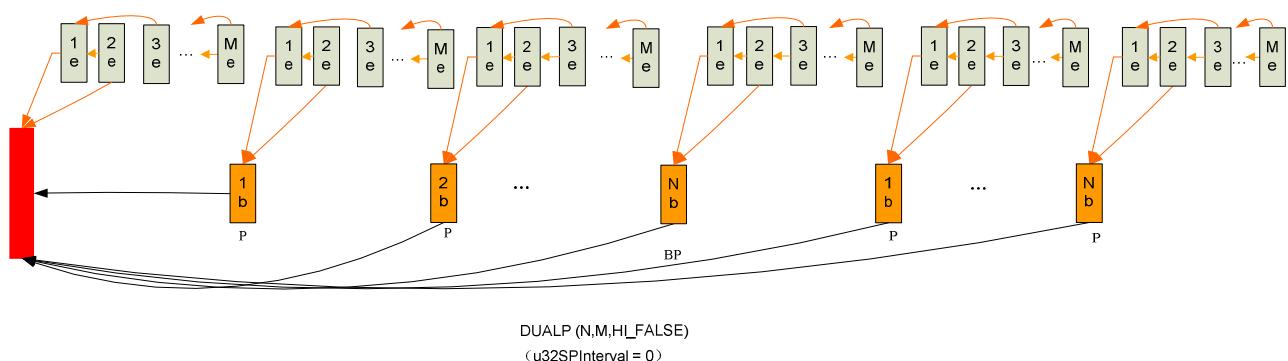
Figure 6-9 Diagram of NormalP advanced frame skipping reference mode**Figure 6-10** Diagram of SmartP advanced frame skipping reference mode

Figure 6-11 Diagram of AdvSmartP advanced frame skipping reference mode**Figure 6-12** Diagram of DualP (u32SPinterval != 0) advanced frame skipping reference mode**Figure 6-13** Diagram of DualP (u32SPinterval = 0) advanced frame skipping reference mode

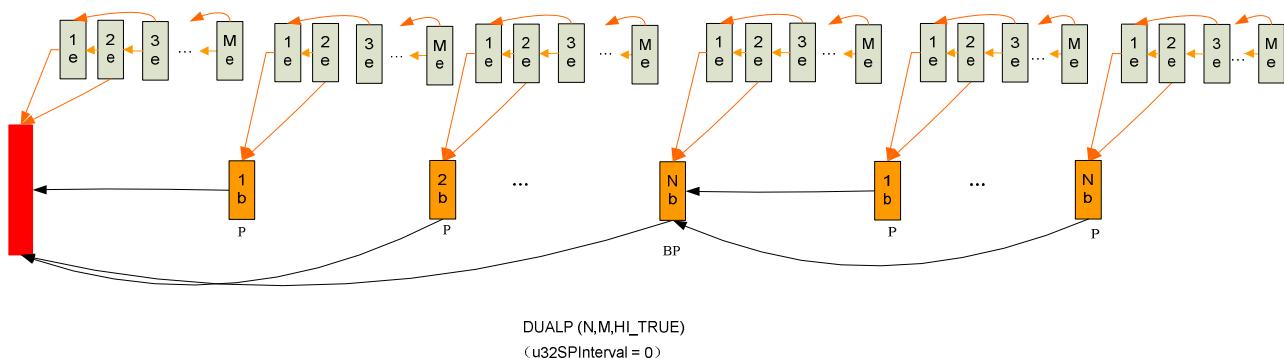
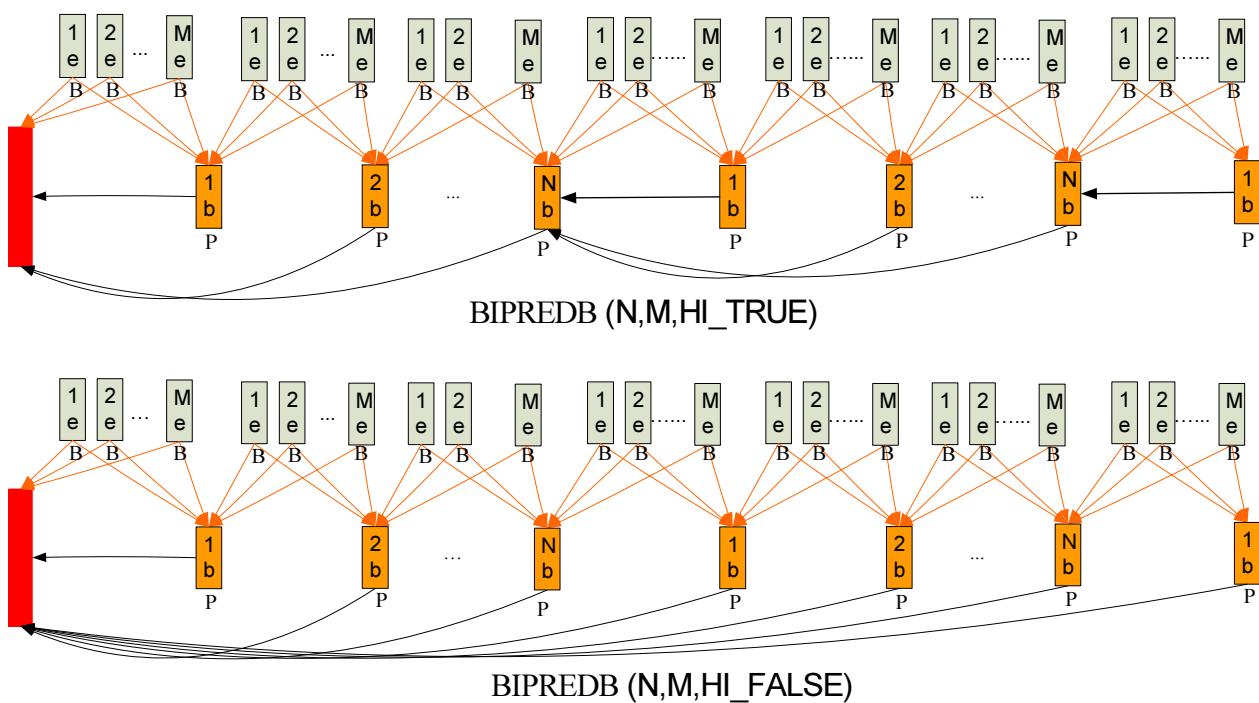


Figure 6-14 Diagram of BIPRED advanced frame skipping reference mode



6.2.6 Color-to-Gray

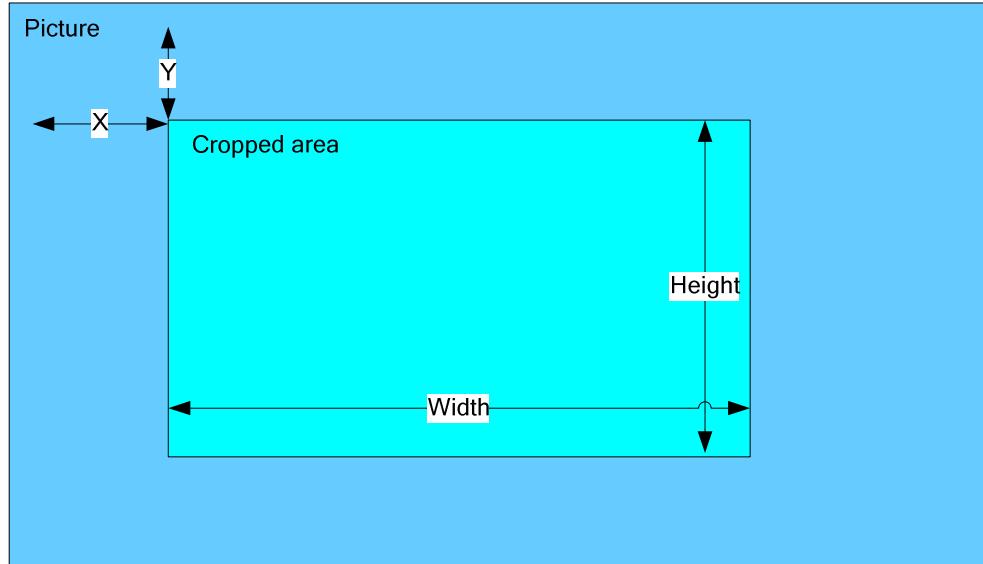
The color-to-gray function enables the VENC to convert color pictures into gray pictures for encoding. For details, see the description of [HI_MPI_VENC_SetColor2Grey](#).

6.2.7 Cropping Encoding

Cropping encoding indicates that the VENC crops a part of a picture for encoding. You can set the start position (X and Y), width, and height. For details, see the description of [HI_MPI_VENC_GetCrop](#) and [HI_MPI_VENC_SetCrop](#).



Figure 6-15 Cropping encoding



6.2.8 ROI

You can set a region of interest (ROI) to control the picture QP value of the region, distinguishing this region from other regions. The system allows you to set the ROI only for the H.264/H.265 channel and provides eight ROIs.

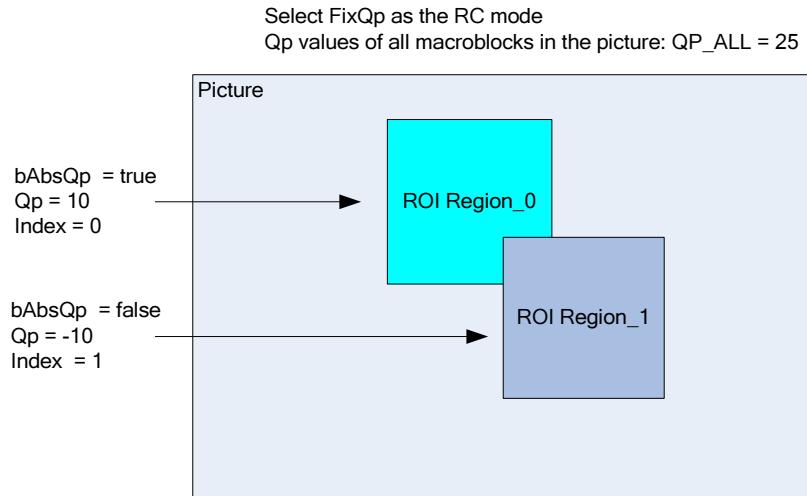
The eight regions can be overlaid with each other and the overlay priority ranges from 0 to 7. The overlay priority indicates the final QP value of picture regions when the regions are overlaid. The final QP value is set according to the region with the highest priority. An ROI can be configured with an absolute QP and a relative QP.

- Absolute QP: indicates the QP value of an ROI set by users.
- Relative QP: indicates the QP value generated during bit rate control plus the offset of the QP values set by users.

[Figure 6-16](#) shows an encoded picture in FixQp mode. The picture QP value is 25, that is, the QP values of all macroblocks of the picture are 25. ROI 0 is configured with absolute QP mode, the QP value is 10, and the index is 0; ROI 1 is configured with relative QP mode, the QP value is -10, and the index is 1. When the two regions are overlaid, the QP is set according to the region with the highest priority (ROI 1). Except the overlaid part, the QP of ROI 0 is 10 and therefore the QP of ROI 1 is 15 ($25 - 10$).



Figure 6-16 ROI



6.2.9 Encoding Non-ROIs at a Low Frame Rate

Encoding non-ROIs at a low frame rate indicates encoding ROIs at a normal frame rate. You can set the frame rate of non-ROIs as required. For details, see the description of [HI_MPI_VENC_SetRoiBgFrameRate](#) and [HI_MPI_VENC_GetRoiBgFrameRate](#).

6.2.10 Low Frame Rates of the OSD, ROI, and Non-ROI in QpMap Mode

As described in section [6.2.3 "Bit Rate Control"](#), the QP value of each 16x16 block in each picture frame is determined by the user in QpMap mode.

- In QpMap mode, the size, start point, QP value, and number of ROIs are determined by the user. As shown in [Figure 6-3](#) and [Figure 6-4](#), the low frame rate of the non-ROI is also determined by the user. In this case, calling [HI_MPI_VENC_SetRoiCfg](#) has no effect.
- In QpMap mode, the region module is used to overlay the Cover and Overlay regions on the picture. At most eight regions can be overlaid. The QP values of the Overlay regions are determined by the QpMap table. The QP values of the Overlay regions configured by the region module do not take effect.

6.2.11 JPEG Snapshot Modes

The JPEG VENC channels of Hi3519 V100, Hi3519 V101, and Hi3516C V300 support two modes: snapshot all mode and flash snapshot mode.

- Snapshot all mode: After the channels start to receive pictures, all received pictures are encoded.
- Flash snapshot mode: After the channels start to receive channels, only the pictures captured when the camera flash is on are encoded.



6.2.12 Intra-P Frame Refresh

By refreshing the Islice in the P frame, smooth stream encoding is ensured and the size of an I frame can be close to that of a P frame. When the network bandwidth is limited (such as the wireless network), the intra-P frame refresh can reduce the heavy network traffic caused by large I frames, reduce network transmission delay, and decrease the probability of network transmission errors.

6.2.12.1 Usage

To use the intra-P frame refresh function, perform the following steps:

- Step 1** Configure the refresh enable (**bRefreshEnable**) and Islice conversion enable (**bISliceEnable**) by calling [HI_MPI_VENC_SetIntraRefresh](#).
- After the refresh function is enabled, the SDK automatically implements intra-macroblock refresh from the first frame of each group of pictures (GOP) from top to bottom based on the configured number of rows to be refreshed (**u32RefreshLineNum**). The refresh interval is one GOP cycle.
 - After Islice conversion is enabled, the SDK automatically converts the intra-frame prediction macroblock in the first frame to be refreshed into an Islice. If the Islice is refreshed, the stream compatibility is better.
- Step 2** Adjust [VENC_GOP_ATTR_S:: stGopAttr.stNormalP.s32IPQpDelta](#) to control the size of the start frame of I macroblock refresh (the bit rate control still takes effect after [HI_MPI_VENC_SetIntraRefresh](#) is called). The recommended value is -1.

----End

6.2.12.2 Precautions

- The number of rows to be refreshed (**u32RefreshLineNum**) must be set to an appropriate value by calling [HI_MPI_VENC_SetIntraRefresh](#) to ensure that all the macroblock rows in the picture are refreshed once within one GOP cycle.
- The number of rows to be refreshed (**u32RefreshLineNum**) is related to the encoding parameter GOP and frame skipping reference parameters. Therefore, after the encoding attributes and advanced frame skipping reference parameters are configured by calling the corresponding MPIs, [HI_MPI_VENC_SetIntraRefresh](#) needs to be called to reconfigure **u32RefreshLineNum**.
- [HI_MPI_VENC_SetIntraRefresh](#) is valid only for the H.264/H.265 protocol.
- If the refresh function (**bISliceEnable**) is enabled by calling [HI_MPI_VENC_SetIntraRefresh](#), the frame in which the Islice is refreshed automatically splits slices based on the size of the refreshed Islice. In this case, the configured slice split MPI does not take effect.
- When **bRefreshEnable** is set to True, **bISliceEnable** must be set to True.

6.2.13 Configuration Modes of Encoding Stream Frames

The encoding stream frames can be configured in single-packet mode and multi-packet mode (when the slice split interface and the user data insertion interface are not called), as shown in [Figure 6-17](#).

- Multi-packet mode: For H.264 encoding, when frames are I frames, [HI_MPI_VENC_GetStream](#) is called. An I frame contains four network abstraction layer

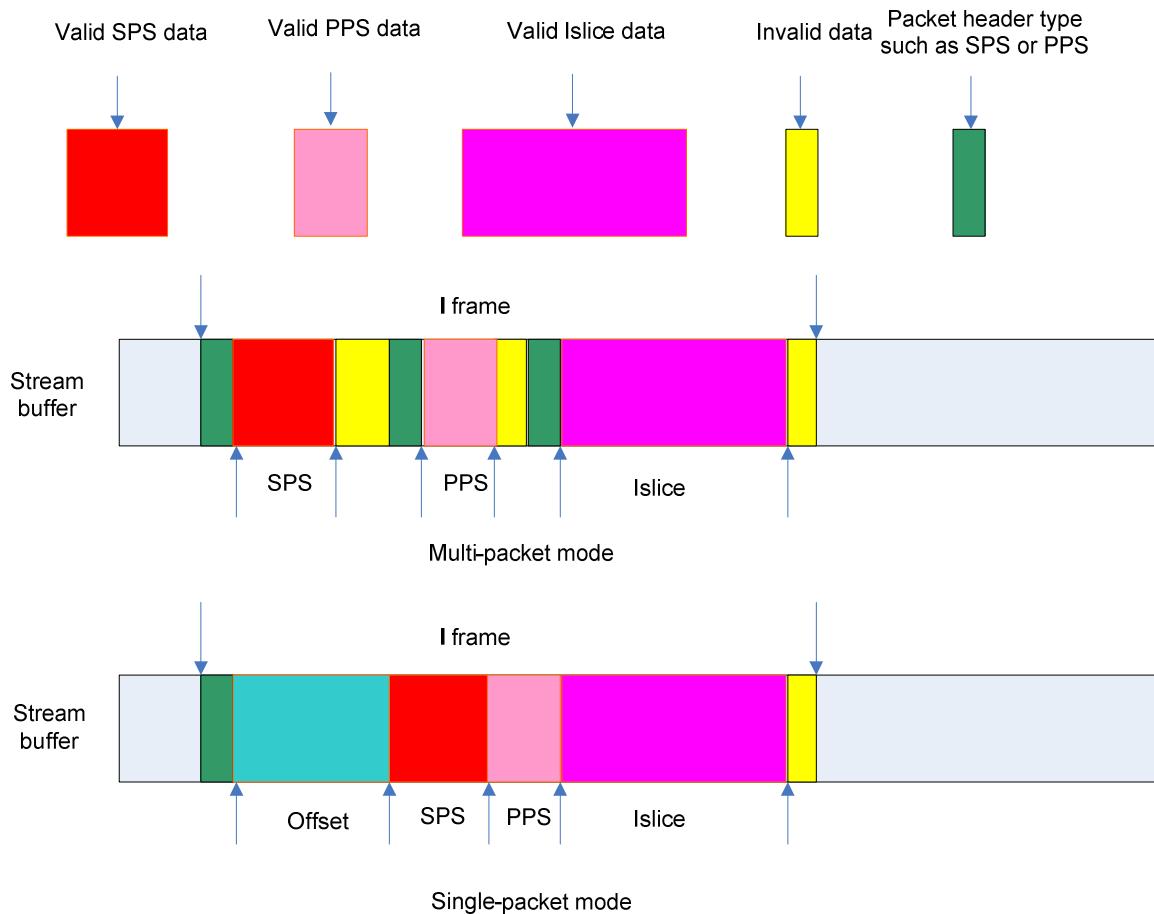


(NAL) packets, including a sequence parameter set (SPS) packet, a picture parameter set (PPS) packet, an SEI packet, and an Islice packet (assuming that there is only one PPS packet and the four NAL packets are independent and differ in the packet type). For JPEG encoding, a frame of picture contains two packets, including a picture parameter packet and a picture data packet (the two packets are independent and differ in the packet type).

- Single-packet mode: For H.264 encoding, when frames are I frames, [HI_MPI_VENC_GetStream](#) is called. An I frame contains one Islice-type NAL packet (the packet contains data of SPS, PPS, SEI, and Islice). For JPEG encoding, a frame of picture contains only one picture data packet (the packet contains data of the picture parameter packet).

The configuration mode can be selected by setting the **H264eOneStreamBuffer**, **H265eOneStreamBuffer**, or **JpegeOneStreamBuffer** parameter when .ko drivers are loaded. If the module parameter is **1**, the single-packet mode is selected. If the module parameter is **0**, the multi-packet mode is selected. The default value is 0.

If the packet split interface such as [HI_MPI_VENC_SetH264SliceSplit](#) is called, a frame is split into multiple slices. If the single-packet mode is selected, the first Islice packet of an I frame contains data of SPS, PPS, and SEI, but other Islice packets of this I frame do not contain such data. For H.264 encoding, data of SPS, PPS, and SEI exists only in the first Islice packet of an I frame and merges into one Islice packet. For JPEG/MJPEG encoding, data of the picture parameter packet exists only in the first data packet of a frame and merges into one data packet.

Figure 6-17 Configuration modes of encoding stream frames

6.2.14 Configuration Modes of Encoding Stream Buffers

The encoding stream buffer supports two configuration modes: normal mode and memory reduction mode.

- Normal mode: In consideration of jumbo frames, the minimum size of the stream buffer is (Channel width x Channel height x 3/4) for H.264 and H.265 encoding and (Channel width x Channel height) for JPEG and MJPEG encoding.
- Memory reduction mode: The minimum size of the stream buffer is (32 x 1024) bytes. The stream buffer size must be set to an appropriate value in this mode. Otherwise, frames may be re-encoded continuously or discarded due to the insufficiency of the stream buffer.

The mode can be specified by setting corresponding module parameters when .ko drivers are loaded. If the parameter value is 1, the memory reduction mode is used; if the parameter value is 0, the normal mode is used.

- Module parameter of **hi35xx_h264e.ko**: **H264eMiniBufMode**
- Module parameter of **hi35xx_h265e.ko**: **H265eMiniBufMode**
- Module parameter of **hi35xx_jpeg.ko**: **JpegMinBufMode**



6.2.15 Recycle of the VBs for Encoding Frames

- At most three reference frames are supported in H.264 and H.265 encoding modes. When the encoding stream with multiple reference frames is switched to the one with fewer reference frames, some VBs for the encoding reference frames are idle. When the AdvSmartP function is disabled, AdvSmartPBuffer is idle. If the VENC module parameter **FrameBufRecycle** is 1, the VBs RefBuffer and AdvSmartPBuffer are recycled when they are idle. That is, the corresponding VB pool is destroyed. If the VBs are recycled, many memory fragments will be generated after the system changes the number of reference frames or disables/enables AdvSmartP for many times. The default value of **FrameBufRecycle** is 0, indicating that the VBs for encoding frames are not recycled. The function of recycling the VBs for encoding frames is valid only in PrivateVB mode.
- FrameBufRecycle** is a module parameter for **hi35xx_venc.ko**.

6.2.16 Calculation of the VB for Encoding Frames

- The size of each VB for encoding frames (reference frames and reconstruction frames) is calculated as follows:
$$\text{FrameSize} = \text{YHeaderSize} + \text{CHeaderSize} + \text{YSize} + \text{CSize} + \text{PmeSize} + \text{PmeInfoSize} + \text{TmvSize}$$
- Table 6-3** shows how to calculate the value of each memory item in the preceding formula. Note that align(x,y) is equal to $(x + y - 1)/y * y$.

Table 6-3 Methods of calculating the value of each memory item

Memory Item		H.264	H.265
YHeader Size	Compressed	$(\text{align}(\text{width}, 256) >> 8) * (\text{align}(\text{height}, 16) >> 4) * 32$	$(\text{align}(\text{width}, 64) >> 6) * (\text{align}(\text{height}, 64) >> 6) * 32$
	Uncompressed	0	0
YSize	Compressed	$\text{align}(\text{width}, 64) * \text{align}(\text{height}, 16)$	$\text{align}(\text{width}, 64) * \text{align}(\text{height}, 64)$
	Uncompressed	$\text{align}(\text{width}, 16) * \text{align}(\text{height}, 16)$	$\text{align}(\text{width}, 16) * \text{align}(\text{height}, 16)$
CHeaderSize		YHeaderSize	YHeaderSize
CSize		YSize/2	YSize/2
PmeSize		$(\text{align}(\text{width}, 64) >> 6) * (\text{align}(\text{height}, 16) >> 4) << 6$	$(\text{align}(\text{width}, 64) >> 6) * (\text{align}(\text{height}, 64) >> 6) << 8$
PmeInfoSize		$((\text{align}(\text{width}, 2048) >> 4) * (\text{align}(\text{height}, 16) >> 4) + 7) >> 3$	$(\text{align}(\text{width}, 512) >> 6) * (\text{align}(\text{height}, 64) >> 6) << 2$
TmvSize		<ul style="list-style-type: none">• VENC_GOPMODE_BIPREDB: $(\text{align}(\text{width}, 64) >> 6) * (\text{align}(\text{height}, 64) >> 4) << 7$• Non-VENC_GOPMODE_BIPREDB:	$(\text{align}(\text{width}, 64) >> 6) * (\text{align}(\text{height}, 64) >> 6) << 7$



Memory Item	H.264	H.265
	0	

NOTE

If the PrivateVB mode is used for encoding, the encoder uses **MaxFrameSize** to allocate the VBs to the reference frames and reconstruction frames, which is obtained by substituting **MaxPicWidth** and **MaxPicHeight** into the formula for calculating the VB size. If **FrameSize** calculated by substituting **PicWidth** and **PicHeight** (within **MAX_WIDTH** and **MAX_HEIGHT** limited by the encoder) into the formula for calculating the VB size does not exceed **MaxFrameSize**, pictures with such a resolution can be encoded. For details about **MAX_WIDTH** and **MAX_HEIGHT**, see [Table 6-4](#).

6.3 API Reference

The VENC has the following functions:

- Creates and destroys a VENC channel.
- Resets a VENC channel.
- Starts or stops picture reception.
- Sets and obtains VENC channel attributes.
- Obtains and releases streams.

The VENC provides the following MPIs:

- [**HI_MPI_VENC_CreateChn**](#): Creates a VENC channel.
- [**HI_MPI_VENC_DestroyChn**](#): Destroys a VENC channel.
- [**HI_MPI_VENC_Resetchn**](#): Resets a VENC channel.
- [**HI_MPI_VENC_StartRecvPic**](#): Enables a VENC channel to start receiving input pictures.
- [**HI_MPI_VENC_StartRecvPicEx**](#): Enables a VENC channel to start receiving input pictures and automatically stop the VENC channel receiving input pictures after the specified number of frames is exceeded.
- [**HI_MPI_VENC_StopRecvPic**](#): Stops a VENC channel receiving input pictures.
- [**HI_MPI_VENC_Query**](#): Queries the status of a VENC channel.
- [**HI_MPI_VENC_SetChnAttr**](#): Sets attributes of a VENC channel.
- [**HI_MPI_VENC_GetChnAttr**](#): Obtains attributes of a VENC channel.
- [**HI_MPI_VENC_GetStream**](#): Obtains encoded streams.
- [**HI_MPI_VENC_ReleaseStream**](#): Releases a stream buffer.
- [**HI_MPI_VENC_GetStreamBufInfo**](#): Obtains the physical address and size of a stream buffer.
- [**HI_MPI_VENC_InsertUserData**](#): Inserts user data.
- [**HI_MPI_VENC_SendFrame**](#): Sends raw pictures for encoding.
- [**HI_MPI_VENC_SendFrameEx**](#): Allows the user to send the raw picture and encode the picture based on the picture information in the QpMap table.
- [**HI_MPI_VENC_SetMaxStreamCnt**](#): Sets the maximum number of frames in a stream buffer.



- [**HI_MPI_VENC_GetMaxStreamCnt**](#): Obtains the maximum number of frames in a stream buffer.
- [**HI_MPI_VENC_RequestIDR**](#): Requests an intermediate data rate (IDR) frame.
- [**HI_MPI_VENC_EnableIDR**](#): Enables an IDR frame.
- [**HI_MPI_VENC_SetH264IdrPicId**](#): Sets the idr_pic_id of an IDR frame.
- [**HI_MPI_VENC_GetH264IdrPicId**](#): Obtains the idr_pic_id of an IDR frame.
- [**HI_MPI_VENC_GetFd**](#): Obtains the device file handle of a VENC channel.
- [**HI_MPI_VENC_CloseFd**](#): Closes the device file handle of a VENC channel.
- [**HI_MPI_VENC_SetRoiCfg**](#): Sets the ROI configuration of a channel.
- [**HI_MPI_VENC_GetRoiCfg**](#): Obtains the ROI configuration of a channel.
- [**HI_MPI_VENC_SetRoiBgFrameRate**](#): Sets the frame rate attribute of non-ROIs in a VENC channel.
- [**HI_MPI_VENC_GetRoiBgFrameRate**](#): Obtains the frame rate attribute of non-ROIs in a VENC channel.
- [**HI_MPI_VENC_SetH264SliceSplit**](#): Sets the slice attribute of an H.264 channel.
- [**HI_MPI_VENC_GetH264SliceSplit**](#): Obtains the slice attribute of an H.264 channel.
- [**HI_MPI_VENC_SetH264InterPred**](#): Sets the inter-prediction attribute of an H.264 channel.
- [**HI_MPI_VENC_GetH264InterPred**](#): Obtains the inter-prediction attribute of an H.264 channel.
- [**HI_MPI_VENC_SetH264IntraPred**](#): Sets the intra-prediction attribute of an H.264 channel.
- [**HI_MPI_VENC_GetH264IntraPred**](#): Obtains the intra-prediction attribute of an H.264 channel.
- [**HI_MPI_VENC_SetH264Trans**](#): Sets the transformation and quantization attribute of an H.264 channel.
- [**HI_MPI_VENC_GetH264Trans**](#): Obtains the transformation and quantization attribute of an H.264 channel.
- [**HI_MPI_VENC_SetH264Entropy**](#): Sets the entropy encoding mode for an H.264 channel.
- [**HI_MPI_VENC_GetH264Entropy**](#): Obtains the entropy encoding mode for an H.264 channel.
- [**HI_MPI_VENC_SetH264Poc**](#): Sets the POC type of an H.264 channel.
- [**HI_MPI_VENC_GetH264Poc**](#): Obtains the POC type of an H.264 channel.
- [**HI_MPI_VENC_SetH264Dblk**](#): Sets the de-blocking type of an H.264 channel.
- [**HI_MPI_VENC_GetH264Dblk**](#): Obtains the de-blocking type of an H.264 channel.
- [**HI_MPI_VENC_SetH264Vui**](#): Sets the video usability information (VUI) parameters of an H.264 channel.
- [**HI_MPI_VENC_GetH264Vui**](#): Obtains the VUI parameter settings of an H.264 channel.
- [**HI_MPI_VENC_SetH265Vui**](#): Sets the VUI parameters of an H.265 VENC channel.
- [**HI_MPI_VENC_GetH265Vui**](#): Obtains the VUI parameter settings of an H.265 VENC channel.
- [**HI_MPI_VENC_SetJpegParam**](#): Sets the parameter set of a JPEG channel.
- [**HI_MPI_VENC_GetJpegParam**](#): Obtains the parameter set of a JPEG channel.
- [**HI_MPI_VENC_SetMjpegParam**](#): Sets the advanced parameters of an MJPEG channel.



- [**HI_MPI_VENC_GetMjpegParam**](#): Obtains the configurations of the advanced parameters of an MJPEG channel.
- [**HI_MPI_VENC_SetFrameRate**](#): Sets the frame rate control attribute of a VENC channel.
- [**HI_MPI_VENC_GetFrameRate**](#): Obtains the frame rate control attribute of a VENC channel.
- [**HI_MPI_VENC_SetRcParam**](#): Sets the RC advanced parameters of a VENC channel.
- [**HI_MPI_VENC_GetRcParam**](#): Obtains the configurations of the advanced parameters for the RC of a VENC channel.
- [**HI_MPI_VENC_SetRefParam**](#): Sets the Advanced frame skipping reference parameters for an H.264/H.265 VENC channel
- [**HI_MPI_VENC_GetRefParam**](#): Obtains the Advanced frame skipping reference parameters for an H.264/H.265 VENC channel
- [**HI_MPI_VENC_SetColor2Grey**](#): Enables or disables the color-to-gray function of a VENC channel.
- [**HI_MPI_VENC_GetColor2Grey**](#): Obtains the enable status of the color-to-gray function of a VENC channel.
- [**HI_MPI_VENC_SetCrop**](#): Sets the cropping attribute of a VENC channel.
- [**HI_MPI_VENC_GetCrop**](#): Obtains the cropping attribute of a VENC channel.
- [**HI_MPI_VENC_SetJpegSnapMode**](#): Sets the snapshot mode of a JPEG snapshot channel.
- [**HI_MPI_VENC_GetJpegSnapMode**](#): Obtains the snapshot mode of a JPEG snapshot channel.
- [**HI_MPI_VENC_SetH265SliceSplit**](#): Sets the slice split attribute for H.265 encoding.
- [**HI_MPI_VENC_GetH265SliceSplit**](#): Obtains the slice split attribute for H.265 encoding.
- [**HI_MPI_VENC_SetH265PredUnit**](#): Sets the prediction unit (PU) attribute for H.265 encoding.
- [**HI_MPI_VENC_GetH265PredUnit**](#): Obtains the PU attribute for H.265 encoding.
- [**HI_MPI_VENC_SetH265Trans**](#): Sets the transformation and quantization attribute for H.265 encoding.
- [**HI_MPI_VENC_GetH265Trans**](#): Obtains the transformation and quantization attribute for H.265 encoding.
- [**HI_MPI_VENC_SetH265Entropy**](#): Sets the entropy encoding attribute of an H.265 channel.
- [**HI_MPI_VENC_GetH265Entropy**](#): Obtains the entropy encoding attribute of an H.265 channel.
- [**HI_MPI_VENC_SetH265Sao**](#): Sets the sample adaptive offset (SAO) attribute for H.265 encoding.
- [**HI_MPI_VENC_GetH265Sao**](#): Obtains the SAO attribute for H.265 encoding.
- [**HI_MPI_VENC_SetH265Dblk**](#): Sets the deblocking attribute for H.265 encoding.
- [**HI_MPI_VENC_GetH265Dblk**](#): Obtains the deblocking attribute for H.265 encoding.
- [**HI_MPI_VENC_SetH265Timing**](#): Sets the timing attribute of VPS packets for H.265 encoding.
- [**HI_MPI_VENC_GetH265Timing**](#): Obtains the timing attribute of VPS packets for H.265 encoding.
- [**HI_MPI_VENC_SetFrameLostStrategy**](#): Sets the frame discarding policies when the instantaneous bit rate is above the threshold.



- [HI_MPI_VENC_GetFrameLostStrategy](#): Obtains the frame discarding policies when the instantaneous bit rate is above the threshold.
- [HI_MPI_VENC_SetSuperFrameCfg](#): Sets the processing mode of jumbo frames for the encoding channel.
- [HI_MPI_VENC_GetSuperFrameCfg](#): Obtains the processing mode of jumbo frames for the encoding channel.
- [HI_MPI_VENC_SetChnlPriority](#): Sets the channel priority.
- [HI_MPI_VENC_GetChnlPriority](#): Obtains the channel priority.
- [HI_MPI_VENC_GetIntraRefresh](#): Obtains the parameter of refreshing the Islice in the P frame.
- [HI_MPI_VENC_SetIntraRefresh](#): Sets the parameter of refreshing the Islice in the P frame.
- [HI_MPI_VENC_SetModParam](#): Sets the module parameters related to encoding.
- [HI_MPI_VENC_GetModParam](#): Obtains the module parameters related to encoding.
- [HI_MPI_VENC_SetSSERegion](#): Sets the sum of squared errors (SSE) attributes of an H.265 channel.
- [HI_MPI_VENC_GetSSERegion](#): Obtains the SSE attributes of an H.265 channel.
- [HI_MPI_VENC_SetVencAdvancedParam](#): Sets the advanced parameters of the VENC.
- [HI_MPI_VENC_GetVencAdvancedParam](#): Obtains the advanced parameters of the VENC.
- [HI_MPI_VENC_EnableAdvSmartP](#): Enables/Disables the advanced SmartP mode.

HI_MPI_VENC_CreateChn

[Description]

Creates a VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_CreateChn(VENC_CHN VeChn, const VENC_CHN_ATTR_S
*pstAttr);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstAttr	Pointer to the VENC channel attributes	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."



[Requirement]

- Header files: hi_comm_venc.h, hi_comm_rc.h, mpi_venc.h
- Library file: libmpi.a

[Difference]

The widths and heights of VENC channels vary depending on chip type. See [Table 6-4](#).

Table 6-4 Widths and heights of VENC channels

Chip	Width or Height (Pixel)	H.264	H.265	JPEG	MOTION JPEG
Hi3519 V100	MIN_WIDTH	192	128	32	32
	MAX_WIDTH	4608	4608	8192	8192
	MIN_HEIGHT	128	128	32	32
	MAX_HEIGHT	4608	4608	8192	8192
	MIN_ALIGN	2	2	4	4
Hi3519 V101	MIN_WIDTH	256	128	32	32
	MAX_WIDTH	4608	4608	8192	8192
	MIN_HEIGHT	128	128	32	32
	MAX_HEIGHT	4608	4608	8192	8192
	MIN_ALIGN	2	2	4	4
Hi3516C V300	MIN_WIDTH	256	128	32	32
	MAX_WIDTH	1920	1920	2048	2048
	MIN_HEIGHT	128	128	32	32
	MAX_HEIGHT	1920	1920	2048	2048
	MIN_ALIGN	2	2	4	4

[Note]

- The attributes of a VENC channel include the attributes of the encoder, bit rate controller, and the frame structure type. The attributes of the frame structure type are called the GOP type attributes.
- To set the attributes of the encoder, select encoding protocols and set attributes of each protocol.
- The maximum width and height defined in the encoder attribute and the channel width and height must meet the following conditions:
 - MaxPicWidth \in [MIN_WIDTH, MAX_WIDTH]
 - MaxPicHeight \in [MIN_HEIGHT, MAX_HEIGHT]



- PicWidth $\in [\text{MIN_WIDTH}, \text{MAX_WIDTH}]$
- PicHeight $\in [\text{MIN_HEIGHT}, \text{MAX_HEIGHT}]$
- The maximum width, maximum height, channel width, or channel height must be an integral multiple of MIN_ALIGN.

MIN_WIDTH, MAX_WIDTH, MIN_HEIGHT, and MAX_HEIGHT indicate the minimum width, maximum width, minimum height, and maximum height of a VENC channel, respectively. MIN_ALIGN indicates the minimum alignment unit (in pixel).

For Hi3516C V300, the width and height of an H.264/H.265 channel cannot be greater than 1440 at the same time, and the product of the width and height cannot exceed 1920 x 1296.

- The encoder attributes including the VENC buffer depth and mode of obtaining streams must be set. [Table 6-5](#) describes the limitations on encoder attributes based on protocols.

NOTE

Note that **MaxPicwidth** and **MaxPicheight** must be 16-byte-aligned before the buffer depth for MJPEG/JPEG encoding is calculated, and be 64-byte-aligned before the buffer depth for H.264/H.265 encoding is calculated.

Table 6-5 Limitations on encoder attributes

Encoding Protocol	Encoding Mode	Depth of the Stream Buffer	Stream Acquisition Mode	Encoding Profile
H.264	Frame	When H264eMiniBufMode is 0 Buffer depth $\geq \text{MaxPicwidth} \times \text{MaxPicheight} \times 3/4$ When H264eMiniBufMode is 1 Buffer depth $\geq 32 \times 1024$ bytes	Frame/Slice	Baseline Main profile High profile
MJPEG	Frame	When JpegeMiniBufMode is 0 Buffer depth $\geq \text{MaxPicwidth} \times \text{MaxPicheight} \times 1$ When JpegeMiniBufMode is 1 Buffer depth $\geq 32 \times 1024$ bytes	Frame/Ecs	-
JPEG	Frame	When JpegeMiniBufMode is 0 Buffer depth $\geq \text{MaxPicwidth} \times \text{MaxPicheight} \times 1$ When JpegeMiniBufMode is 1 Buffer depth $\geq 32 \times 1024$ bytes	Frame/Ecs	Baseline
H.265	Frame	When H265eMiniBufMode is 0 Buffer depth $\geq \text{MaxPicwidth} \times \text{MaxPicheight} \times 3/4$ When H265eMiniBufMode is 1 Buffer depth $\geq 32 \times 1024$ bytes	Frame/Slice	Main profile



- Recommended width and height of a VENC channel: 1920 x 1080 (1080p), 1280 x 720 (720p), 960 x 540, 640 x 360, 704 x 576, 704 x 480, 352 x 288, and 352 x 240.
- Encoder attributes are static attributes except the channel width (**u32PicWidth**) and channel height (**u32PicHeight**). After a VENC channel is created successfully, the static attributes cannot be modified unless the channel is destroyed and then recreated. For details about the precautions to be taken during configuration, see the description of [HI_MPI_VENC_SetChnAttr](#).
- For H.264/H.265 encoding, the VB of the encoding frames consists of **YHeaderSize**, **CHeaderSize**, **YSize**, **CSize**, **PmeSize**, **PmeInfoSize**, and **TmvSize**. The encoder calculates the VB size and allocates the memory based on the maximum width and height by default. When configuring the channel width and height, ensure that the size of each VB part calculated based on the channel width and height does not exceed that calculated based on the maximum width and height. For details about how to calculate the VB size, see section [6.2.16 "Calculation of the VB for Encoding Frames."](#)
- For JPEG/MJPEG encoding, the channel width and height must meet the following requirements: $u32PicWidth \times u32PicHeight \leq \text{MaxPicWidth} \times \text{MaxPicHeight}$.
- The RC mode (one of RC attributes) must be set first. The JPEG capture channel does not need to be configured with RC attributes. Other protocol channels (H.264, H.265, and MJPEG are supported) must be configured with RC attributes. The RC mode of the RC must match the protocol type of the encoder.
- During H.264 encoding, H.265 encoding, or MJPEG encoding the bit control modes CBR, VBR, and FIXQP are supported. For different protocols, the attribute variables in the same RC mode are the same. [Table 6-6](#) describes the public attributes of three RC modes.

Table 6-6 Public attributes of three RC modes

RC Mode	GOP	StatTime (s)	FrmRate (SrcFrmRate/DstFrmRate)
CBR	≥ 1	≥ 1	$\text{SrcFrmRate} \geq \text{DstFrmRate}$
VBR	≥ 1	≥ 1	$\text{SrcFrmRate} \geq \text{DstFrmRate}$
FixQp	≥ 1	≥ 1	$\text{SrcFrmRate} \geq \text{DstFrmRate}$
QpMap	≥ 1	≥ 1	$\text{SrcFrmRate} \geq \text{DstFrmRate}$

- **SrcFrmRate** must be set to the actual frame rate at which TimeRef is generated. The RC needs to calculate the actual frame rate and control the bit rate based on the value of **SrcFrmRate**. For details, see the **Note** field of [VENC_ATTR_H264_CBR_S](#).
- In addition to the attributes described in [Figure 6-2](#), the average bit rate (in kbit/s) and bit rate fluctuation level must be set for the CBR. The average bit rate relates to the width and height of a VENC channel and picture frame rate. [Table 6-7](#) describes the typical average bit rate. Assume that the full encoding frame rate is 25 fps. When the frame rate is not full (for example, 10 fps), you can divide the following bit rate by 2.5 (the product 25 divided by 10) to obtain the actual bit rate.

**Table 6-7** Typical configuration of the average bit rate

Width and Height of a Picture/Bit Rate Level	D1 (720 x 576)	720p (1280 x 720)	1080p (1920 x 1080)
Low bit rate	< 400 kbit/s	< 800 kbit/s	< 2000 kbit/s
Medium bit rate	400–1000 kbit/s	800–4000 kbit/s	2000–8000 kbit/s
High bit rate	> 1000 kbit/s	> 4000 kbit/s	> 8000 kbit/s

- There are five fluctuation levels for the bit rate. A higher level indicates a wider range of bit rate fluctuation. If the fluctuation level is high, the picture quality is more stable when the picture contents of adjacent frames vary significantly. The high level applies to the scenario in which the bandwidth margin is large. If the fluctuation level is low, the encoding bit rate is more stable. When the picture contents of adjacent frames vary significantly, the picture quality at a low level may be worse than that at a high level. The low level applies to the scenario in which the bandwidth margin is small. The function of setting the fluctuation level is reserved and not used currently.
- In VBR mode, besides the preceding attributes, you also need to set **MaxBitRate**, **MaxQp**, **MinQp**, and **u32MinIQp**. **MaxBitRate** indicates the maximum bit rate of a VENC channel within the bit rate statistics period; **MaxQp** indicates the maximum QP value allowed by the picture; **MinQp** indicates the minimum QP value allowed by the picture; **u32MinIQp** indicates the minimum QP value allowed by the I frame.
- In FixQp mode, besides the attributes described in [Figure 6-2](#), you also need to set **Iqp**, **PQp** and **BQp**. **Iqp** is the fixed QP value for pictures in the I frame. **PQp** is the fixed QP value for pictures in the P frame. **BQp** is the fixed QP value for pictures in the B frame. When setting the **Iqp** and **PQp**, you can increase or decrease them based on the current bandwidth. To reduce the respiratory effect, ensure that the **Iqp** is greater than the **PQp** by 2 or 3.
- In QpMap mode, besides the preceding attributes, you also need to set **enQpMapMode** as follows:
 - When the size of the CU is 32 x 32, the QP value for the CU is selected as follows:
 - **VENC_RC_QPMAP_MODE_MEANQP**: indicates the average QP value of four 16 x 16 blocks.
 - **VENC_RC_QPMAP_MODE_MINNQP**: indicates the minimum QP value of four 16 x 16 blocks.
 - **VENC_RC_QPMAP_MODE_MAXNQP**: indicates the maximum QP value of four 16 x 16 blocks.
 - When the size of the CU is 64 x 64, the QP value for the CU is selected as follows:
 - **VENC_RC_QPMAP_MODE_MEANQP**: indicates the average QP value of sixteen 16 x 16 blocks.
 - **VENC_RC_QPMAP_MODE_MINNQP**: indicates the minimum QP value of 16 x 16 blocks.
 - **VENC_RC_QPMAP_MODE_MAXQP**: indicates the maximum QP value of 16 x 16 blocks. Hi3519 V100 does not support this bit rate control mode.
- The GOP mode must first be configured for the GOP structure attribute of the H.264/H.265 channel (the GOP structure attribute does not need to be configured for the JPEG/MJPEG channel).

[Example]



```
HI_S32 StartVenc(HI_VOID)
{
    HI_S32 s32Ret;
    VI_CHN ViChn = 0;
    VENC_CHN VeChn = 0;
    VENC_CHN_ATTR_S stAttr;
    MPP_CHN_S stSrcChn, stDestChn;

    /* set h264 channel video encoding attribute */
    stAttr.stVeAttr.enType = PT_H264;
    stAttr.stVeAttr.stAttrH264e.u32PicWidth = u32PicWidth;
    stAttr.stVeAttr.stAttrH264e.u32PicHeight = u32PicHeigh;
    stAttr.stVeAttr.stAttrH264e.u32MaxPicWidth = u32MaxPicWidth;
    stAttr.stVeAttr.stAttrH264e.u32MaxPicHeight = u32MaxPicHeigh;
    stAttr.stVeAttr.stAttrH264e.u32Profile = 2;
    ..... // omit other video encode assignments here.

    /* set h264 channel rate control attribute */
    stAttr.stRcAttr.enRcMode = VENC_RC_MODE_H264CBR ;
    stAttr.stRcAttr.stAttrH264Cbr.u32BitRate = 10*1024;
    stAttr.stRcAttr.stAttrH264Cbr.fr32DstFrmRate = 30;
    stAttr.stRcAttr.stAttrH264Cbr.u32SrcFrmRate = 30;
    stAttr.stRcAttr.stAttrH264Cbr.u32Gop = 30;
    stAttr.stRcAttr.stAttrH264Cbr.u32FluctuateLevel = 1;
    stAttr.stRcAttr.stAttrH264Cbr.u32StatTime = 1;
    stAttr.stGopAttr.enGopMode = VENC_GOPMODE_NORMALP;
    stAttr.stGopAttr.stNormalP = 1;
    ..... // omit other rate control assignments here.

    s32Ret = HI_MPI_VENC_CreateChn(VeChn, &stAttr);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_CreateChn err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }
    s32Ret = HI_MPI_VENC_StartRecvPic(VeChn);
    if (s32Ret != HI_SUCCESS)
    {
        printf("HI_MPI_VENC_StartRecvPic err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }

    ..... // omit other code here.

    return HI_SUCCESS;
}.
```



[See Also]

None

HI_MPI_VENC_DestroyChn

[Description]

Destroys a VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_DestroyChn(VENC_CHN VeChn);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- If a channel that does not exist is reset, an error code indicating failure is returned.
- Before destroying a VENC channel, ensure that picture reception is stopped. Otherwise, an error code indicating failure is returned.
- If the on-screen display (OSD) function is enabled, you need to call HI_MPI_RGN_DetachFrmChn to detach the OSD region from the current encoding channel before destroying the encoding channel.

[Example]

```
HI_S32 StopVenc(HI_VOID)
{
    HI_S32 s32Ret;
    VENC_CHN VeChn = 0;

    s32Ret = HI_MPI_VENC_StopRecvPic(VeChn);
```



```
if (s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VENC_StopRecvPic err 0x%x\n", s32Ret);
    return HI_FAILURE;
}
s32Ret = HI_MPI_VENC_DestroyChn(VeChn);
if (s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VENC_DestroyChn err 0x%x\n", s32Ret);
    return HI_FAILURE;
}
return HI_SUCCESS;
}
```

[See Also]

None

HI_MPI_VENC_Resetchn

[Description]

Resets a VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_Resetchn(VENC_CHN VeChn);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]



- If a channel that does not exist is reset, the error code `HI_ERR_VENC_UNEXIST` is returned.
- If you reset a channel whose function of receiving picture is not stopped, an error code indicating failure is returned.
- The OSD is enabled in B frame mode. To destroy the OSD, call `HI_MPI_VENC_Resetchn` before calling `HI_MPI_RGN_Destroy`.

[Example]

```
HI_S32 ResetVenc(HI_VOID)
{
    HI_S32 s32Ret;
    VI_CHN ViChn = 0;
    VENC_CHN VeChn = 0;
    VENC_CHN_ATTR_S stAttr;
    MPP_CHN_S stSrcChn, stDestChn;
    /* set h264 channel video encode attribute */
    stAttr.stVeAttr.enType = PT_H264;
    stAttr.stVeAttr.stAttrH264e.u32PicWidth = u32PicWidth;
    stAttr.stVeAttr.stAttrH264e.u32PicHeight = u32PicHeigh;
    stAttr.stVeAttr.stAttrH264e.u32MaxPicWidth = u32MaxPicWidth;
    stAttr.stVeAttr.stAttrH264e.u32MaxPicHeight = u32MaxPicHeigh;
    stAttr.stVeAttr.stAttrH264e.u32Profile = 2;
    ..... // omit other video encode assignments here.
    /* set h264 channel rate control attribute */
    stAttr.stRcAttr.enRcMode = VENC_RC_MODE_H264CBR ;
    stAttr.stRcAttr.stAttrH264Cbr.u32BitRate = 10*1024;
    stAttr.stRcAttr.stAttrH264Cbr.fr32DstFrmRate = 30;
    stAttr.stRcAttr.stAttrH264Cbr.u32SrcFrmRate = 30;
    stAttr.stRcAttr.stAttrH264Cbr.u32Gop = 30;
    stAttr.stRcAttr.stAttrH264Cbr.u32FluctuateLevel = 1;
    stAttr.stRcAttr.stAttrH264Cbr.u32StatTime = 1;
    ..... // omit other rate control assignments here.

    s32Ret = HI_MPI_VENC_CreateChn(VeChn, &stAttr);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_CreateChn err 0x%x\n",s32Ret);
        return HI_FAILURE;
    }

    s32Ret = HI_MPI_VENC_StartRecvPic(VeChn);
    if (s32Ret != HI_SUCCESS)
    {
        printf("HI_MPI_VENC_StartRecvPic err 0x%x\n",s32Ret);
        return HI_FAILURE;
    }
}
```



```
}

s32Ret = HI_MPI_VENC_StopRecvPic(VeChn);
if (s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VENC_StopRecvPic err 0x%x\n", s32Ret);
    return HI_FAILURE;
}

s32Ret = HI_MPI_VENC_ReSetChn(VeChn);
if (s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VENC_ReSetChn err 0x%x\n", s32Ret);
    return HI_FAILURE;
}

..... // omit other code here.

return HI_SUCCESS;
}
```

[See Also]

None

HI_MPI_VENC_StartRecvPic

[Description]

Enables a VENC channel to start receiving pictures.

[Syntax]

```
HI_S32 HI_MPI_VENC_StartRecvPic(VENC_CHN VeChn);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."



[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- If a VENC channel is not created, error [HI_ERR_VENC_UNEXIST](#) is returned.
- This MPI does not check whether the function of receiving picture is enabled. That is, the function can be repeatedly enabled, and no error is returned.
- The encoder starts receiving pictures only after this function is enabled.

[Example]

See the example of [HI_MPI_VENC_CreateChn](#).

[See Also]

None

HI_MPI_VENC_StartRecvPicEx

[Description]

Enables a VENC channel to start receiving input pictures and automatically stop the VENC channel receiving input pictures after the specified number of frames is exceeded.

[Syntax]

```
HI_S32 HI_MPI_VENC_StartRecvPicEx(VENC_CHN VeChn, VENC_RECV_PIC_PARAM_S  
*pstRecvParam);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID. Value range: [0, VENC_MAX_CHN_NUM)	Input
pstRecvParam	Pointer to the received picture parameter structure. This pointer indicates the number of frames to be received.	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code.

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a



[Note]

- If the VENC channel is not created, [HI_ERR_VENC_UNEXIST](#) is returned.
- If the VENC channel is enabled to start receiving pictures without stop by calling [HI_MPI_VENC_StartRecvPic](#) or has not received sufficient pictures by calling [HI_MPI_VENC_StartRecvPicEx](#) last time, an error code is returned if [HI_MPI_VENC_StartRecvPicEx](#) is called again, indicating that this operation is prohibited.
- This MPI is used to receive and encode consecutive N frames. When N is **0**, [HI_MPI_VENC_StartRecvPicEx](#) is equivalent to [HI_MPI_VENC_StartRecvPic](#).
- If [HI_MPI_VENC_StartRecvPic](#) has been called to receive pictures before (picture reception is stopped) and you want to call [HI_MPI_VENC_StartRecvPicEx](#) to start encoding again, you are advised to call [HI_MPI_VENC_Resetchn](#) to clear the buffered pictures and streams before calling [HI_MPI_VENC_StartRecvPicEx](#).
- If a JPEG snapshot channel is created, you are advised to call [HI_MPI_VENC_StartRecvPicEx](#).

[Example]

```
HI_S32 JpegSnapProcess(HI_VOID)
{
    HI_S32 s32Ret;
    VENC_CHN VeChn = 0;
    VENC_CHN_ATTR_S stAttr;
    VENC_RECV_PIC_PARAM_S stRecvParam;
    /*Set the video encoding attributes of the JPEG channel. */
    stAttr.stVeAttr.enType = PT_JPEG;
    stAttr.stVeAttr.stAttrJpeg.u32PicWidth = u32PicWidth;
    stAttr.stVeAttr.stAttrJpeg.u32PicHeight = u32PicHeigh;
    stAttr.stVeAttr.stAttrJpeg.u32MaxPicWidth = u32MaxPicWidth;
    stAttr.stVeAttr.stAttrJpeg.u32MaxPicHeight = u32MaxPicHeigh;
    .... // omit other video encode assignments here.
    // Create a JPEG channel.
    s32Ret = HI_MPI_VENC_CreateChn(VeChn, &stAttr);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_CreateChn err 0x%x\n",s32Ret);
        return HI_FAILURE;
    }
    // start snapping
    stRecvParam.s32RecvPicNum = 2;
    s32Ret = HI_MPI_VENC_StartRecvPicEx(VeChn, &stRecvParam);
    if (s32Ret != HI_SUCCESS)
    {
        printf("HI_MPI_VENC_StartRecvPicEx err 0x%x\n",s32Ret);
        return HI_FAILURE;
    }
    .... // wait until all pictures have been encoded.
```



```
s32Ret = HI_MPI_VENC_StopRecvPic(VeChn);  
if (s32Ret != HI_SUCCESS)  
{  
    printf("HI_MPI_VENC_StopRecvPic err 0x%x\n", s32Ret);  
    return HI_FAILURE;  
}  
// Destroy the JPEG channel.  
s32Ret = HI_MPI_VENC_DestroyChn(VeChn);  
if (s32Ret != HI_SUCCESS)  
{  
    printf("HI_MPI_VENC_DestroyChn err 0x%x\n", s32Ret);  
    return HI_FAILURE;  
}  
return HI_SUCCESS;  
}
```

[See Also]

None

HI_MPI_VENC_StopRecvPic

[Description]

Stops a VENC channel receiving input pictures.

[Syntax]

```
HI_S32 HI_MPI_VENC_StopRecvPic(VENC_CHN VeChn);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpci.a



[Note]

- If a VENC channel is not created, an error code indicating failure is returned.
- This MPI does not check whether picture reception is stopped. That is, picture reception can be repeatedly stopped, and no error is returned.
- This MPI is used to stop a VENC channel receiving pictures. Before being destroyed or reset, the channel must stop receiving pictures.
- When this MPI is called, only receiving raw data is stopped but the stream buffer is not cleared.
- Picture reception that starts by calling [HI_MPI_VENC_StartRecvPic](#) or [HI_MPI_VENC_StartRecvPicEx](#) can be stopped by calling [HI_MPI_VENC_StopRecvPic](#).

[Example]

See the example of [HI_MPI_VENC_DestroyChn](#).

[See Also]

None

HI_MPI_VENC_Query

[Description]

Queries the status of a VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_Query(VENC_CHN VeChn, VENC_CHN_STAT_S *pstStat);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstStat	Pointer to the status of a VENC channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a



[Note]

- If a VENC channel is not created, an error code indicating failure is returned.
- This MPI is called to query the status of the encoder. The **pstStat** parameter contains the following information:
 - **u32LeftPics** indicates the number of frames to be encoded.
Before resetting a VENC channel, you can check whether there are pictures to be encoded, preventing the frames from being cleared during reset.
 - **u32LeftStreamBytes** indicates the number of remaining bytes in the stream buffer.
Before resetting a VENC channel, you can check whether there are streams to be processed, preventing the streams from being cleared during reset.
 - **u32LeftStreamFrames** indicates the number of remaining frames in the stream buffer.
Before resetting a VENC channel, you can check whether there are streams that are not obtained, preventing the streams from being cleared during reset.
 - **u32CurPacks** indicates the number of stream packets in the current frames. Before calling [HI_MPI_VENC_GetStream](#), ensure that **u32CurPacks** is greater than 0.
When streams are obtained by packet, the current frame may not be a complete frame (part of the frame is obtained). When streams are obtained by frame, this parameter indicates the number of packets in the current frame (if it is not a complete frame, this parameter is set to 0). When obtaining streams by frame, you need to check the number of stream packets of a complete frame. Typically, after the select function is successfully called, you can query the number of stream packets in the selected frame, which is specified by the **u32CurPacks** parameter.
 - **u32LeftRecvPics** indicates the number of frames to be received after [HI_MPI_VENC_StartRecvPicEx](#) is called.
 - **u32LeftEncPics** indicates the number of frames to be encoded after [HI_MPI_VENC_StartRecvPicEx](#) is called.
The number of frames to be received or encoded is 0 if [HI_MPI_VENC_StartRecvPicEx](#) is not called.

[Example]

See [HI_MPI_VENC_GetStream](#).

[See Also]

None

HI_MPI_VENC_SetChnAttr

[Description]

Sets attributes of a VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetChnAttr(VENC_CHN VeChn, const VENC_CHN_ATTR_S*  
pstAttr);
```

[Parameter]



Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstAttr	Pointer to the VENC channel attributes	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- The attributes of a VENC channel such as maximum width and maximum height cannot be dynamically changed.
- The attributes of a VENC channel include the encoder attributes and RC attributes.
- When you attempt to set attributes for a nonexistent VENC channel, an error code indicating failure is returned.
- If **pstAttr** is null, an error code indicating failure is returned.
- The limitations for setting the size of the pictures to be encoded are the same as those for creating a channel. For details, see [Table 6-5](#).
- A VENC channel has dynamic attributes and static attributes. The dynamic attributes are configured when the channel is created and can be modified before the channel is destroyed. The static attributes are configured when the channel is created and cannot be modified after the channel is created.
- This MPI is called to configure only dynamic attributes. If you attempt to configure static attributes by using this MPI, an error code indicating a failure is returned. The static attributes include encoding protocol, method of obtaining streams (streams are obtained by frame or by packet), and the maximum width and height of encoded pictures. Static attributes are specified by each protocol module. For details, see [**VENC_CHN_ATTR_S**](#).
- After the channel height and width in the encoder attributes are configured, all parameters for the encoding channel are restored to the default values except the channel priority, the OSD function is disabled, and the stream buffer and picture buffer queue are cleared.
- To reconfigure the stream resolution when the OSD function is enabled, perform the following steps:
 - Call [HI_MPI_VENC_StopRecvPic](#) to stop the encoding channel from receiving pictures.



2. Call `HI_MPI_RGN_DetachFrmChn` to detach the OSD region from the encoding channel.
 3. Call `HI_MPI_VENC_SetChnAttr` to reconfigure the encoding stream resolution.
 4. Call `HI_MPI_RGN_AttachToChn` to overlay the OSD region to the encoding channel.
 5. Call `HI_MPI_VENC_StartRecvPic` to start receiving pictures.
- When the encoding channel attribute is configured and **enGopMode** is not modified, the channel attribute immediately takes effect. When the **enGopMode** is modified, the channel attribute takes effect after the delay, and the maximum delay time is shown as follows (**VENC_GOPMODE_LOWDELAYB** is not supported and not shown in [Table 6-8](#).):

Table 6-8 Mapping between enGopMode and the maximum delay time

enGopMode Before Modification	enGopMode After Modification	Maximum Delay Time (Unit: Number of Encoded Frames)
VENC_GOPMODE_BIPREDB	Non-VENC_GOPMODE_BIPREDB	1 + u32BFrmNum
Non-VENC_GOPMODE_BIPREDB	Any mode that is different from enGopMode before modification	1

- When **enGopMode** is modified, the frame rate of the non-ROI, the attribute of refreshing the Islice in the P frame, frame discarding policy when the instantaneous bit rate of the encoding channel exceeds the threshold, frame skipping reference attribute (H.264/H.265), pic_order_cnt_type (H.264) are restored to the default values. To continue using these attributes, reconfigure them.

[Example]

None

[See Also]

None

HI_MPI_VENC_GetChnAttr

[Description]

Obtains attributes of a VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetChnAttr(VENC_CHN VeChn, VENC_CHN_ATTR_S*pstAttr);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID	Input



Parameter	Description	Input/Output
	Value range: [0, VENC_MAX_CHN_NUM)	
pstAttr	Pointer to the VENC channel attributes	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- When you attempt to obtain attributes of a nonexistent VENC channel, error [HI_ERR_VENC_UNEXIST](#) is returned.
- If **pstAttr** is null, an error code indicating failure is returned.
- During the switching of the GOP mode, the obtained channel attributes are the latest configured attributes.

[Example]

None

[See Also]

None

HI_MPI_VENC_GetStream

[Description]

Obtains encoded streams.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetStream(VENC_CHN VeChn, VENC_STREAM_S *pstStream,  
HI_S32 s32MilliSec);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input



Parameter	Description	Input/Output
pstStream	Pointer to the stream structure	Output
s32MilliSec	Timeout period for obtaining streams Value range: $[-1, +\infty)$ • -1 : block mode • 0 : non-block mode • > 0 : timeout period	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: `hi_comm_venc.h`, `mpi_venc.h`
- Library file: `libmpi.a`

[Note]

- If a VENC channel is not created, an error code indicating failure is returned.
- If `pstStream` is null, the error code `HI_ERR_VENC_NULL_PTR` is returned.
- If the `s32MilliSec` is smaller than -1 , the error code `HI_ERR_VENC_ILLEGAL_PARAM` is returned.
- Streams can be obtained in timeout mode. In addition, the select or poll function can be called.
 - If `s32MilliSec` is **0**, streams are obtained in non-block mode. To be specific, if there is no data in the buffer, `HI_ERR_VENC_BUF_EMPTY` is returned.
 - If `s32MilliSec` is **-1**, streams are obtained in block mode. To be specific, if there is no data in the buffer, the code indicating that streams are successfully obtained is returned only after there is data in the buffer.
 - If `s32MilliSec` is greater than **0**, streams are obtained in timeout mode. To be specific, if there is no data in the buffer, the code indicating that streams are successfully obtained is returned when there is data within the configured period. Otherwise, a code indicating failure due to timeout is returned.
- Streams can be obtained by packet or frame. When streams are obtained by packet:
 - For the H.264 protocol, an NAL unit is obtained each time this MPI is called.
 - For the JPEG protocol (Hi3519 V100 does not support the function of obtaining streams by packet), a parameter packet or data packet is obtained each time.
 - For the H.265 protocol, an NAL unit is obtained each time this MPI is called.
- The stream structure `VENC_STREAM_S` contains the following information:
 - Pointer to a stream packet `pstPack`



This parameter specifies the memory space of the **VENC_PACK_S**. The space is allocated by the caller. When streams are obtained by packet, the space is not smaller than the size specified by the **VENC_PACK_S**; when streams are obtained by frame, the space is not smaller than $n \times$ size specified by the **VENC_PACK_S**. Here, n indicates the number of stream packets in the current frame and can be obtained by calling the query MPI after the Select invocation.

- Number of stream packets **u32PackCount**

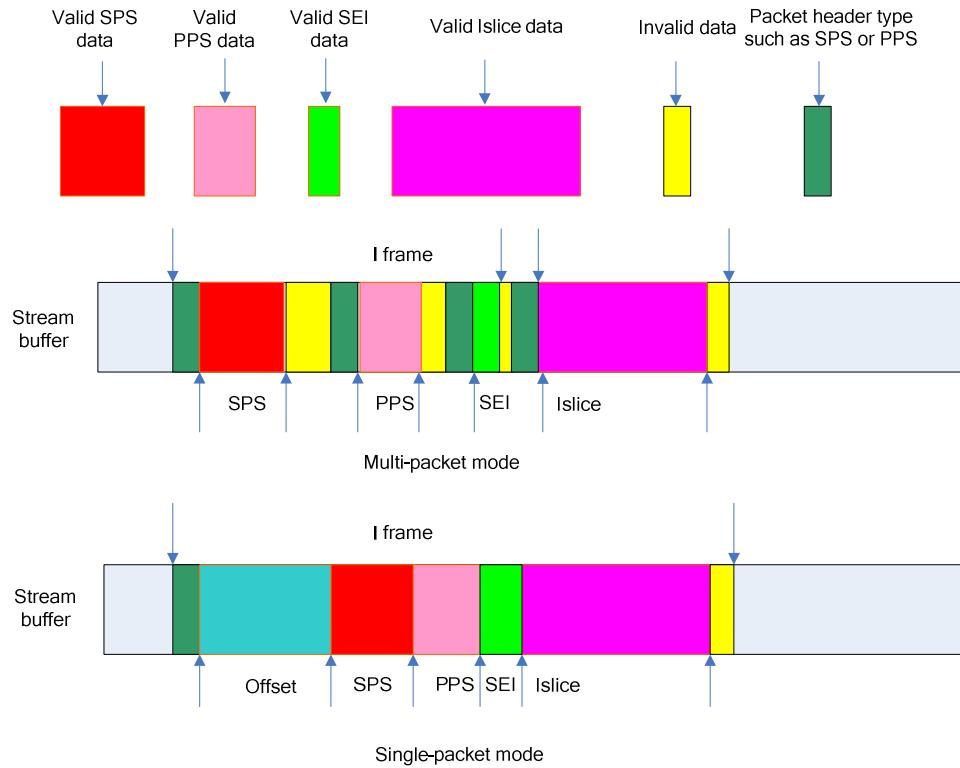
This parameter specifies the value of **VENC_PACK_S** in **pstPack**. When streams are obtained by packet, **u32PackCount** should not be smaller than 1; when streams are obtained by frame, **u32PackCount** should not be less than the number of packets in the current frame. After this MPI is successfully called, the value of **u32PackCount** is the number of packets specified in **pstPack**.

- Sequence number **u32Seq**

When streams are obtained by frame, it is the sequence number of a frame; when streams are obtained by packet, it is the sequence number of a packet.

- Stream features **stH624Info**, **stJpegInfo**, and **stH265Info**. The data structure is a union, including stream features corresponding to various encoding protocols. The output stream features are used for upper-layer applications of users.
- Advanced stream features **stAdvanceH264Info**, **stAdvanceJpegInfo**, and **stAdvanceH265Info**. The data structure is a union, including advanced stream features corresponding to various encoding protocols. The output advanced stream features are used for upper-layer applications of users.

- If streams are not fetched for a long period of time, the stream buffer becomes full. If the stream buffer corresponding to a VENC channel is full, encoding is not started until streams are fetched. Stream encoding starts only when the buffer for encoding is sufficient.
- You are advised to call **HI_MPI_VENC_GetStream** and **HI_MPI_VENC_ReleaseStream** in pairs and release the stream buffer as soon as possible. Otherwise, the stream buffer is full when streams are obtained in user mode. As a result, encoding stops.
- When **H264eOneStreamBuffer**, **H265eOneStreamBuffer**, or **JpegeOneStreamBuffer** is 1, if one frame of stream is obtained, the address for only one stream packet (excluding user data) is obtained. That is, **u32PackCount** is 1, and the address is **pstPack[0].pu8Addr.u32Offset** is added to **VENC_PACK_S** for specifying the address for the valid data in a frame stream and **pstPack[0].pu8Addr** offset.
- You are advised to obtain streams by calling the select function and perform the following steps:
 1. Call **HI_MPI_VENC_Query** to query the status of the encoding channel.
 2. Check the values of **u32CurPacks** and **u32LeftStreamFrames**, and ensure that they are greater than 0.
 3. Call the malloc function to allocate **u32CurPacks** packet information data structures.
 4. Call **HI_MPI_VENC_GetStream** to obtain the encoding stream.
 5. Call **HI_MPI_VENC_ReleaseStream** to release the stream buffer.
- The H.264 protocol is used as an example, as shown in [Figure 6-18](#). If **pstPack[0].u32DataNum = 3**, there are three types of NAL packets, including SPS, PPS, and SEI. **pstPack[0].stPackInfo[0].u32PackType.enH264EType = H264E_NALU_SPS**; **pstPack[0].stPackInfo[1].u32PackType.enH264EType = H264E_NALU_PPS**; **pstPack[0].stPackInfo[2].u32PackType.enH264EType = H264E_NALU_SEI**. The NAL packet types for other protocols are similar.

Figure 6-18 Stream packet structure

The configuration mode can be selected by setting the **H264eOneStreamBuffer** parameter when .ko drivers of the protocols are loaded.

- If **H264eOneStreamBuffer** is set to **1**, the single-packet mode is selected.
- If **H264eOneStreamBuffer** is set to **0**, the multi-packet mode is selected. The default value of **H264eOneStreamBuffer** is **0**.

Streams can be obtained in cache mode. The mode can be specified by configuring **VencBufferCache** when the VENC .ko files are loaded. If **VencBufferCache** is set to **1**, streams are obtained in cache mode. If **VencBufferCache** is set to **0**, streams are obtained in non-cache mode. **VencBufferCache** is set to **0** by default. The cache mode is not recommended if each frame is copied only once from the stream buffer.

[Example]

```
HI_S32 VencGetH264Stream(HI_VOID)
{
    HI_S32 i;
    HI_S32 s32Ret;
    HI_S32 s32VencFd;
    HI_U32 u32FrameIdx = 0;
    VENC_CHN VeChn = 0;
    VENC_CHN_STAT_S stStat;
    VENC_STREAM_S stStream;
    fd_set read_fds;
```



```
FILE *pFile = NULL;

pFile = fopen("stream.h264", "wb");
if(pFile == NULL)
{
    return HI_FAILURE;
}

s32VencFd = HI_MPI_VENC_GetFd(VeChn);
do{
    FD_ZERO(&read_fds);
    FD_SET(s32VencFd, &read_fds);

    s32Ret = select(s32VencFd+1, &read_fds, NULL, NULL, NULL);
    if (s32Ret < 0)
    {
        printf("select err\n");
        return HI_FAILURE;
    }
    else if (0 == s32Ret)
    {
        printf("time out\n");
        return HI_FAILURE;
    }
    else
    {
        if (FD_ISSET(s32VencFd, &read_fds))
        {
            s32Ret = HI_MPI_VENC_Query(VeChn, &stStat);
            if (s32Ret != HI_SUCCESS)
            {
                return HI_FAILURE;
            }
        }
    }
}

/*********************************************
suggest to check both u32CurPacks and u32LeftStreamFrames at
the same time,for example:
if(0 == stStat.u32CurPacks || 0 == stStat.u32LeftStreamFrames)
{
    continue;
}
*****************************************/
if(0 == stStat.u32CurPacks)
{
```



```
        continue;
    }

    stStream.pstPack = (VENC_PACK_S*)
        malloc(sizeof(VENC_PACK_S)*stStat.u32CurPacks);
    if (NULL == stStream.pstPack)
    {
        return HI_FAILURE;
    }

    stStream.u32PackCount = stStat.u32CurPacks;
    s32Ret = HI_MPI_VENC_GetStream(VeChn, &stStream, -1);
    if (HI_SUCCESS != s32Ret)
    {
        free(stStream.pstPack);
        stStream.pstPack = NULL;
        return HI_FAILURE;
    }

    for (i=0; i< stStream.u32PackCount; i++)
    {
        fwrite(stStream.pstPack[i].pu8Addr+
stStream.pstPack[i].u32Offset,
               1, stStream.pstPack[i].u32Len-
stStream.pstPack[i].u32Offset, pFile);

    }

    s32Ret = HI_MPI_VENC_ReleaseStream(VeChn, &stStream);
    if (HI_SUCCESS != s32Ret)
    {
        free(stStream.pstPack);
        stStream.pstPack = NULL;
        return HI_FAILURE;
    }

    free(stStream.pstPack);
    stStream.pstPack = NULL;
}

}

u32FrameIdx++;
}while (u32FrameIdx < 0xff);

fclose(pFile);
return HI_SUCCESS;
```



}

For details, see the related sample codes.

[See Also]

None

HI_MPI_VENC_ReleaseStream

[Description]

Releases a stream buffer.

[Syntax]

```
HI_S32 HI_MPI_VENC_ReleaseStream(VENC_CHN VeChn, VENC_STREAM_S  
*pstStream);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstStream	Pointer to the stream structure	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- If a VENC channel is not created, error [HI_ERR_VENC_UNEXIST](#) is returned.
- If **pstStream** is null, error [HI_ERR_VENC_NULL_PTR](#) is returned.
- This MPI must work with the [HI_MPI_VENC_GetStream](#) MPI. You are advised to release the obtained stream buffer. Otherwise, the stream buffer may be full, affecting encoding. In addition, you must comply with the principle of "first obtained, first released".
- When a VENC channel is reset, all stream packets that are not released are invalid and the stream buffer cannot be used or released any more.



- When you release invalid streams, error `HI_ERR_VENC_ILLEGAL_PARAM` is returned.

[Example]

See [HI_MPI_VENC_GetStream](#).

[See Also]

None

HI_MPI_VENC_GetStreamBufInfo

[Description]

Obtains the physical address and size of a stream buffer.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetStreamBufInfo(VENC_CHN VeChn,  
VENC_STREAM_BUF_INFO_S *pstStreamBufInfo)
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstStreamBufInfo	Pointer to stream buffer information	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: `hi_comm_venc.h`, `mpi_venc.h`
- Library file: `libmpi.a`

[Note]

- If no VENC channel is created, `HI_ERR_VENC_UNEXIST` is returned.
- If `pstStreamBufInfo` is null, `HI_ERR_VENC_NULL_PTR` is returned.
- If you want to use the physical address for a frame stream, you need to obtain the start physical address and size of the stream buffer by calling this MPI for checking whether the stream is wrapped. If the following condition is met, the frame stream is wrapped and two physical addresses are obtained: `pstPack[0].u32PhyAddr + u32Len > pstStreamBufInfo->u32PhyAddr + pstStreamBufInfo->u32BufSize`. This ensures



that the correct physical address is used. **pstPack[0]** is the variable of **VENC_STREAM_S**, and **u32Len** indicates the length of a frame stream. See [0](#).

[Example]

```
HI_S32 VencGetH264StreamBufInfo(HI_VOID)
{
    HI_S32 i;
    HI_S32 s32Ret;
    HI_S32 s32VencFd;
    HI_U32 u32FrameIdx = 0;
    VENC_CHN VeChn = 0;
    VENC_CHN_STAT_S stStat;
    VENC_STREAM_S stStream;
    fd_set read_fds;
    VENC_STREAM_BUF_INFO_S stStreamBufInfo;
    HI_U32 u32Left;
    HI_U32 u32SrcPhyAddr, u32DestPhyAddr;

    s32Ret = HI_MPI_VENC_GetStreamBufInfo (VeChn, &stStreamBufInfo);
    if (HI_SUCCESS != s32Ret)
    {
        return HI_FAILURE;
    }

    do{
        s32Ret = HI_MPI_VENC_Query(VeChn, &stStat);
        if (s32Ret != HI_SUCCESS)
        {
            return HI_FAILURE;
        }

        stStream.pstPack = (VENC_PACK_S*)
            malloc(sizeof(VENC_PACK_S)*stStat.u32CurPacks);
        if (NULL == stStream.pstPack)
        {
            return HI_FAILURE;
        }

        stStream.u32PackCount = stStat.u32CurPacks;
        s32Ret = HI_MPI_VENC_GetStream(VeChn, &stStream, -1);
        if (HI_SUCCESS != s32Ret)
        {
            free(stStream.pstPack);
            stStream.pstPack = NULL;
        }
    }while(0);
}
```



```
        return HI_FAILURE;
    }

//FUNC transfer, use only the physical address
for (i=0; i< stStream.u32PackCount; i++)
{
    if(stStream.pstPack[i].u32PhyAddr+
    stStream.pstPack[i].u32Len >=
    stStreamBufInfo.u32PhyAddr +
    stStreamBufInfo.u32BufSize)
    {
        if(stStream.pstPack[i].u32PhyAddr+
        stStream.pstPack[i].u32Offset >=
        stStreamBufInfo.u32PhyAddr +
        stStreamBufInfo.u32BufSize)
        {
            // (a) branch in picture
            u32SrcPhyAddr = stStreamBufInfo.u32PhyAddr+
            ((stStream.pstPack[i].u32PhyAddr+
            stStream.pstPack[i].u32Offset) -
            (stStreamBufInfo.u32PhyAddr +
            stStreamBufInfo. u32BufSize));

            FUNC_Transfer(u32SrcPhyAddr, u32DestPhyAddr,
            stStream.pstPack[i].u32Len-
            stStream.pstPack[i].u32Offset);
        }
        else
        {
            // (b) branch in picture
            u32Left = (stStreamBufInfo.u32PhyAddr +
            stStreamBufInfo.u32BufSize) -
            stStream.pstPack[i].u32PhyAddr;

            FUNC_Transfer(stStream.pstPack[i].u32PhyAddr +
            stStream.pstPack[i].u32Offset,
            u32DestPhyAddr,u32Left -
            stStream.pstPack[i].u32Offset);

            FUNC_Transfer(stStreamBufInfo.u32PhyAddr,
            u32DestPhyAddr, stStream.pstPack[i].u32Len-
            u32Left);
        }
    }
}
```



```
    else
    {
        // (c) branch in picture
        FUNC_Transfer (stStream.pstPack[i].u32PhyAddr +
                      stStream.pstPack[i].u32Offset,
                      stStream.pstPack[i].u32Len-
                      stStream.pstPack[i].u32Offset);
    }
}

s32Ret = HI_MPI_VENC_ReleaseStream(VeChn, &stStream);
if (HI_SUCCESS != s32Ret)
{
    free(stStream.pstPack);
    stStream.pstPack = NULL;
    return HI_FAILURE;
}

free(stStream.pstPack);
stStream.pstPack = NULL;

u32FrameIdx++;
}while (u32FrameIdx < 0xff);
return HI_SUCCESS;
}
```

NOTE

FUNC_Transfer() in the preceding example is a dummy function.

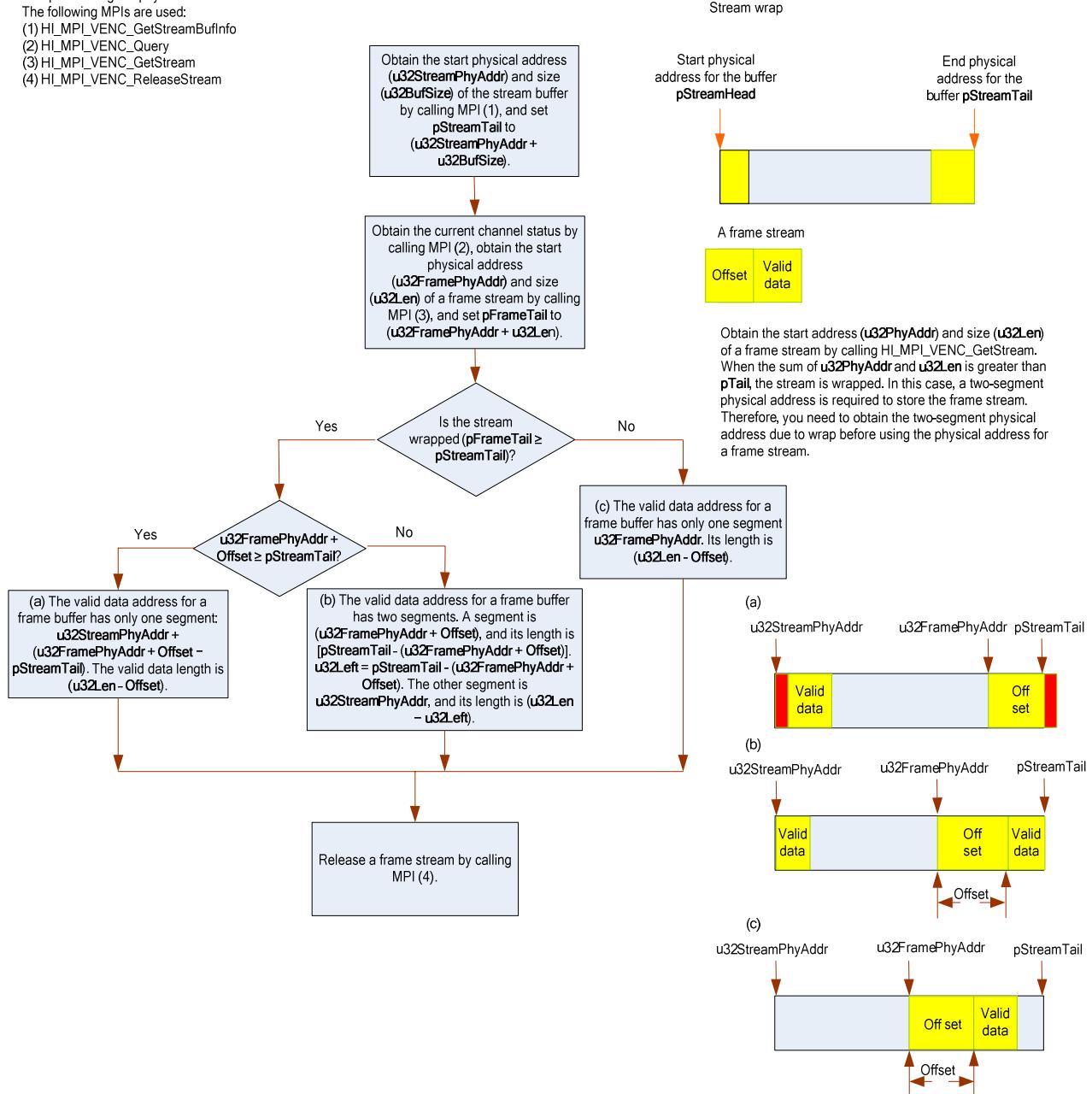


Figure 6-19 Checking whether the stream buffer is wrapped

Sample of using the physical address for a stream buffer.

The following MPIs are used:

- (1) HI_MPI_VENC_GetStreamBufInfo
- (2) HI_MPI_VENC_Query
- (3) HI_MPI_VENC_GetStream
- (4) HI_MPI_VENC_ReleaseStream



[See Also]

None

HI_MPI_VENC_InsertUserData

[Description]

Inserts user data.



[Syntax]

```
HI_S32 HI_MPI_VENC_InsertUserData(VENC_CHN VeChn, HI_U8 *pu8Data, HI_U32 u32Len);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pu8Data	Pointer to user data	Input
u32Len	User data length, in byte Value range: (0, 1024]	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- If a VENC channel is not created, an error code indicating failure is returned.
- If **pu8Data** is null, an error code indicating failure is returned.
- User data can be inserted by following the H.264/H.265 and MJPEG/JPEG protocols.
- For the H.264/H.265 channel, at most four memory spaces are provided for buffering user data, and the size of each data segment is not greater than 1 KB. If more than four data segments are inserted or a user data segment is greater than 1 KB, an error is returned. Each user data segment is inserted as an SEI packet before the latest picture stream packet. After a user data segment is encoded and sent, the memory space in the H.264 channel for buffering the user data is cleared for storing new user data.
- For the JPEG/MJPEG channel, at most one memory space is provided for buffering 1 KB user data. The user data is inserted into picture streams as APP segment (0xFFEF). After user data is encoded and sent, the memory space in the JPEG/MJPEG channel for buffering the user data is cleared for storing new user data.

[Example]

```
HI_U8 au8UserData[] = "hisilicon2011";
s32Ret = HI_MPI_VENC_InsertUserData(VeChn, au8UserData,
                                     sizeof(au8UserData));
```



```
if(HI_SUCCESS != s32Ret)
{
    printf("HI_MPI_VENC_InsertUserData err 0x%xn",s32Ret);
}
```

[See Also]

None

HI_MPI_VENC_SendFrame

[Description]

Sends raw pictures for encoding.

[Syntax]

```
HI_S32 HI_MPI_VENC_SendFrame(VENC_CHN VeChn, VIDEO_FRAME_INFO_S *pstFrame,
HI_S32 s32MilliSec)
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstFrame	Pointer to a raw picture structure	Input
s32MilliSec	Timeout period for sending pictures Value range: [-1, +∞) • -1: block mode • 0: non-block mode • > 0: timeout period	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI allows you to send pictures to a VENC channel.



- If the **s32MilliSec** is smaller than **-1**, the error code **HI_ERR_VENC_ILLEGAL_PARAM** is returned.
- The source pictures must be semi-planar YUV4:2:0 or semi-planar YUV4:2:2 or **PIXEL_FORMAT_YUV_400** pictures. The H.264/H.265 VENC channel receives semi-planar420 or **PIXEL_FORMAT_YUV_400** pictures, and the JPEG/MJPEG VENC channel receives semi-planar420, semi-planar422, or **PIXEL_FORMAT_YUV_400** pictures.
- The source picture size must be greater than or equal to the VENC channel size.
- Before calling this MPI to send pictures, ensure that the encoding channel is created and enabled to receive input pictures.

[Example]

For details, see the related sample codes.

[See Also]

None

HI_MPI_VENC_SendFrameEx

[Description]

Allows the user to send the raw picture and encode the picture based on the picture information in the QpMap table.

[Syntax]

```
HI_S32 HI_MPI_VENC_SendFrameEx(VENC_CHN VeChn, USER_FRAME_INFO_S  
*pstFrame, HI_S32 s32MilliSec)
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstFrame	Pointer to the structure of the raw picture information	Input
s32MilliSec	Timeout period for sending pictures Value range: [-1, +∞) • -1: block mode • 0: non-block mode > 0: timeout period	Input

[Return Value]

Return Value	Description
0	Success



Return Value	Description
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI allows you to send pictures to the VENC channel.
- If **s32MilliSec** is less than **-1**, the error code [HI_ERR_VENC_ILLEGAL_PARAM](#) is returned.
- Raw pictures to be sent must in the format of semi-planar YUV4:2:0 or PIXEL_FORMAT_YUV_400. The H.264/H.265 encoding channel receives semi-planar 420/PIXEL_FORMAT_YUV_400 pictures.
- The source picture size must be greater than or equal to the VENC channel size.
- Before calling this MPI to send pictures, ensure that the VENC channel is created and enabled to receive the input pictures. If the bit rate control mode is CBR or VBR, the user needs to call [HI_MPI_VENC_SetRcParam](#) to enable the QpMap function. Note that only the relative mode is supported.

[Example]

None

[See Also]

None

HI_MPI_VENC_SetMaxStreamCnt

[Description]

Sets the maximum number of frames in a stream buffer.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetMaxStreamCnt(VENC_CHN VeChn, HI_U32 u32MaxStrmCnt);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
u32MaxStrmCnt	Maximum number of frames in a stream buffer	Input

[Return Value]



Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This API is used to set the maximum number of frames in a stream buffer.
- If the number of stream frames in the buffer reaches the maximum value, the current pictures to be encoded are not encoded because the stream buffer is full.
- If the number of stream frames in the buffer reaches the maximum number of stream frames, the picture to be encoded is discarded.
- By default, the maximum number of stream frames is set to **200** by the system when a VENC channel is created.
- This MPI must be called after a VENC channel is created and before the channel is destroyed. This MPI takes effect before a frame is encoded and can be called repeatedly. You are advised to set the maximum number of stream frames after a VENC channel is created and before frames are encoded. Dynamic settings are not recommended.

[Example]

None

[See Also]

None

HI_MPI_VENC_GetMaxStreamCnt

[Description]

Obtains the maximum number of frames in a stream buffer.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetMaxStreamCnt(VENC_CHN VeChn, HI_U32  
*pu32MaxStrmCnt);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pu32MaxStrmCnt	Pointer to the maximum number of frames in a stream buffer	Output



[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

None

[Example]

None

[See Also]

None

HI_MPI_VENC_RequestIDR

[Description]

Requests IDR frames.

[Syntax]

```
HI_S32 HI_MPI_VENC_RequestIDR(VENC_CHN VeChn, HI_BOOL bInstant);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
bInstant	IDR frame instant encoding enable	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."



[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- If a VENC channel is not created, an error code indicating failure is returned.
- After the request for the IDR frame is received, if **bInstant** is **0**, the IDR frame is encoded at the current frame rate after the current frame is encoded; if **bInstant** is **1**, the IDR frame is encoded immediately regardless of the frame rate.
- An IDR frame request request can be sent in compliance with the H.264/H.265 protocol.
- This MPI is not affected by frame rate control, and an IDR frame is encoded each time this MPI is called. The frame rate and bit rate are unstable if this MPI is frequently called.
- When the GOP mode is SmartP mode or B frame mode, the request for the IDR frame takes effect after a delay.

[Example]

```
HI_S32 s32Ret;
VENC_CHN VeChn = 0;
s32Ret = HI_MPI_VENC_RequestIDR(VeChn, HI_TRUE);
if(HI_SUCCESS != s32Ret)
{
    printf("HI_MPI_VENC_RequestIDR err 0x%x\n", s32Ret);
}
```

[See Also]

None

HI_MPI_VENC_EnableIDR

[Description]

Enables an IDR frame.

[Syntax]

```
HI_S32 HI_MPI_VENC_EnableIDR(VENC_CHN VeChn, HI_BOOL bEnableIDR);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
bEnableIDR	Whether the IDR frame is enabled	Input

[Return Value]



Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- If no channel is created, an error code indicating failure is returned.
- If the IDR frame is disabled, no IDR frame or I frame can be encoded in the next frame unless the IDR frame is enabled.
- If the IDR frame is not enabled, a request for the IDR frame is made and takes effect.
- If the GOP mode is set to **VENC_GOPMODE_SMARTP**, only the IDR frame can be enabled.
- This MPI supports only the H.264/H.265 encoding protocol.

[Example]

```
HI_S32 s32Ret;
VENC_CHN VeChn = 0;
s32Ret = HI_MPI_VENC_EnableIDR(VeChn);
if(HI_SUCCESS != s32Ret)
{
    printf("HI_MPI_VENC_EnableIDR err 0x%xn", s32Ret);
}
```

[See Also]

None

HI_MPI_VENC_SetH264IdrPicId

[Description]

Sets the idr_pic_id of an IDR frame.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetH264IdrPicId(VENC_CHN VeChn,
VENC_H264_IDRPICID_CFG_S* pstH264eIdrPicIdCfg);
```

[Parameter]



Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstH264eIdrPicIdCfg	Pointer to the idr_pic_id	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- If no channel is created, an error code indicating failure is returned.
- The idr_pic_id depends on the following two parameters:
 - enH264eIdrPicIdMode:** This parameter is used to set the mode of setting the idr_pic_id of the IDR frame. If **enH264eIdrPicIdMode** is **H264E_IDR_PICTURE_ID_MODE_AUTO**, the idr_pic_id is automatically generated in the internal encoding process. If **enH264eIdrPicIdMode** is **H264E_IDR_PICTURE_ID_MODE_USR**, the idr_pic_id is user-defined.
 - u32H264eIdrPicId:** This parameter is used to specify the idr_pic_id. **u32H264eIdrPicId** is valid only when **enH264eIdrPicIdMode** is **H264E_IDR_PICTURE_ID_MODE_USR**.
- This MPI supports only the H.264 encoding protocol.
- This MPI can be called after a VENC channel is created or before a VENC channel is destroyed. The configured idr_pic_id takes effect before the next frame is encoded. In addition, this MPI can be repeatedly called. You are advised to set the idr_pic_id after a VENC channel is successfully created and before encoding starts. That is, it is not recommended that the idr_pic_id be dynamically changed during encoding.

[Example]

None

[See Also]

None

HI_MPI_VENC_GetH264IdrPicId

[Description]



Obtains the idr_pic_id of an IDR frame.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetH264IdrPicId(VENC_CHN VeChn,  
VENC_H264_IDRPICID_CFG_S* pstH264eIdrPicIdCfg);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstH264eIdrPicIdCfg	Pointer to the idr_pic_id	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- If no channel is created, an error code indicating failure is returned.
- This MPI can be called after a VENC channel is created or before a VENC channel is destroyed.

[Example]

None

[See Also]

None

HI_MPI_VENC_GetFd

[Description]

Obtains the device file handle of a VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetFd(VENC_CHN VeChn);
```

[Parameter]



Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input

[Return Value]

Return Value	Description
A positive value	Valid
A non-positive value	Invalid

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

None

[Example]

None

[See Also]

None

HI_MPI_VENC_CloseFd

[Description]

Closes the device file handle of a VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_CloseFd(VENC_CHN VeChn);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input

[Return Value]

Return Value	Description
0	Success



Return Value	Description
-1	Failure

[Error Code]

None

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

None

[Example]

None

[See Also]

None

HI_MPI_VENC_SetRoiCfg

[Description]

Sets the ROI attributes of an H.264/H.265 channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetRoiCfg (VENC_CHN VeChn, VENC_ROI_CFG_S  
*pstVencRoiCfg);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstVencRoiCfg	ROI attributes of the H.264 streams	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."



[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to set the ROI attribute of an H.264/H.265 channel.
- The ROI parameters include:
 - **u32Index**: The system can be configured with eight ROIs for each H.264 channel, internally numbered 0 to 7 for management. The value of this parameter indicates the index number of an ROI set by users. ROIs can be overlaid. When ROIs are overlaid, the priority of the ROIs increases with the index number.
 - **bEnable**: Whether the current ROI is enabled.
 - **bAbsQp**: Whether the current ROI uses an absolute QP or a relative QP.
 - **s32Qp**: When **bAbsQp** is set to **true**, this parameter indicates the QP value of all macroblocks in an ROI; when **bAbsQp** is set to **false**, this parameter indicates the relative QP value of all macroblocks in an ROI.
 - **stRect**: Specifies the coordinate position and size of the current ROI. The initial position along the horizontal and vertical axes must be within a picture and a multiple of 16. The height and width of the ROI must be a multiple of 16. The ROI must be within a picture.
- This MPI is an enhanced MPI. By default, the ROI function is not enabled. You must call this MPI to enable the ROI function.
- This MPI can be set after a VENC channel is created and before the channel is destroyed. This MPI takes effect after a next frame is encoded.
- You are advised to call this MPI after a VENC channel is created and before frames are encoded, to minimize times of calling this MPI during frame encoding.
- You are advised to call the [HI_MPI_VENC_GetRoiCfg](#) MPI to obtain the ROI configuration of the current channel before calling this MPI.
- After this MPI is called to set the ROI attributes, if the current frame is checked to be encoded as a Pskip frame, the effect of the Pskip frame takes priority.

[Example]

```
HI_S32 SetRoi(HI_VOID)
{
    HI_S32 s32Ret = HI_FAILURE;
    VENC_ROI_CFG_S     stRoiCfg;
    VENC_CHN_ATTR_S   stChnAttr;
    HI_S32 index = 0;
    VENC_CHN VeChnId = 0;
    //...omit other thing
    s32Ret = HI_MPI_VENC_GetChnAttr(VeChnId, &stChnAttr);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_GetChnAttr err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }
}
```



```
s32Ret = HI_MPI_VENC_GetRoiCfg(VeChnId, index, &stRoiCfg);  
if (HI_SUCCESS != s32Ret)  
{  
    printf("HI_MPI_VENC_GetRoiCfg err 0x%x\n", s32Ret);  
    return HI_FAILURE;  
}  
  
stSrcVencRoiCfg.bEnable          = HI_TRUE;  
stSrcVencRoiCfg.bAbsQp          = HI_TRUE;  
stSrcVencRoiCfg.s32Qp           = 10;  
stSrcVencRoiCfg.stRect.s32X     = 16;  
stSrcVencRoiCfg.stRect.s32Y     = 16;  
stSrcVencRoiCfg.stRect.u32Width = 16;  
stSrcVencRoiCfg.stRect.u32Height = 16;  
stSrcVencRoiCfg.u32Index        = 0;  
s32Ret = HI_MPI_VENC_SetRoiCfg(VeChnId, &stSrcVencRoiCfg);  
if (HI_SUCCESS != s32Ret)  
{  
    printf("HI_MPI_VENC_SetRoiCfg err 0x%x\n", s32Ret);  
    return HI_FAILURE;  
}  
return HI_SUCCESS;  
}
```

[See Also]

None

HI_MPI_VENC_GetRoiCfg

[Description]

Obtains the ROI attribute of an H.264/H.265 channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetRoiCfg(VENC_CHN VeChn, HI_U32 u32Index,  
VENC_ROI_CFG_S *pstVencRoiCfg);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
u32Index	Index of an ROI on an H.264/H.265 channel	Input
pstVencRoiCfg	Configuration of the ROI	Output



[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to obtain the index ROI attribute of an H.264/H.265 channel.
- This MPI must be called after a VENC channel is created and before the channel is destroyed.
- You are advised to call this MPI after a VENC channel is created and before frames are encoded, to minimize times of calling this MPI during frame encoding.

[Example]

See [HI_MPI_VENC_SetRoiCfg](#).

[See Also]

None

HI_MPI_VENC_SetRoiBgFrameRate

[Description]

Sets the frame rate attribute of non-ROIs in an H.264/H.265 VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetRoiBgFrameRate(VENC_CHN VeChn, const  
VENC_ROIBG_FRAME_RATE_S *pstRoiBgFrmRate);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstRoiBgFrmRate	Frame rate control for non-ROIs	Input

[Return Value]



Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to set the frame rate control parameters for non-ROIs in an H.264/H.265 VENC channel.
- Before calling this MPI, you must enable ROIs.
- This MPI is an advanced interface. The frame rate attributes of non-ROIs are not enabled by default. If you want to use these attributes, you need to enable them by calling this MPI.
- This MPI can be called after a VENC channel is created and before the channel is destroyed. If the MPI is called during encoding, the operation takes effect when the next frame is encoded.
- When the frame rate control attribute of the non-ROI is configured, ensure that the input frame rate (**SrcFrmRate**) is greater than the output frame rate (**DstFrmRate**) and **DstFrmRate** is greater than or equal to 0, or ensure that both **SrcFrmRate** and **DstFrmRate** are set to **-1**.
- You are advised to call this MPI after creating a channel but before starting encoding. This reduces the number of times of calling this MPI during encoding.
- Before calling this MPI, you are advised to obtain the frame rate of the non-ROIs in the current VENC channel by calling [HI_MPI_VENC_GetRoiBgFrameRate](#).
- When the frame rate of the non-ROI is lower than that of the ROI, you are advised not to use the frame skipping reference mode or SVC-T encoding. In frame skipping reference mode or SVC-T encoding mode, if the frame rate of the non-ROI is lower than that of the ROI, the target frame rate of the non-ROI may be greater than the configured target frame rate because the non-ROI at the base layer cannot be encoded as Pskip blocks.
- This MPI is valid only for the VENC channels of which the GOP mode is set to **VENC_GOPMODE_NORMALP**.
- This MPI becomes invalid after the GOP mode is switched. To use this MPI after mode switching, reconfigure it. After the GOP mode is switched from non-NORMALP to NORMALP, calling this MPI takes effect after the GOP takes effect.
- After this MPI is called to set the ROI attributes, if the current frame is checked to be encoded as a Pskip frame, the effect of the Pskip frame takes priority. If the OSD is configured, the OSD effect takes priority for the frames with OSD update.
- When the frame rate of the non-ROI is lower than that of the ROI in the H.265 channel, SAO filtering is disabled in the module. Therefore, you are advised to call [HI_MPI_VENC_SetH265Sao](#) to disable the SAO function.

[Example]

```
HI_S32 SetRoiFrameRate(HI_VOID)
```



```
{  
    HI_S32 s32Ret = HI_FAILURE;  
    VENC_ROI_CFG_S     stRoiCfg;  
    VENC_CHN_ATTR_S   stChnAttr;  
    VENC_ROIBG_FRAME_RATE_S stRoiBgFrameRate;  
    HI_S32 index = 0;  
    VENC_CHN VeChnId = 0;  
  
    //...omit other thing  
    s32Ret = HI_MPI_VENC_GetChnAttr(VeChnId, &stChnAttr);  
    if (HI_SUCCESS != s32Ret)  
    {  
        printf("HI_MPI_VENC_GetChnAttr err 0x%x\n", s32Ret);  
        return HI_FAILURE;  
    }  
  
    s32Ret = HI_MPI_VENC_GetRoiCfg(VeChnId, index, &stRoiCfg);  
    if (HI_SUCCESS != s32Ret)  
    {  
        printf("HI_MPI_VENC_GetRoiCfg err 0x%x\n", s32Ret);  
        return HI_FAILURE;  
    }  
    stSrcVencRoiCfg.bEnable      = HI_TRUE;  
    stSrcVencRoiCfg.bAbsQp       = HI_TRUE;  
    stSrcVencRoiCfg.s32Qp       = 10;  
    stSrcVencRoiCfg.stRect.s32X  = 16;  
    stSrcVencRoiCfg.stRect.s32Y  = 16;  
    stSrcVencRoiCfg.stRect.u32Width = 16;  
    stSrcVencRoiCfg.stRect.u32Height = 16;  
    stSrcVencRoiCfg.u32Index    = 0;  
    s32Ret = HI_MPI_VENC_SetRoiCfg(VeChnId, &stSrcVencRoiCfg);  
    if (HI_SUCCESS != s32Ret)  
    {  
        printf("HI_MPI_VENC_SetRoiCfg err 0x%x\n", s32Ret);  
        return HI_FAILURE;  
    }  
    stRoiBgFrameRate.s32SrcFrmRate = 30;  
    stRoiBgFrameRate.s32DstFrmRate = 15;  
    s32Ret = HI_MPI_VENC_SetRoiBgFrameRate (VeChnId, &stRoiBgFrameRate);  
    if (HI_SUCCESS != s32Ret)  
    {  
        printf("HI_MPI_VENC_SetRoiBgFrameRate err 0x%x\n", s32Ret);  
        return HI_FAILURE;  
    }  
}
```



```
    return HI_SUCCESS;  
}
```

[See Also]

None

HI_MPI_VENC_GetRoiBgFrameRate

[Description]

Obtains the frame rate attribute of non-ROIs in an H.264/H.265 VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetRoiBgFrameRate(VENC_CHN VeChn,  
VENC_ROIBG_FRAME_RATE_S *pstRoiBgFrmRate);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstRoiBgFrmRate	Frame rate of non-ROIs	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI can be called after a VENC channel is created and before the channel is destroyed.
- This MPI is valid only for the VENC channels of which the GOP mode is set to **VENC_GOPMODE_NORMALP**.
- You are advised to call this MPI after creating a channel but before starting encoding. This reduces the number of times of calling this MPI during encoding.
- During the switching of the GOP mode, the obtained attributes are the latest configured attributes.

[Example]



See the example of [HI_MPI_VENC_SetRoiBgFrameRate](#).

[See Also]

None

HI_MPI_VENC_SetH264SliceSplit

[Description]

Sets the slice attribute of an H.264 channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetH264SliceSplit(VENC_CHN VeChn, const  
VENC_PARAM_H264_SLICE_SPLIT_S * pstSliceSplit);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstSliceSplit	Slice mode of an H.264 stream	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to set the slice attribute of an H.264 channel.
- The slice parameters include:
 - bSplitEnable:** Whether the current frame is sliced.
 - u32SplitMode:** indicates a slice mode. When this parameter is set to **0**, the frame is sliced by byte; when this parameter is set to **1**, the frame is sliced by macroblock row. This parameter cannot be set to be greater than 1.
This parameter is effective only when **bSplitEnable** is set to **true**.
 - u32SliceSize:** indicates the size of a slice. The meaning of this parameter varies with the value of **u32SplitMode**. When **u32SplitMode** is set to **0**, this parameter indicates the average number of byte in each slice. The encoder encodes a picture from the top down and from the left right in the raster order by macroblock. Each slice is



segmented based on the value of this parameter. The actual size of a slice, however, may not be strictly consistent with the value set by users. An error may exist and the offset is always positive. This is because when the last slice is encoded, the remaining byte of the macroblocks are regarded as a complete slice though the number of bits is less than the value of this parameter. When **u32SplitMode** is set to **1**, this parameter indicates the number of macroblock rows occupied by a slice. The unit of this parameter is row of macroblocks. The remaining rows of macroblocks are regarded as a slice though the number of rows of macroblocks is less than the value of this parameter.

- This MPI is an enhanced interface. You can call it as required. However, you are advised not to call this MPI. **bSplitEnable** is set to **false by default**.
- This MPI can be set after a VENC channel is created and before the channel is destroyed. This MPI takes effect after a next frame is encoded.
- You are advised to call this MPI after a VENC channel is created and before frames are encoded, to minimize times of calling this MPI during frame encoding.
- You are advised to call [HI_MPI_VENC_GetH264SliceSplit](#) to obtain the slice configuration of the current channel before calling this MPI.

[Example]

```
HI_S32 SetsliceSplit(HI_VOID)
{
    HI_S32 s32Ret = HI_FAILURE;
    VENC_PARAM_H264_SLICE_SPLIT_S stSlice;
    VENC_CHN VeChnId = 0;

    //...omit other thing

    s32Ret = HI_MPI_VENC_GetH264SliceSplit(VeChnId, &stSlice);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_GetH264SliceSplit err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }

    stSlice.bSplitEnable = HI_TRUE;
    stSlice.u32SplitMode = 0;
    stSlice.u32SliceSize = 2048;
    s32Ret = HI_MPI_VENC_SetH264SliceSplit(VeChnId, &stSlice);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_SetH264SliceSplit err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }

    return HI_SUCCESS;
}
```



[See Also]

None

HI_MPI_VENC_GetH264SliceSplit

[Description]

Obtains the slice attribute of an H.264 channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetH264SliceSplit(VENC_CHN VeChn,  
VENC_PARAM_H264_SLICE_SPLIT_S *pstSliceSplit);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstSliceSplit	Slice mode of an H.264 stream	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpapi.a

[Note]

- This MPI is used to set the slice attribute of an H.264 channel.
- This MPI must be called after a VENC channel is created and before the channel is destroyed.
- You are advised to call this MPI after a VENC channel is created and before frames are encoded, to minimize times of calling this MPI during frame encoding.

[Example]

See [HI_MPI_VENC_SetH264SliceSplit](#).

[See Also]

None



HI_MPI_VENC_SetH264InterPred

[Description]

Sets the inter-prediction attribute of an H.264 channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetH264InterPred(VENC_CHN VeChn, const  
VENC_PARAM_H264_INTER_PRED_S *pstH264InterPred);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstH264InterPred	Inter-frame prediction attribute of an H.264 channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Difference]

None

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to set the inter-prediction configuration of an H.264 channel.
- The inter-prediction parameters include:
 - **u32HWSize**: size of a horizontal search window during inter-prediction. This parameter is reserved and not used currently.
 - **u32VWSize**: size of a vertical search window during inter-prediction. This parameter is reserved and not used currently.
 - **bInter16x16PredEn**: 16x16 inter-prediction enable
 - **bInter16x8PredEn**: 16x8 inter-prediction enable. This parameter is reserved and not used currently.



- **bInter8x16PredEn:** 8x16 inter-prediction enable. This parameter is reserved and not used currently.
- **bInter8x8PredEn:** 8x8 inter-prediction enable
- **bExtedgeEn:** inter-prediction edge extension enable. When inter-prediction is performed on the macroblocks on the boundary of a picture, the search window may exceed the picture boundary. In this case, you can enable edge extension to extend the picture edges for inter-prediction. If edge extension is disabled, inter-prediction is not performed on the part of the search window beyond the picture.
- This MPI is an advanced interface. You can call it as required. However, you are advised not to call this MPI. The inter-prediction attribute of an H.264 channel has default values. The default size of the search window is 0, and the other five flags are enabled by default.
- The prediction enable of **bInter16x16PredEn** can only be enabled.
- This MPI can be set after a VENC channel is created and before the channel is destroyed. This MPI takes effect after a next I frame is encoded.
- You are advised to call this MPI after a VENC channel is created and before frames are encoded, to minimize times of calling this MPI during frame encoding.
- You are advised to call the [HI_MPI_VENC_GetH264InterPred](#) MPI to obtain the InterPred configuration of the current channel before calling this MPI.

[Example]

```
HI_S32 SetInterPred(HI_VOID)
{
    HI_S32 s32Ret = HI_FAILURE;
    VENC_PARAM_H264_INTER_PRED_S stInterPred;
    VENC_CHN VeChnId = 0;
    //...omit other thing
    s32Ret = HI_MPI_VENC_GetH264InterPred(VeChnId, &stInterPred);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_GetH264InterPred err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }
    stInterPred.bExtedgeEn      = HI_TRUE;
    stInterPred.bInter16x16PredEn = HI_TRUE;
    stInterPred.bInter16x8PredEn = HI_FALSE;
    stInterPred.bInter8x16PredEn = HI_FALSE;
    stInterPred.bInter8x8PredEn = HI_FALSE;
    stInterPred.u32HWSIZE       = 4;
    stInterPred.u32VWSIZE       = 2;
    s32Ret = HI_MPI_VENC_SetH264InterPred(VeChnId, &stInterPred);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_SetH264InterPred err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }
}
```



```
    return HI_SUCCESS;  
}
```

[See Also]

None

HI_MPI_VENC_GetH264InterPred

[Description]

Obtains the inter-prediction attribute of an H.264 channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetH264InterPred(VENC_CHN VeChn,  
VENC_PARAM_H264_INTER_PRED_S *pstH264InterPred);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstH264InterPred	Inter-frame prediction attribute of an H.264 channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to obtain the inter-prediction mode of an H.264 channel.
- This MPI must be called after a VENC channel is created and before the channel is destroyed.
- You are advised to call this MPI after a VENC channel is created and before frames are encoded, to minimize times of calling this MPI during frame encoding.

[Example]

See [HI_MPI_VENC_SetH264InterPred](#).



[See Also]

None

HI_MPI_VENC_SetH264IntraPred

[Description]

Sets the intra-prediction attribute of an H.264 channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetH264IntraPred(VENC_CHN VeChn, const  
VENC_PARAM_H264_INTRA_PRED_S *pstH264IntraPred);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstH264IntraPred	Intra-frame prediction attribute of an H.264 channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Difference]

Hi3519 V100/Hi3519 V101/Hi3516C V300 supports IP camera prediction. You can enable or disable this function. This function is enabled by default.

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to set the intra-prediction configuration of an H.264 channel.
- The intra-prediction parameters include:
 - **bIntra16x16PredEn:** a flag used to enable 16x16 intra-prediction.
 - **bIntraNxNPredEn:** a flag used to enable NxN intra-prediction. Here, NxN indicates 4x4 or 8x8.
 - **bIpemEn:** a flag used to enable IPCM intra-prediction.
 - **constrained_intra_pred_flag:** For details, see the H.264 protocol.



- This MPI is an advanced interface. You can call it as required. However, you are advised not to call this MPI. The intra-prediction attribute of an H.264 channel has default values. By default, **bIntra16x16PredEn**, **bIntraNxNPredEn**, and **bIPCPredEn** are enabled and **constrained_intra_pred_flag** is set to **0**. The default values vary depending on chip type.
- Either **bIntra16x16PredEn** or **bIntraNxNPredEn** must be enabled. The system does not disable the two flags at the same time.
- This MPI can be set after a VENC channel is created and before the channel is destroyed. This MPI takes effect after a second I frame is encoded.
- You are advised to call this MPI after a VENC channel is created and before frames are encoded, to minimize times of calling this MPI during frame encoding.
- You are advised to call the [HI_MPI_VENC_GetH264IntraPred](#) MPI to obtain the intra-prediction configuration of the current channel before calling this MPI.
- Hi3516C V300 does not support quantization tables, and therefore does not support the configurations of **bScalingListValid**, **InterScalingList8X8**, and **IntraScalingList8X8**.

[Example]

```
HI_S32 SetIntraPred(HI_VOID)
{
    HI_S32 s32Ret = HI_FAILURE;
    VENC_PARAM_H264_INTRA_PRED_S stIntraPred;
    VENC_CHN VeChnId = 0;

    //...omit other thing

    s32Ret = HI_MPI_VENC_GetH264IntraPred(VeChnId, &stIntraPred);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_GetH264IntraPred err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }

    stIntraPred.bIntra16x16PredEn = HI_TRUE;
    stIntraPred.bIntraNxNPredEn = HI_FALSE;
    stIntraPred.bIpclEn = HI_FALSE;
    stIntraPred.constrained_intra_pred_flag = 0;
    s32Ret = HI_MPI_VENC_SetH264IntraPred(VeChnId, &stIntraPred);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_SetH264IntraPred err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }

    return HI_SUCCESS;
}
```



[See Also]

None

HI_MPI_VENC_GetH264IntraPred

[Description]

Obtains the intra-prediction attribute of an H.264 channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetH264IntraPred(VENC_CHN VeChn,  
VENC_PARAM_H264_INTRA_PRED_S *pstH264IntraPred);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstH264IntraPred	Intra-frame prediction attribute of an H.264 channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to obtain the intra-prediction mode of an H.264 channel.
- This MPI must be called after a VENC channel is created and before the channel is destroyed.
- You are advised to call this MPI after a VENC channel is created and before frames are encoded, to minimize times of calling this MPI during frame encoding.

[Example]

See [HI_MPI_VENC_SetH264IntraPred](#).

[See Also]

None



HI_MPI_VENC_SetH264Trans

[Description]

Sets the transformation and quantization attribute of an H.264 channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetH264Trans(VENC_CHN VeChn, const  
VENC_PARAM_H264_TRANS_S *pstH264Trans);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstH264Trans	Transformation and quantization attribute of an H.264 channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to set the conversion and quantization configurations of an H.264 channel.
- The transformation and quantization parameters include:
 - **u32IntraTransMode:** transformation attribute of an intra-macroblock. When this parameter is set to **0**, 4x4 and 8x8 transformations are supported for the intra-macroblock; when this parameter is set to **1**, only 4x4 transformation is supported for the intra-macroblock; when this parameter is set to **2**, only 8x8 transformation is supported for the intra-macroblock. 4x4 transformation (this parameter is set to **1**) is available to the H.264 baseline profile and main profile. 8x8 transformation is available only to the H.264 high profile.
 - **u32InterTransMode:** transformation attribute of an inter-macroblock. When this parameter is set to **0**, 4x4 and 8x8 transformations are supported for the intra-macroblock; when this parameter is set to **1**, only 4x4 transformation is supported for the intra-macroblock; when this parameter is set to **2**, only 8x8 transformation is supported for the intra-macroblock. 4x4 transformation (this parameter is set to **1**) is



- available to the H.264 baseline profile and main profile. 8x8 transformation is available only to the H.264 high profile.
- **bScalingListValid**: Whether **InterScalingList8x8** and **IntraScalingList8x8** are valid. **bScalingListValid** cannot be set to **true**.
 - **InterScalingList8x8**: array used by the user to provide a quantization table when 8x8 transformation is performed on an inter-frame prediction macroblock. This parameter is reserved and not used currently.
 - **IntraScalingList8x8**: array used by the user to provide a quantization table when 8x8 transformation is performed on an intra-frame prediction macroblock. This parameter is reserved and not used currently.
 - **chroma_qp_index_offset**: For details, see the H.264 protocol.
 - This MPI is an advanced interface. You can call it as required. However, you are advised not to call this MPI. The transformation and quantization attribute of an H.264 channel have default values. By default, the system sets the parameters based on the profile type.

Table 6-9 Default trans parameter values for different profile types

Profile	u32IntraTransMode	u32InterTransMode	bScalingListValid
Baseline/main profile	1	1	false
High profile	0	0	false

- This MPI can be set after a VENC channel is created and before the channel is destroyed. This MPI takes effect after the next I frame is encoded.
- You are advised to call this MPI after a VENC channel is created and before frames are encoded. This minimizes the times of calling this MPI during frame encoding.
- You are advised to call the [HI_MPI_VENC_GetH264Trans](#) MPI to obtain the trans configuration of the current channel before calling this MPI.

[Example]

```
HI_S32 SetTrans(HI_VOID)
{
    HI_S32 s32Ret = HI_FAILURE;
    VENC_PARAM_H264_TRANS_S stTrans;
    VENC_CHN VeChnId = 0;

    //...omit other thing

    s32Ret = HI_MPI_VENC_GetH264Trans(VeChnId, &stTrans);
    if (HI_SUCCESS != s32Ret)
    {
        printf("stTrans err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }
}
```



```
    stTrans.u32IntraTransMode      = 2;
    stTrans.u32InterTransMode     = 2;
    stTrans.bScalingListValid    = HI_FALSE;
    stTrans.chroma_qp_index_offset = 2;
    s32Ret = HI_MPI_VENC_SetH264Trans(VeChnId, &stTrans);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_SetH264Trans err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }

    return HI_SUCCESS;
}
```

[See Also]

None

HI_MPI_VENC_GetH264Trans

[Description]

Obtains the transformation and quantization attribute of an H.264 channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetH264Trans(VENC_CHN VeChn, VENC_PARAM_H264_TRANS_S
*pstH264Trans);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstH264Trans	Transformation and quantization attribute of an H.264 channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]



- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to obtain the transformation and quantization configurations of an H.264 channel.
- This MPI must be called after a VENC channel is created and before the channel is destroyed.
- You are advised to call this MPI after a VENC channel is created and before frames are encoded, to minimize times of calling this MPI during frame encoding.

[Example]

See [HI_MPI_VENC_SetH264Trans](#).

[See Also]

None

HI_MPI_VENC_SetH264Entropy

[Description]

Sets the entropy encoding mode for an H.264 channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetH264Entropy(VENC_CHN VeChn, const  
VENC_PARAM_H264_ENTROPY_S *pstH264EntropyEnc);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstH264EntropyEnc	Entropy encoding mode for an H.264 channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a



[Note]

- This MPI is used to set the entropy encoding configuration of an H.264 channel.
- The entropy encoding parameters include:
 - **u32EntropyEncModeI**: entropy encoding mode of an I frame. When this parameter is set to **0**, the I frame is encoded in **context adaptive variable length coding (CAVLC)** mode; when this parameter is set to **1**, the I frame is encoded in context-based adaptive binary arithmetic coding (CABAC) mode.
 - **u32EntropyEncModeP**: entropy encoding mode of a P frame. When this parameter is set to **0**, the P frame is encoded in CAVLC mode; when this parameter is set to **1**, the P frame is encoded in CABAC mode.
 - **u32EntropyEncModeB**: entropy encoding mode of a B frame. When this parameter is set to **0**, the B frame is encoded in CAVLC mode; when this parameter is set to **1**, the B frame is encoded in CABAC mode.
 - **cabac_stuff_en**: enable for CABAC encoding stuffing. The default value is 0. For details, see the H.264 protocol.
 - **Cabac_init_idc**: index for CABAC initialization. The default value is 0. For details, see the H.264 protocol.
- The entropy modes of I/P/B frame can be set independently.
- The baseline profile does not support CABAC encoding and B frames, but supports CAVLC encoding mode. The main profile and high profile support both the CABAC and CAVLC encoding modes.
- Compared with the CAVLC encoding, CABAC encoding requires greater calculation amount but generates fewer streams. If system performance is low, you are advised to use CAVLC encoding for I frame and CABAC encoding for P/B frame.
- This MPI is an advanced interface. You can call it as required. However, you are advised not to call this MPI. The entropy encoding mode for an H.264 channel has a default value. By default, the system sets parameters based on the profile type.

Table 6-10 Default entropy encoding parameter values for different profile types

Profile	u32EntropyEncModeI	u32EntropyEncModeP	u32EntropyEncModeB
Baseline	0	0	0
Main profile/High profile	1	1	1
svc-t	1	1	1

- This MPI can be set after a VENC channel is created and before the channel is destroyed. This MPI takes effect after a next I frame is encoded.
- You are advised to call this MPI after a VENC channel is created and before frames are encoded, to minimize times of calling this MPI during frame encoding.
- You are advised to call the [HI_MPI_VENC_GetH264Entropy](#) MPI to obtain the entropy configuration of the current channel before calling this MPI.

[Example]



```
HI_S32 SetEntropy(HI_VOID)
{
    HI_S32 s32Ret = HI_FAILURE;
    VENC_PARAM_H264_ENTROPY_S stEntropy;
    VENC_CHN VeChnId = 0;

    //...omit other thing
    s32Ret = HI_MPI_VENC_GetH264Entropy(VeChnId, &stTrans);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_GetH264Entropy err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }

    stEntropy.u32EntropyEncModeI = 0;
    stEntropy.u32EntropyEncModeP = 0;
    stEntropy.cabac_stuff_en = 0;
    stEntropy.Cabac_init_idc = 0;
    s32Ret = HI_MPI_VENC_SetH264Entropy(VeChnId, &stEntropy);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_SetH264Entropy err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }
    return HI_SUCCESS;
}
```

[See Also]

None

HI_MPI_VENC_GetH264Entropy

[Description]

Obtains the entropy encoding attribute of an H.264 channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetH264Entropy(VENC_CHN VeChn,
VENC_PARAM_H264_ENTROPY_S *pstH264EntropyEnc);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input



Parameter	Description	Input/Output
pstH264EntropyEnc	Entropy encoding mode of an H.264 channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to obtain the entropy encoding configuration of an H.264 channel.
- This MPI must be called after a VENC channel is created and before the channel is destroyed.
- You are advised to call this MPI after a VENC channel is created and before frames are encoded, to minimize times of calling this MPI during frame encoding.

[Example]

See [HI_MPI_VENC_SetH264Entropy](#).

[See Also]

None

HI_MPI_VENC_SetH264Poc

[Description]

Sets the picture order count (POC) type of an H.264 channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetH264Poc(VENC_CHN VeChn, const VENC_PARAM_H264_POC_S  
*pstH264Poc);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstH264Poc	POC type of an H.264 channel	Input



[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to set the POC configuration of an H.264 channel.
- The POC attribute mainly indicates the POC type of H.264 streams. For details, see the H.264 protocol.
- There are three POC types, which are specified by **pic_order_cnt_type** (the value can be set to **0**, **1**, or **2**). By default, it is set to **2**.
- This MPI can be set after a VENC channel is created and before the channel is destroyed. This MPI takes effect after a next I frame is encoded.
- You are advised to call this MPI after a VENC channel is created and before frames are encoded, to minimize times of calling this MPI during frame encoding.
- You are advised to call the [HI_MPI_VENC_GetH264Poc](#) MPI to obtain the POC configuration of the current channel before calling this MPI.
- When **GopMode** is set to **VENC_GOPMODE_BIPREDB**, **pic_order_cnt_type** cannot be set to **2**.

[Example]

```
HI_S32 SetPoc(HI_VOID)
{
    HI_S32 s32Ret = HI_FAILURE;
    VENC_PARAM_H264_POC_S           stPoc;
    VENC_CHN VeChnId = 0;

    //...omit other thing

    s32Ret = HI_MPI_VENC_GetH264Poc(VeChnId, &stPoc);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_GetH264Poc err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }

    stPoc.pic_order_cnt_type = 0;
```



```
s32Ret = HI_MPI_VENC_SetH264Poc(VeChnId, &stPoc);  
if (HI_SUCCESS != s32Ret)  
{  
    printf("HI_MPI_VENC_SetH264Poc err 0x%x\n", s32Ret);  
    return HI_FAILURE;  
}  
  
return HI_SUCCESS;  
}
```

[See Also]

None

HI_MPI_VENC_GetH264Poc

[Description]

Obtains the POC type of an H.264 channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetH264Poc(VENC_CHN VeChn, VENC_PARAM_H264_POC_S  
*pstH264Poc);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstH264Poc	POC type of an H.264 channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to obtain the POC configuration of an H.264 channel.



- This MPI must be called after a VENC channel is created and before the channel is destroyed.
- You are advised to call this MPI after a VENC channel is created and before frames are encoded, to minimize times of calling this MPI during frame encoding.

[Example]

See [HI_MPI_VENC_SetH264Poc](#).

[See Also]

None

HI_MPI_VENC_SetH264Dbblk

[Description]

Sets the de-blocking type of an H.264 channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetH264Dbblk(VENC_CHN VeChn, const  
VENC_PARAM_H264_DBLK_S *pstH264Dbblk);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstH264Dbblk	De-blocking type of an H.264 parameter	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to set the de-blocking configuration of an H.264 channel.
- The de-blocking parameters include:
 - **disable_deblocking_filter_idc**: For details, see the H.264 protocol.
 - **slice_alpha_c0_offset_div2**: For details, see the H.264 protocol.



- **slice_beta_offset_div2**: For details, see the H.264 protocol.
- The de-blocking function is enabled by default, that is, **disable_deblocking_filter_idc**, **slice_alpha_c0_offset_div2**, and **slice_beta_offset_div2** are **0** by default.
- To disable the de-blocking function, set **disable_deblocking_filter_idc** to **1**.
- This MPI can be set after a VENC channel is created and before the channel is destroyed. This MPI takes effect after a next I frame is encoded.
- You are advised to call this MPI after a VENC channel is created and before frames are encoded, to minimize times of calling this MPI during frame encoding.
- You are advised to call the **HI_MPI_VENC_GetH264Dbblk** MPI to obtain the de-blocking configuration of the current channel before calling this MPI.

[Example]

```
HI_S32 SetDbblk(HI_VOID)
{
    HI_S32 s32Ret = HI_FAILURE;
    VENC_PARAM_H264_DBULK_S           stDbblk;
    VENC_CHN VeChnId = 0;

    //...omit other thing

    s32Ret = HI_MPI_VENC_GetH264Dbblk(VeChnId, &stDbblk);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_GetH264Dbblk err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }

    stDbblk.disable_deblocking_filter_idc = 0;
    stDbblk.slice_alpha_c0_offset_div2   = 6;
    stDbblk.slice_beta_offset_div2      = 5;
    s32Ret = HI_MPI_VENC_SetH264Dbblk(VeChnId, &stDbblk);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_SetH264Dbblk err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }

    return HI_SUCCESS;
}
```

[See Also]

None



HI_MPI_VENC_GetH264Dbblk

[Description]

Obtains the de-blocking type of an H.264 channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetH264Dbblk(VENC_CHN VeChn, VENC_PARAM_H264_DBLK_S  
*pstH264Dbblk);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstH264Dbblk	De-blocking type of an H.264 channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to obtain the de-blocking configuration of an H.264 channel.
- This MPI must be called after a VENC channel is created and before the channel is destroyed.
- You are advised to call this MPI after a VENC channel is created and before frames are encoded, to minimize times of calling this MPI during frame encoding.

[Example]

See [HI_MPI_VENC_SetH264Dbblk](#).

[See Also]

None

HI_MPI_VENC_SetH264Vui

[Description]

Sets the VUI parameters of an H.264 channel.



[Syntax]

```
HI_S32 HI_MPI_VENC_SetH264Vui(VENC_CHN VeChn, const VENC_PARAM_H264_VUI_S *pstH264Vui);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstH264Vui	VUI of an H.264 channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to set the VUI configuration of an H.264 channel.
- This MPI can be set after a VENC channel is created and before the channel is destroyed. This MPI takes effect after a next I frame is encoded.
- You are advised to call this MPI after a VENC channel is created and before frames are encoded, to minimize times of calling this MPI during frame encoding.
- You are advised to call [HI_MPI_VENC_GetH264Vui](#) to obtain the VUI configuration of the current channel before calling this MPI.

[Example]

```
HI_S32 SetVui(HI_VOID)
{
    HI_S32 s32Ret = HI_FAILURE;
    VENC_PARAM_H264_VUI_S          stVui;
    VENC_CHN VeChnId = 0;

    //...omit other thing

    s32Ret = HI_MPI_VENC_SetH264Vui(VeChnId, &stVui);
    if (HI_SUCCESS != s32Ret)
```



```
{  
    printf("HI_MPI_VENC_GetH264Vui err 0x%x\n", s32Ret);  
    return HI_FAILURE;  
}  
  
stVui.timing_info_present_flag = 1;  
s32Ret = HI_MPI_VENC_SetH264Vui(VeChnId, &stVui);  
if (HI_SUCCESS != s32Ret)  
{  
    printf("HI_MPI_VENC_SetH264Vui err 0x%x\n", s32Ret);  
    return HI_FAILURE;  
}  
  
return HI_SUCCESS;  
}
```

[See Also]

None

HI_MPI_VENC_GetH264Vui

[Description]

Obtains the VUI parameter settings of an H.264 channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetH264Vui(VENC_CHN VeChn, VENC\_PARAM\_H264\_VUI\_S  
*pstH264Vui);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstH264Vui	VUI of an H.264 channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]



- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to obtain the VUI configuration of an H.264 channel.
- This MPI must be called after a VENC channel is created and before the channel is destroyed.
- You are advised to call this MPI after a VENC channel is created and before frames are encoded, to minimize times of calling this MPI during frame encoding.

[Example]

See [HI_MPI_VENC_SetH264Vui](#).

[See Also]

None

HI_MPI_VENC_SetH265Vui

[Description]

Sets the VUI parameters of an H.265 VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetH265Vui(VENC_CHN VeChn, const VENC_PARAM_H265_VUI_S *pstH265Vui);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstH265Vui	VUI parameter of an H.265 VENC channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]



- This MPI is used to configure the VUI parameters of an H.265 VENC channel.
- This MPI can be called after a VENC channel is created and before the channel is destroyed. If this MPI is called during encoding, the operation takes effect only when the next I frame is encoded.
- You are advised to call this MPI after creating a channel but before starting encoding. This reduces the number of times of calling this MPI during encoding.
- Before calling this MPI, you are advised to call [HI_MPI_VENC_GetH265Vui](#) to obtain the VUI parameter settings of the current VENC channel.

[Example]

```
HI_S32 SetVui(HI_VOID)
{
    HI_S32 s32Ret = HI_FAILURE;
    VENC_PARAM_H265_VUI_S          stVui;
    VENC_CHN VeChnId = 0;

    //...omit other thing

    s32Ret = HI_MPI_VENC_GetH265Vui(VeChnId, &stVui);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_GetH265Vui err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }

    stVui.timing_info_present_flag = 1;
    s32Ret = HI_MPI_VENC_SetH265Vui(VeChnId, &stVui);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_SetH265Vui err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }
    return HI_SUCCESS;
}
```

[See Also]

None

[HI_MPI_VENC_GetH265Vui](#)

[Description]

Obtains the VUI parameter settings of an H.265 VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetH265Vui(VENC_CHN VeChn, VENC_PARAM_H264_VUI_S
*pstH265Vui);
```



[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstH264Vui	VUI attribute of an H.265 VENC channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to obtain the VUI parameter settings of an H.265 VENC channel.
- This MPI can be called after a VENC channel is created and before the channel is destroyed.
- You are advised to call this MPI after creating a channel but before starting encoding. This reduces the number of times of calling this MPI during encoding.

[Example]

See the example of [HI_MPI_VENC_SetH265Vui](#).

[See Also]

None

HI_MPI_VENC_SetJpegParam

[Description]

Sets the advanced parameters of a JPEG channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetJpegParam(VENC_CHN VeChn, const VENC_PARAM_JPEG_S  
*pstJpegParam);
```

[Parameter]



Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstJpegParam	Advanced parameters of a JPEG channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to set the advanced parameters of a JPEG channel.
- The advanced parameters include:
 - u32Qfactor:** The value of a quantization factor ranges from **1** to **99**. When this parameter is set to a larger value, the quantization coefficient in the quantization table becomes smaller, the obtained picture quality becomes better, and the encoding compression rate becomes lower; when this parameter is set to a smaller value, the quantization coefficient in the quantization table becomes larger, the obtained picture quality becomes poorer, and the encoding compression rate becomes higher. For details about the relationship between this parameter and the quantization table, see the related RFC2435 standard.
 - u8YQt[64], u8CbQt[64], and u8CrQt[64]:** These parameters correspond to three quantization tables. You can set a quantization table by using these three parameters.
 - u32MCUPerECS:** indicates the number of minimum coded units (MCUs) of each ECS. When this parameter is set to **0**, all MCUs in the current frame are encoded into an ECS. The value range is $(\text{picwidth} + 15) \gg 4 \times (\text{picheight} + 15) \gg 4 \times 2$.
- To use a quantization table, set **Qfactor** to **50** when setting the quantization table.
- This MPI can be set after a VENC channel is created and before the channel is destroyed. This MPI takes effect after a next I frame is encoded.
- You are advised to call this MPI after a VENC channel is created and before frames are encoded, to minimize times of calling this MPI during frame encoding.
- You are advised to call the [HI_MPI_VENC_SetJpegParam](#) MPI to obtain the JPEG configuration of the current channel before calling this MPI.

[Example]

```
HI_S32 SetJpegParam(HI_VOID)
{
```



```
HI_S32 s32Ret = HI_FAILURE;
VENC_PARAM_JPEG_S stParamJpeg;
VENC_CHN VeChnId = 0;

//...omit other thing

s32Ret = HI_MPI_VENC_GetJpegParam(VeChnId, &stParamJpeg);
if (HI_SUCCESS != s32Ret)
{
    printf("HI_MPI_VENC_GetJpegParam err 0x%x\n", s32Ret);
    return HI_FAILURE;
}

stParamJpeg.u32MCUPerECS = 100;
for (i = 0; i < 64; i++)
{
    stParamJpeg.u8YQt[i] = 16;
    stParamJpeg.u8CbQt[i] = 17;
    stParamJpeg.u8CrQt[i] = 18;
}
s32Ret = HI_MPI_VENC_SetJpegParam(VeChnId, &stParamJpeg);
if (HI_SUCCESS != s32Ret)
{
    printf("HI_MPI_VENC_SetJpegParam err 0x%x\n", s32Ret);
    return HI_FAILURE;
}
return HI_SUCCESS;
}
```

[See Also]

None

HI_MPI_VENC_GetJpegParam

[Description]

Obtains the advanced parameters of a JPEG channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetJpegParam(VENC_CHN VeChn, VENC_PARAM_JPEG_S
*pstJpegParam);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID	Input



Parameter	Description	Input/Output
	Value range: [0, VENC_MAX_CHN_NUM)	
pstJpegParam	Advanced parameters of a JPEG channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to obtain the configurations of advanced parameters of a JPEG channel.
- This MPI must be called after a VENC channel is created and before the channel is destroyed.
- You are advised to call this MPI after a VENC channel is created and before frames are encoded, to minimize times of calling this MPI during frame encoding.

[Example]

See [HI_MPI_VENC_SetJpegParam](#).

[See Also]

None

HI_MPI_VENC_SetMjpegParam

[Description]

Sets the advanced parameters of an MJPEG channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetMjpegParam(VENC_CHN VeChn, const VENC_PARAM_MJPEG_S *pstMjpegParam);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input



Parameter	Description	Input/Output
pstMjpegParam	Advanced parameter of an MJPEG channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to set the advanced parameters of an MJPEG channel.
- The following describes four advanced parameters:
 - u8YQt[64], u8CbQt[64], and u8CrQt[64]: The three parameters correspond to three quantization tables. You can set quantization tables by setting the parameters.
 - u32MCUPerECS: It indicates the number of MCUs in each ECS. When u32MCUPerECS is set to 0, all MCUs of the current frame are encoded as an ECS. The minimum value of u32MCUPerECS is 0, and the maximum value of u32MCUPerECS cannot be greater than {[picwidth + 15] >> 4] x {[picheight + 15] >> 4} x 2.
- To use the user-defined quantization tables, set **Qfactor** to **50** when setting quantization tables.
- This MPI must be called after an MJPEG channel is created and before the channel is destroyed. If the MPI is called during encoding, the operation takes effect only after the next frame is encoded.
- You are advised to call this MPI after creating a channel and before starting encoding. This reduces the number of times of calling this MPI during encoding.
- Before calling this MPI, you are advised to call [HI_MPI_VENC_GetMjpegParam](#) to obtain the MjpegParam configurations of the current encoding channel.

[Example]

```
HI_S32 SetMjpegParam(HI_VOID)
{
    HI_S32 s32Ret = HI_FAILURE;
    VENC_PARAM_MJPEG_S stParamMjpeg;
    VENC_CHN VeChnId = 0;
    //...omit other thing

    s32Ret = HI_MPI_VENC_GetMjpegParam(VeChnId, &stParamMjpeg);
```



```
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_GetMjpegParam err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }

    stParamMjpeg.u32MCUPerECS = 100;
    for (i = 0; i < 64; i++)
    {
        stParamMjpeg.u8YQt[i] = 16;
        stParamMjpeg.u8CbQt[i] = 17;
        stParamMjpeg.u8CrQt[i] = 18;
    }
    s32Ret = HI_MPI_VENC_SetMjpegParam(VeChnId, &stParamMjpeg);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_SetMjpegParam err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }

    return HI_SUCCESS;
}
```

[See Also]

None

HI_MPI_VENC_GetMjpegParam

[Description]

Obtains the advanced parameters of an MJPEG channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetMjpegParam(VENC_CHN VeChn, VENC\_PARAM\_MJPEG\_S
*pstMjpegParam);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstMjpegParam	Advanced parameters of an MJPEG channel	Output

[Return Value]



Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to obtain the configurations of advanced parameters of an MJPEG channel.
- This MPI must be called after an MJPEG channel is created and before the channel is destroyed.
- You are advised to call this MPI after creating a channel and before starting encoding. This reduces the number of times of calling this MPI during encoding.

[Example]

See the example of [HI_MPI_VENC_SetMjpegParam](#).

[See Also]

None

HI_MPI_VENC_SetFrameRate

[Description]

Sets the frame rate control attribute of a VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetFrameRate(VENC_CHN VeChn, const VENC_FRAME_RATE_S  
*pstFrameRate);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstFrameRate	Pointer to frame rate control attribute of a VENC channel	Input

[Return Value]



Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Request]

- Header files: hi_comm_venc.h, mpi_venc.h.
- Library file: libmpi.a

[Note]

- The frame rate control attribute of a VENC channel includes the input frame rate **SrcFrmRate** and output frame rate **DstFrmRate**.
- If you set the attributes of a VENC channel that does not exist, an error code indicating failure is returned.
- If **pstFrameRate** is not set, an error code indicating failure is returned.
- Both **SrcFrmRate** and **DstFrmRate** must be greater than 0 or equal to -1 when you set the frame rate control attribute of the channel.
- When **SrcFrmRate** and **DstFrmRate** are -1, the frame rate is not controlled.
- The frame adding mode is used if the input frame rate **SrcFrmRate** is less than the output frame rate **DstFrmRate**. In this mode, **DstFrmRate** is used as the absolute output frame rate. For example, when **SrcFrmRate** is less than **DstFrmRate** and **DstFrmRate** is 30, 30 frames are output finally.
- The frame discarding (frame reduction) mode is used if the input frame rate **SrcFrmRate** is greater than or equal to the output frame rate **DstFrmRate**. In this mode, some input frames are chosen for encoding. For example, when **SrcFrmRate** is 30 and **DstFrmRate** is 15, only 15 frames of the 30 input pictures are encoded.
- In frame discarding (frame reduction) mode, when the frequency of the front-end source (such as the VI) is fixed, the controlled channel frame rate is accurate. When the frequency of the front-end source (such as the VDEC) is not fixed, the controlled channel frame rate is inaccurate.
- The JPEG encoding mode does not support the frame adding mode, that is, the input frame rate (**SrcFrmRate**) must be greater than or equal to the output frame rate (**DstFrmRate**).

[Example]

None

[See Also]

None

HI_MPI_VENC_GetFrameRate

[Description]

Obtains the frame rate control attribute of a VENC channel.

[Syntax]



```
HI_S32 HI_MPI_VENC_GetFrameRate(VENC_CHN VeChn, VENC_FRAME_RATE_S  
*pstFrameRate);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstFrameRate	Pointer to frame rate control attribute of a VENC channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Request]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- If you obtain the attribute of channel that does not exist, the error code [HI_ERR_VENC_UNEXIST](#) is returned.
- If **pstFrameRate** is null, an error code indicating failure is returned.

[Example]

None

[See Also]

None

HI_MPI_VENC_SetRcParam

[Description]

Sets the advanced RC parameters of a VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetRcParam(VENC_CHN VeChn, const VENC_RC_PARAM_S  
*pstRcParam);
```

[Parameter]



Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstRcParam	Pointer to the advanced RC parameters of a VENC channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- The advanced parameters have default values. You do not need to call this MPI to enable a VENC channel.
- You are advised to call [HI_MPI_VENC_GetRcParam](#) to obtain the values of the advanced parameters for the RC, modify parameters values, and call [HI_MPI_VENC_SetRcPara](#) to set advanced parameters.
- Currently the advanced RC parameters only apply to the CBR mode and VBR mode of H.264/H.265 MJPEG as well as the H.264/H.265 AVBR mode.
- The advanced parameters for the RC are as follows:
 - u32Thrd[RC_TEXTURE_THR_SIZE], u32ThrdP[RC_TEXTURE_THR_SIZE], and u32ThrdB[RC_TEXTURE_THR_SIZE]:** indicates three groups of thresholds for evaluating the macroblock complexity of the I frame, P frame, and B frame respectively. The thresholds in each group are sorted in ascending order. The value of each threshold ranges from 0 to 255. When the bit rate is controlled based on the macroblock, the thresholds can be used to adjust the QP values of macroblocks based on picture complexity.

According to the H.264/H.265 protocol for Hi3519 V100/ Hi3519 V101/Hi3516C V300, the QP value of the current macroblock is the start QP value of the macroblock row plus or minus a number (the maximum number is 24). The added number is controlled by the first eight values in the threshold array, while the subtracted number is controlled by the last eight values in the threshold array. If the picture complexity of the current macroblock is less than or equal to **u32Thr[7]**, the QP value of the current macroblock is the start QP value of the macroblock row minus x ; if the picture complexity of the current macroblock is greater than **u32Thr[7]**, the QP value of the current macroblock is the start QP value of the macroblock row plus y . x and y are calculated as follows (C indicates the picture complexity):



if $C < \text{u32Thrd}[0]$, $x = \sum_0^{j-1} \text{u32QpDeltaLevel}[i]$

if $\text{u32Thrd}[0] \leq C < \text{u32Thr}[1]$, $x = \sum_1^{j-1} \text{u32QpDeltaLevel}[i]$

if $\text{u32Thrd}[1] \leq C < \text{u32Thr}[2]$, $x = \sum_2^{j-1} \text{u32QpDeltaLevel}[i]$

if $\text{u32Thrd}[2] \leq C < \text{u32Thr}[3]$, $x = \sum_3^{j-1} \text{u32QpDeltaLevel}[i]$

if $\text{u32Thrd}[3] \leq C < \text{u32Thr}[4]$, $x = \sum_{i=4}^7 \text{u32QpDeltaLevel}[i]$

if $\text{u32Thrd}[4] \leq C < \text{u32Thr}[5]$, $x = \sum_{i=5}^7 \text{u32QpDeltaLevel}[i]$

if $\text{u32Thrd}[5] \leq C < \text{u32Thr}[6]$, $x = \sum_{i=6}^7 \text{u32QpDeltaLevel}[i]$

if $\text{u32Thrd}[6] \leq C < \text{u32Thr}[7]$, $x = \text{u32QpDeltaLevel}[7]$

if $\text{u32Thrd}[7] \leq C \leq \text{u32Thr}[8]$, $x = y = 0$,

if $\text{u32Thrd}[8] < C \leq \text{u32Thr}[9]$, $y = \text{u32QpDeltaLevel}[8]$

if $\text{u32Thrd}[9] < C \leq \text{u32Thr}[10]$, $y = \sum_{i=8}^9 \text{u32QpDeltaLevel}[i]$

if $\text{u32Thrd}[10] < C \leq \text{u32Thr}[11]$, $y = \sum_{i=8}^{10} \text{u32QpDeltaLevel}[i]$

if $\text{u32Thrd}[11] < C \leq \text{u32Thr}[12]$, $y = \sum_{i=8}^{11} \text{u32QpDeltaLevel}[i]$

if $\text{u32Thrd}[12] < C \leq \text{u32Thr}[13]$, $y = \sum_{i=8}^{12} \text{u32QpDeltaLevel}[i]$

if $\text{u32Thrd}[13] < C \leq \text{u32Thr}[14]$, $y = \sum_{i=8}^{13} \text{u32QpDeltaLevel}[i]$

if $\text{u32Thrd}[14] < C \leq \text{u32Thr}[15]$, $y = \sum_{i=8}^{14} \text{u32QpDeltaLevel}[i]$

if $\text{u32Thrd}[15] < C$, $y = \sum_{i=8}^{15} \text{u32QpDeltaLevel}[i];$

For H.264/H.265 encoding, the default values are **au32ThrdI**[0, 0, 0, 0, 3, 3, 5, 5, 8, 8, 8, 15, 15, 20, 25, 25], **au32ThrdP**[0, 0, 0, 0, 3, 3, 5, 5, 8, 8, 8, 15, 15, 20, 25, 25], and **au32ThrdB**[0, 0, 0, 0, 3, 3, 5, 5, 8, 8, 8, 15, 15, 20, 25, 25].

- **u32QpDeltaLevelI**[[RC_TEXTURE_THR_SIZE](#)],
u32QpDeltaLevelP[[RC_TEXTURE_THR_SIZE](#)], and
u32QpDeltaLevelB[[RC_TEXTURE_THR_SIZE](#)]



Indicate the QP delta level that is added to or subtracted from the start QP value of the current macroblock row to calculate the QP value of the current macroblock when the picture complexity of the current macroblock is between two consecutive thresholds during macroblock-level bit rate control. If C (picture complexity) is greater than or equal to **u32Thrd[6]** and less than **u32Thrd[7]**, the QP value of the current macroblock is the start QP value of the current macroblock row minus **u32QpDeltaLevel[7]**. The value range of **u32QpDeltaLevel[7]** is [0, 3].

- **u32RowQpDelta**

Indicates the fluctuation amplitude of the start QP value of each macroblock row relative to the start QP value of the frame when the bit rate is controlled based on the macroblock. If the system requires that the bit rate fluctuates slightly, you can increase the value of **u32QpDelta** to control the bit rate accurately. However, the quality of the macroblocks of some pictures may be different. In high bit rate mode, the recommended value of **u32QpDelta** is **0**; in medium bit rate mode, the recommended value is **0** or **1**; in low bit rate mode, the recommended value is **2, 3, 4**, or **5**.

- **s32FirstFrameStartQp**

Indicates the start QP value of the first frame, valid in CBR, VBR and AVBR mode. If **s32FirstFrameStartQp** is **-1**, the start QP value of the first frame is internally calculated by the encoder. If **s32FirstFrameStartQp** is set to any other valid value, the user specifies this valid value as the start QP value of the first frame. The default value is **-1**. The first frame indicates the first IDR frame after the channel is created or the GOP mode, RC mode, or resolution is switched.

The CBR parameters are as follows:

- **u32MinIprop** and **u32MaxIprop**

Indicate the minimum IP proportion and maximum IP proportion respectively. The two parameters are advanced CBR parameters. The IP proportion is the ratio of the bits of I frames to the bits of P frames. The two parameters control the range of each IP proportion. When the value of **u32MinIprop** increases, the I frame becomes distinct and the P frame becomes blurred. When the value of **u32MaxIprop** decreases, the I frame becomes blurred and the P frame becomes distinct. You are advised not to limit the IP proportion. This avoids the respiratory effect and bit rate fluctuation. The default value of **u32MinIprop** is **1**, and the default value of **u32MaxIprop** is **100**. If the size of the I frame is limited, you can set the values of **u32MinIprop** and **u32MaxIprop** based on requirement on the size change of the I frame.

- **u32MaxQp** and **u32MinQp**

Indicate the maximum QP value and minimum QP value of the current frame, respectively. The two parameters are advanced CBR parameters. The clamping effect is strong. All other adjustments on the picture QP value, such as the macroblock bit rate control, are restricted to fall within the range of **u32MinQp** to **u32MaxQp**. The default values of **u32MinQp** and **u32MaxQp** are **10** and **51**, respectively. You are advised to retain the default values if there are no special quality requirements.

- **u32MaxIQp** and **u32MinIQp**

Indicate the maximum QP value and minimum QP value of the current IDR frame, respectively. The two parameters are advanced CBR parameters. The clamping effect is the strongest. All other adjustments on the picture QP value, such as the macroblock bit rate control, are restricted to fall within the range of **u32MaxIQp** and **u32MinIQp**. The default values of **u32MinIQp** and **u32MaxIQp** are **10** and **45**, respectively. You are advised to retain the default values if there are no special quality requirements.



The VBR parameters are as follows:

- **s32ChangePos**

Indicates the ratio of the bit rate when the QP value starts to be adjusted in VBR mode to the maximum bit rate. This parameter is an advanced VBR parameter, and its default value is 90. If the macroblocks of consecutive frames in the similar positions have significant differences but the maximum bit rate cannot be exceeded, you are advised to decrease the value of **s32ChangePos** and increase the value of **s32DeltaQP**. However, when the bit rate becomes stable, the bit rate is low and the picture quality is poor.

- **u32MinIprop** and **u32MaxIprop**

Indicate the minimum I/P ratio and maximum I/P ratio, respectively. The two parameters are advanced VBR parameters. The I/P ratio is the ratio of the bits of the I frame to the bits of the P frame. The two parameters control the range of the I/P ratio. When the value of **u32MinIprop** is increased, the I frame becomes clear and the P frame becomes blurred. When the value of **u32MaxIprop** is decreased, the I frame becomes blurred and the P frame becomes clear. You are advised not to limit the I/P ratio in normal cases to avoid the respiratory effect and bit rate fluctuation. The default values of **u32MinIprop** and **u32MaxIprop** are **1** and **100**, respectively. In the scenario where the size of the I frame is limited, you can set **u32MinIprop** and **u32MaxIprop** based on the requirement on the size fluctuation of the I frame.

The AVBR parameters are as follows:

- **s32ChangePos**

Indicates the ratio of the bit rate when the QP value starts to be adjusted in AVBR mode to the maximum bit rate. This parameter is an advanced AVBR parameter, and its default value is 90. If the macroblocks of consecutive frames in the similar positions have significant differences but the maximum bit rate cannot be exceeded, you are advised to decrease the value of **s32ChangePos** and increase the value of **s32DeltaQP**. However, when the bit rate becomes stable, the bit rate is low and the picture quality is poor.

- **u32MinIprop** and **u32MaxIprop**

Indicate the minimum I/P ratio and maximum I/P ratio, respectively. The two parameters are advanced AVBR parameters. The I/P ratio is the ratio of the bits of the I frame to the bits of the P frame. The two parameters control the range of the I/P ratio. When the value of **u32MinIprop** is increased, the I frame becomes clear and the P frame becomes blurred. When the value of **u32MaxIprop** is decreased, the I frame becomes blurred and the P frame becomes clear. You are advised not to limit the I/P ratio in normal cases to avoid the respiratory effect and bit rate fluctuation. The default values of **u32MinIprop** and **u32MaxIprop** are **1** and **100**, respectively. In the scenario where the size of the I frame is limited, you can set **u32MinIprop** and **u32MaxIprop** based on the requirement on the size fluctuation of the I frame.

- **u32MaxQp** and **u32MinQp**

Indicate the maximum QP value and minimum QP value of the current frame, respectively. The two parameters are advanced AVBR parameters. The clamping effect is the strongest. All other adjustments on the picture QP value, such as the macroblock bit rate control, are restricted to fall within the range of **u32MaxIQp** and **u32MinIQp**. The default values of **u32MinQp** and **u32MaxQp** are **16** and **51** respectively. You are advised to retain the default values if there are no special quality requirements.

- **u32MaxIQp** and **u32MinIQp**

Indicate the maximum QP value and minimum QP value of the current IDR frame, respectively. The two parameters are advanced AVBR parameters. The clamping



effect is the strongest. All other adjustments on the picture QP value, such as the macroblock bit rate control, are restricted to fall within the range of **u32MaxIQp** and **u32MinIQp**. The default values of **u32MinIQp** and **u32MaxIQp** are **16** and **51**, respectively. You are advised to retain the default values if there are no special quality requirements.

- **PRcParam**

Indicates an advanced user-defined parameter of the RC. It is reserved.

- You are advised to call this MPI after creating a channel and before starting encoding. This reduces the number of times of calling this MPI during encoding.

[Example]

```
HI_S32 SetRcParam(HI_VOID)
{
    HI_S32 s32Ret = HI_FAILURE;
    VENC_RC_PARAM_S stVencRcPara;
    VENC_CHN VeChnId = 0;
    //...omit other thing
    s32Ret = HI_MPI_VENC_GetRcParam(VeChnId, &stVencRcPara);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_GetRcParam err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }
    stVencRcPara.stParamH264Cbr.enSuperFrmMode = SUPERFRM_DISCARD;

    s32Ret = HI_MPI_VENC_SetRcParam(VeChnId, &stVencRcPara);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_SetRcParam err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }
    //...omit other thing
    return HI_SUCCESS;
}
```

[See Also]

None

HI_MPI_VENC_GetRcParam

[Description]

Obtains the advanced RC parameters of a VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetRcParam(VENC_CHN VeChn, VENC_RC_PARAM_S
*pstRcParam);
```



[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstRcParam	Pointer to the advanced RC parameters of a VENC channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpapi.a

[Note]

- This MPI is used to obtain the configurations of the advanced parameters for the RC of an H.264/H.265/MJPEG channel. For details about parameters, see the description of [VENC_RC_PARAM_S](#).
- If **pstRcParam** is null, an error code indicating failure is returned.

[Example]

None

[See Also]

None

HI_MPI_VENC_SetRefParam

[Description]

Sets the advanced frame skipping reference parameters for an H.264/H.265 VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetRefParam(VENC_CHN VeChn, const VENC_PARAM_REF_S *  
pstRefParam);
```

[Parameter]



Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstRefParam	Advanced frame skipping reference parameters for an H.264/H.265 VENC channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- If **pstRefParam** is null, an error code indicating failure is returned.
- When an H.264/H.265 VENC channel is created, the 1x frame skipping reference mode is used by default. If you want to change the reference mode, you are advised to call this MPI to set the reference mode after creating the VENC channel but before starting encoding. This reduces the times of calling the MPI during encoding.
- If the MPI is called during encoding, the operation takes effect only when the next I frame is encoded.
- If the profile of an H.264 VENC channel is svc-t, an error code is returned after this MPI is called, indicating that this operation is prohibited.
- If the 1x frame skipping reference mode is selected, set **bEnablePred** to **HI_TRUE**, **u32Enhance** to **0**, and **u32Base** to **1**. If the 2x frame skipping reference mode is selected, set **bEnablePred** to **HI_TRUE**, **u32Enhance** to **1**, and **u32Base** to **1**. If the 4x frame skipping reference mode is selected, set **bEnablePred** to **HI_TRUE**, **u32Enhance** to **1**, and **u32Base** to **2**.
- If the channel GOP mode is set to **VENC_GOPMODE_DUALP**, **u32SPInterval** can only be set to **(u32Enhance + 1) x u32Base**.
- If the channel GOP mode is set to **VENC_GOPMODE_SMARTP**, **bEnablePred** must be set to **HI_TRUE**.
- If the channel GOP mode is set to **VENC_GOPMODE_BIPREDB**, **u32Enhance** must be equal to **BFrmNum**.

[Example]

None

[See Also]

None



HI_MPI_VENC_GetRefParam

[Description]

Obtains the advanced frame skipping reference parameters for an H.264/H.265 VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetRefParam(VENC_CHN VeChn, VENC_PARAM_REF_S *  
pstRefParam);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstRefParam	Advanced frame skipping reference parameters for an H.264/H.265 VENC channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

If **pstRefParam** is null, an error code indicating failure is returned.

[Example]

None

[See Also]

None

HI_MPI_VENC_SetColor2Grey

[Description]

Enables or disables the color-to-gray function of a VENC channel.

[Syntax]



```
HI_S32 HI_MPI_VENC_SetColor2Grey(VENC_CHN VeChn, const VENC_COLOR2GREY_S*  
pstChnColor2Grey)
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID	Input
pstChnColor2Grey	Parameter for enabling or disabling the color-to-gray function	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to set the color-to-gray attribute of a VENC channel.
- This MPI can be called after a VENC channel is created and before the channel is destroyed. If the MPI is called during encoding, the operation takes effect only after the next I frame is encoded.

[Example]

None

[See Also]

None

HI_MPI_VENC_GetColor2Grey

[Description]

Obtains the enable status of the color-to-gray function of a VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetColor2Grey(VENC_CHN VeChn, VENC_COLOR2GREY_S*  
pstChnColor2Grey)
```

[Parameter]



Parameter	Description	Input/Output
VeChn	VENC channel ID	Input
pstChnColor2Grey	Parameter for enabling or disabling the color-to-gray function	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

If **pstChnColor2Grey** is null, an error code indicating failure is returned.

[Example]

None

[See Also]

None

HI_MPI_VENC_SetCrop

[Description]

Sets the cropping attribute of a VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetCrop(VENC_CHNVeChn, const VENC_CROP_CFG_S  
*pstCropCfg);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID	Input
pstCropCfg	Channel cropping attributes	Input

[Return Value]



Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- The VENC channel crops pictures, and then determines whether to zoom out on pictures after comparing the cropped picture size with the VENC channel size.
- This MPI must be called before a VENC channel is created and before the channel is destroyed.
- Cropping attribute parameters consist of:
 - **bEnable:** This parameter is used to enable or disable the crop function of the channel.
 - **stRect:** The parameter is used to set the crop region attributes including the start point coordinates of the crop region and the crop region size.

[Example]

None

[See Also]

None

HI_MPI_VENC_GetCrop

[Description]

Obtains the cropping attribute of a VENC channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetCrop(VENC_CHN VeChn, VENC_CROP_CFG_S *pstCropCfg);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID	Input
pstCropCfg	Channel cropping attributes	Output

[Return Value]

Return Value	Description
0	Success



Return Value	Description
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

This MPI must be called after a VENC channel is created and before the channel is destroyed.

[Example]

None

[See Also]

None

HI_MPI_VENC_SetJpegSnapMode

[Description]

Sets the snapshot mode of a JPEG snapshot channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetJpegSnapMode(VENC_CHN VeChn, VENC_JPEG_SNAP_MODE_E enJpegSnapMode);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID	Input
enJpegSnapMode	Channel snapshot mode	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h



- Library file: libmpi.a

[Note]

- This MPI must be called after a JPEG VENC channel is created and before the channel is destroyed.
- This MPI applies only to the JPEG VENC channel.
- A JPEG channel can work in either of the following modes:
 - JPEG_SNAP_ALL mode. After the channel starts to receive pictures, all received pictures are encoded.
 - JPEG_SNAP_FLASH mode. After the channel starts to receive pictures, only the pictures that are captured when the camera flash is on are encoded.
- After a JPEG channel is created, it is in JPEG_SNAP_ALL mode by default. To capture pictures only when the camera flash is on, call this MPI to set the JPEG channel mode to JPEG_SNAP_FLASH.
- The JPEG_SNAP_FLASH mode is available only when the front-end camera flash works.

[Example]

None

[See Also]

None

HI_MPI_VENC_GetJpegSnapMode

[Description]

Obtains the snapshot mode of a JPEG snapshot channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetJpegSnapMode (VENC_CHN VeChn, VENC_JPEG_SNAP_MODE_E *penJpegSnapMode);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID	Input
penJpegSnapMode	Channel snapshot mode	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."



[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI applies only to the JPEG VENC channel.
- This MPI must be called after a JPEG VENC channel is created and before the channel is destroyed.

[Example]

None

[See Also]

None

HI_MPI_VENC_SetH265SliceSplit

[Description]

Sets the slice split attribute for H.265 encoding.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetH265SliceSplit(VENC_CHN VeChn, const  
VENC_PARAM_H265_SLICE_SPLIT_S * pstSliceSplit);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID	Input
pstSliceSplit	Slice split parameters for H.265 streams	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to configure the stream split mode of an H.265 VENC channel.



- The slice split attribute depends on the following four parameters:
 - **bSplitEnable**: Indicates whether the slice of the current frame is split.
 - **u32SplitMode**: Indicates the mode of splitting slices. If **u32SplitMode** is set to **0**, slices are split by byte; if **u32SplitMode** is set to **1**, slices are split by LCU line.**u32SplitMode** cannot be set to a value greater than or equal to 2.
u32SplitMode is valid only when **bSplitEnable** is set to **HI_TRUE**.
 - **u32SliceSize**: Indicates the slice size. The definition of **u32SliceSize** varies according to the value of **u32SplitMode**. When **u32SplitMode** is **0**, **u32SliceSize** indicates the average number of bytes in each slice. The encoder starts to encode a picture from its upper left corner in raster sequence (that is, from left to right and from top to bottom) by macroblocks. The size of each slice depends on **u32SliceSize**. The size of encoded slices may be different from the configured value of **u32SliceSize**, and the deviation is always a positive number. When the last slice is encoded and the number of bytes in the remaining LCUs is less than **u32SliceSize**, the LCUs are encoded as a slice.
When **u32SplitMode** is **1**, **u32SliceSize** indicates the number of LCU lines occupied by each slice, and its unit is LCU line. When the last line of the picture is to be encoded and the number of remaining LCU lines is less than **u32SliceSize**, the LCU lines are considered as a packet.
 - **loop_filter_across_slices_enabled_flag**: Indicates whether filtering is performed on the left and top borders of slices.
- This MPI is an advanced interface. You can call it as required. However, you are advised not to call this MPI. **bSplitEnable** is **false** by default.
- This MPI can be called after a VENC channel is created and before the channel is destroyed. If the MPI is called during encoding, the operation takes effect when the next frame is encoded.
- You are advised to call this MPI after creating a channel but before starting encoding. This reduces the number of times of calling this MPI during encoding.
- Before calling this MPI, you are advised to call [HI_MPI_VENC_SetH265SliceSplit](#) to obtain the slice split configurations of the current VENC channel.

[Example]

See the example of [HI_MPI_VENC_SetH264SliceSplit](#).

[See Also]

None

HI_MPI_VENC_GetH265SliceSplit

[Description]

Obtains the slice split attribute for H.265 encoding.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetH265SliceSplit(VENC_CHN VeChn,  
VENC_PARAM_H265_SLICE_SPLIT_S * pstSliceSplit);
```

[Parameter]



Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstSliceSplit	Slice split parameters for H.265 streams	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to obtain the slice split mode of an H.265 VENC channel.
- This MPI can be called after a VENC channel is created and before the channel is destroyed.
- You are advised to call this MPI after creating a channel but before starting encoding. This reduces the number of times of calling this MPI during encoding.

[Example]

See the example of [HI_MPI_VENC_SetH264SliceSplit](#).

[See Also]

None

HI_MPI_VENC_SetH265PredUnit

[Description]

Sets the PU attribute for H.265 encoding.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetH265PredUnit(VENC_CHN VeChn, const  
VENC_PARAM_H265_PU_S *pstPredUnit);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID	Input



Parameter	Description	Input/Output
pstPredUnit	PU configuration of an H.265 VENC channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to configure PU attribute of an H.265 VENC channel.
- The PU attribute depends on the following nine parameters:
 - **bPu32x32En:** Indicates the PU32x32 block enable flag.
 - **bPu16x16En:** Indicates the PU16x16 block enable flag.
 - **bPu8x8En:** Indicates the PU8x8 block enable flag.
 - **bPu4x4En:** Indicates the PU4x4 block enable flag.
 - **constrained_intra_pred_flag:** For details about the parameter definition, see the H.265 protocol.
 - **strong_intra_smoothing_enabled_flag:** For details about the parameter definition, see the H.265 protocol.
 - **pcm_enabled_flag:** For details about the parameter definition, see the H.265 protocol.
 - **pcm_loop_filter_disabled_flag:** For details about the parameter definition, see the H.265 protocol.
 - **u32MaxNumMergeCand:** For details about the parameter definition, see the H.265 protocol.
- This MPI is an advanced interface. You can call it as required. However, you are advised not to call this MPI. The MPI parameters have default values.
- Among **bPu32x32En**, **bPu16x16En**, **bPu8x8En**, and **bPu4x4En**, bPu16x16En and bPu8x8En must be set to HI_TRUE.
- This MPI can be called after a VENC channel is created and before the channel is destroyed. If the MPI is called during encoding, the operation takes effect when the next I frame is encoded.
- You are advised to call this MPI after creating a channel but before starting encoding. This reduces the number of times of calling this MPI during encoding.
- Before calling this MPI, you are advised to call [HI_MPI_VENC_GetH265PredUnit](#) to obtain the PU configurations of the current VENC channel.



[Example]

None

[See Also]

None

HI_MPI_VENC_GetH265PredUnit

[Description]

Obtains the PU attribute for H.265 encoding.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetH265PredUnit(VENC_CHN VeChn, VENC_PARAM_H265_PU_S  
*pstPredUnit);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstPredUnit	PU configuration of an H.265 VENC channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to obtain the PU attribute of an H.265 VENC channel.
- This MPI can be called after a VENC channel is created and before the channel is destroyed.
- You are advised to call this MPI after creating a channel but before starting encoding. This reduces the number of times of calling this MPI during encoding.

[Example]

None

[See Also]



None

HI_MPI_VENC_SetH265Trans

[Description]

Sets the transformation and quantization attribute for H.265 encoding.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetH265Trans(VENC_CHN VeChn, const  
VENC_PARAM_H265_TRANS_S *pstH265Trans);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID	Input
pstH265Trans	Transformation and quantization configurations of an H.265 VENC channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to configure transformation and quantization attribute of an H.265 VENC channel.
- The transformation and quantization attribute depends on the following twenty parameters:
 - transquant_bypass_enabled_flag:** For details about the parameter definition, see the H.265 protocol.
 - transform_skip_enabled_flag:** For details about the parameter definition, see the H.265 protocol.
 - cb_qp_offset:** For details about the parameter definition, see **slice_cb_qp_offset** in the H.265 protocol.
 - cr_qp_offset:** For details about the parameter definition, see **slice_cr_qp_offset** in the H.265 protocol.
 - bScalingListTu4Valid:** indicates whether the **InterScalingList4X4** and **IntraScalingList4X4** quantization tables are valid.



- **InterScalingList4X4 [2][16]:** **InterScalingList4X4 [0][16]** indicates the inter-frame luminance quantization table, and **InterScalingList4X4 [1][16]** indicates the inter-frame chrominance quantization table. Users can provide the quantization tables by using this array.
- **IntraScalingList4X4 [2][16]:** **IntraScalingList4X4 [0][16]** indicates the intra-frame luminance quantization table, and **IntraScalingList4X4 [1][16]** indicates the intra-frame chrominance quantization table. Users can provide the quantization tables by using this array.
- **bScalingListTu8Valid:** indicates whether the **InterScalingList8X8** and **IntraScalingList8X8** quantization tables are valid.
- **InterScalingList8X8 [2][64]:** **InterScalingList8X8 [0][64]** indicates the inter-frame luminance quantization table, and **InterScalingList8X8 [1][64]** indicates the inter-frame chrominance quantization table. Users can provide the quantization tables by using this array.
- **IntraScalingList8X8 [2][64]:** **IntraScalingList8X8 [0][64]** indicates the intra-frame luminance quantization table, and **IntraScalingList8X8 [1][64]** indicates the intra-frame chrominance quantization table. Users can provide the quantization tables by using this array.
- **bScalingListTu16Valid:** indicates whether the **InterScalingList16X16** and **IntraScalingList16X16** quantization tables are valid.
- **InterScalingList16X16 [2][64]:** **InterScalingList16X16 [0][64]** indicates the inter-frame luminance quantization table, and **InterScalingList16X16 [1][64]** indicates the inter-frame chrominance quantization table. Users can provide the quantization tables by using this array.
- **IntraScalingList16X16 [2][64]:** **IntraScalingList16X16 [0][64]** indicates the intra-frame luminance quantization table, and **IntraScalingList16X16 [1][64]** indicates the intra-frame chrominance quantization table. Users can provide the quantization tables by using this array.
- **IntraTu16Dc[2]:** **IntraTu16Dc[0]** indicates the luminance DC value when the size of the intra-frame quantization table is 16 x 16. **IntraTu16Dc[1]** indicates the chrominance DC value when the size of the intra-frame quantization table is 16 x 16. For details, see the H.265 protocol.
- **InterTu16Dc[2]:** **InterTu16Dc[0]** indicates the luminance DC value when the size of the inter-frame quantization table is 16 x 16. **InterTu16Dc[1]** indicates the chrominance DC value when the size of the inter-frame quantization table is 16 x 16. For details, see the H.265 protocol.
- **bScalingListTu32Valid:** indicates whether the **InterScalingList32X32** and **IntraScalingList32X32** quantization tables are valid.
- **InterScalingList32X32 [64]:** indicates the inter-frame luminance quantization table. Users can provide the quantization table by configuring this array.
- **IntraScalingList32X32 [64]:** indicates the intra-frame luminance quantization table. Users can provide the quantization table by configuring this array.
- **IntraTu32Dc:** indicates the luminance DC value when the size of the intra-frame quantization table is 32 x 32. For details, see the H.265 protocol.
- **InterTu32Dc:** indicates the luminance DC value when the size of the inter-frame quantization table is 32 x 32. For details, see the H.265 protocol.
- This MPI is an advanced interface. You can call it as required. However, you are advised not to call this MPI. The MPI parameters have default values.



- This MPI can be called after a VENC channel is created and before the channel is destroyed. If the MPI is called during encoding, the operation takes effect when the next I frame is encoded.
- You are advised to call this MPI after creating a channel but before starting encoding. This reduces the number of times of calling this MPI during encoding.
- Before calling this MPI, you are advised to call [HI_MPI_VENC_GetH265Trans](#) to obtain the transformation and quantization configurations of the current VENC channel.

[Example]

None

[See Also]

None

HI_MPI_VENC_GetH265Trans

[Description]

Obtains the transformation and quantization attribute for H.265 encoding.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetH265Trans(VENC_CHN VeChn, VENC_PARAM_H265_TRANS_S  
*pstH265Trans);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstH265Trans	Transformation and quantization configurations of an H.265 VENC channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]



- This MPI is used to obtain the transformation and quantization attribute of an H.265 VENC channel.
- This MPI can be called after a VENC channel is created and before the channel is destroyed.
- You are advised to call this MPI after creating a channel but before starting encoding. This reduces the number of times of calling this MPI during encoding.

[Example]

None

[See Also]

None

HI_MPI_VENC_SetH265Entropy

[Description]

Sets the entropy encoding attribute of an H.265 channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetH265Entropy (VENC_CHN VeChn, const  
VENC_PARAM_H265_ENTROPY_S *pstH265Entropy);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID	Input
pstH265Entropy	Entropy encoding configuration of an H.265 VENC channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to configure the entropy encoding attribute of an H.265 VENC channel.
- The entropy encoding attribute depends on the **cabac_init_flag** parameter. For details about the parameter definition, see the H.265 protocol.



- This MPI is an advanced interface. You can call it as required. However, you are advised not to call this MPI. The MPI parameters have default values.
- This MPI can be called after a VENC channel is created and before the channel is destroyed. If the MPI is called during encoding, the operation takes effect when the next frame is encoded.
- You are advised to call this MPI after creating a channel but before starting encoding. This reduces the number of times of calling this MPI during encoding.
- Before calling this MPI, you are advised to call [HI_MPI_VENC_GetH265Entropy](#) to obtain the entropy encoding configurations of the current VENC channel.

[Example]

None

[See Also]

None

HI_MPI_VENC_GetH265Entropy

[Description]

Obtains the entropy encoding attribute of an H.265 channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetH265Entropy(VENC_CHN VeChn,  
VENC_PARAM_H265_ENTROPY_S *pstH265Entropy);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstH265Entropy	Entropy encoding configuration of an H.265 VENC channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpapi.a

[Note]



- This MPI is used to obtain the entropy encoding attribute of an H.265 VENC channel.
- This MPI can be called after a VENC channel is created and before the channel is destroyed.
- You are advised to call this MPI after creating a channel but before starting encoding. This reduces the number of times of calling this MPI during encoding.

[Example]

None

[See Also]

None

HI_MPI_VENC_SetH265Dbblk

[Description]

Sets the deblocking attribute for H.265 encoding.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetH265Dbblk (VENC_CHN VeChn, const  
VENC_PARAM_H265_DBULK_S *pstH265Dbblk);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID	Input
pstH265Dbblk	Deblocking configuration of an H.265 VENC channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to configure the deblocking attribute of an H.265 VENC channel.
- The deblocking attribute depends on the following three parameters:
 - **slice_deblocking_filter_disabled_flag**: For details, see the H.265 protocol.
 - **slice_beta_offset_div2**: For details, see the H.265 protocol.



- **slice_tc_offset_div2**: For details, see the H.265 protocol.
- This MPI is an advanced interface. You can call it as required. However, you are advised not to call this MPI. Deblocking is enabled by default. The default values of **slice_deblocking_filter_disabled_flag**, **slice_tc_offset_div2**, and **slice_beta_offset_div2** are **0**.
- If you want to disable deblocking, set **slice_deblocking_filter_disabled_flag** to **1**.
- This MPI can be called after a VENC channel is created and before the channel is destroyed. If the MPI is called during encoding, the operation takes effect when the next frame is encoded.
- You are advised to call this MPI after creating a channel but before starting encoding. This reduces the number of times of calling this MPI during encoding.
- Before calling this MPI, you are advised to call [HI_MPI_VENC_GetH265Dbblk](#) to obtain the deblocking configurations of the current VENC channel.

[Example]

None

[See Also]

None

HI_MPI_VENC_GetH265Dbblk

[Description]

Obtains the deblocking attribute for H.265 encoding.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetH265Dbblk(VENC_CHN VeChn, VENC_PARAM_H265_DBULK_S  
*pstH265Dbblk);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstH265Dbblk	Deblocking configuration of an H.265 VENC channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]



- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to obtain the deblocking attribute of an H.265 VENC channel.
- This MPI can be called after a VENC channel is created and before the channel is destroyed.
- You are advised to call this MPI after creating a channel but before starting encoding. This reduces the number of times of calling this MPI during encoding.

[Example]

None

[See Also]

None

HI_MPI_VENC_SetH265Sao

[Description]

Sets the SAO attribute for H.265 encoding.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetH265Sao(VENC_CHN VeChn, const VENC_PARAM_H265_SAO_S *pstH265Sao);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID	Input
pstH265Sao	SAO configuration of an H.265 VENC channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to configure the SAO attribute of an H.265 VENC channel.



- The SAO attribute depends on the following two parameters:
 - **slice_sao_luma_flag**: Indicates whether SAO filtering is performed on the luminance component of the current slice.
 - **slice_sao_chroma_flag**: Indicates whether SAO filtering is performed on the chrominance component of the current slice.
- This MPI is an advanced interface. You can call it as required. However, you are advised not to call this MPI. The SAO function is enabled by default. The default values of **slice_sao_luma_flag** and **slice_sao_chroma_flag** are **1**.
- This MPI can be called after a VENC channel is created and before the channel is destroyed. If the MPI is called during encoding, the operation takes effect when the next frame is encoded.
- You are advised to call this MPI after creating a channel but before starting encoding. This reduces the number of times of calling this MPI during encoding.
- Before calling this MPI, you are advised to call [HI_MPI_VENC_GetH265Sao](#) to obtain the SAO configurations of the current VENC channel.

[Example]

None

[See Also]

None

HI_MPI_VENC_GetH265Sao

[Description]

Obtains the SAO attribute for H.265 encoding.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetH265Sao(VENC_CHN VeChn, VENC_PARAM_H265_SAO_S
*pstH265Sao);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstH265Sao	SAO configuration of an H.265 VENC channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."



[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to obtain the SAO attribute of an H.265 VENC channel.
- This MPI can be called after a VENC channel is created and before the channel is destroyed.
- You are advised to call this MPI after creating a channel but before starting encoding. This reduces the number of times of calling this MPI during encoding.

[Example]

None

[See Also]

None

HI_MPI_VENC_SetH265Timing

[Description]

Sets the timing attribute of VPS packets for H.265 encoding.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetH265Timing(VENC_CHN VeChn, const  
VENC_PARAM_H265_TIMING_S *pstH265Timing);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID	Input
pstH265Timing	Timing configuration of an H.265 VENC channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]



- This MPI is used to obtain the timing attribute of an H.265 VENC channel.
- The timing attribute depends on the following four parameters:
 - **timing_info_present_flag**: For details, see the H.265 protocol.
 - **num_units_in_tick**: For details, see the H.265 protocol.
 - **time_scale**: For details, see the H.265 protocol.
 - **num_ticks_poc_diff_one**: For details, see the H.265 protocol.
- This MPI is an advanced interface. You can call it as required. However, you are advised not to call this MPI. The timing function is disabled by default. The default value of **timing_info_present_flag** is **0**, the default values of **num_units_in_tick** and **num_ticks_poc_diff_one** are **1**, and the default value of **time_scale** is **60**.
- This MPI can be called after a VENC channel is created and before the channel is destroyed. If the MPI is called during encoding, the operation takes effect when the next I frame is encoded.
- You are advised to call this MPI after creating a channel but before starting encoding. This reduces the number of times of calling this MPI during encoding.
- Before calling this MPI, you are advised to call [HI_MPI_VENC_GetH265Timing](#) to obtain the timing configurations of the current VENC channel.

[Example]

None

[See Also]

None

HI_MPI_VENC_GetH265Timing

[Description]

Obtains the timing attribute of VPS packets for H.265 encoding.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetH265Timing(VENC_CHN VeChn, VENC_PARAM_H265_TIMING_S  
*pstH265Timing);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstH265Timing	Timing configuration of an H.265 VENC channel	Output

[Return Value]

Return Value	Description
0	Success



Return Value	Description
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to obtain the timing attribute of an H.265 VENC channel.
- This MPI can be called after a VENC channel is created and before the channel is destroyed.
- You are advised to call this MPI after creating a channel but before starting encoding. This reduces the number of times of calling this MPI during encoding.

[Example]

None

[See Also]

None

HI_MPI_VENC_SetFrameLostStrategy

[Description]

Sets the frame discarding policies when the instantaneous bit rate is above the threshold.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetFrameLostStrategy(VENC_CHN VeChn, const  
VENC_PARAM_FRAMELOST_S *pstFrmLostParam);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstFrmLostParam	Frame discarding parameters for a VENC channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."



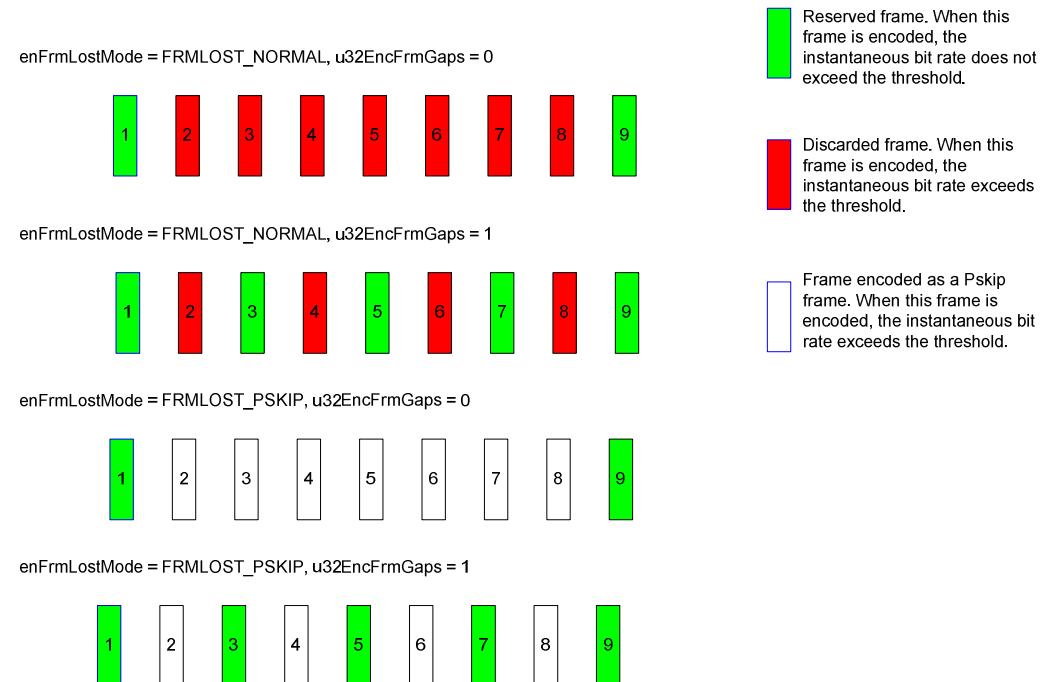
[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- If **pstFrmLostParam** is null, an error code indicating failure is returned.
- **pstFrmLostParam** depends on the following four parameters:
 - **enFrmLostMode**: Indicates the frame discarding mode.
 - **u32EncFrmGaps**: Indicates the frame discarding interval.
 - **bFrmLostOpen**: Indicates the frame discarding enable.
 - **u32FrmLostBpsThr**: Indicates the frame discarding threshold.
- This MPI is an advanced interface. You can call it as required. However, the MPI parameters have default values. Frames are discarded when the instantaneous bit rate is above the threshold by default.
- When the instantaneous bit rate is above the threshold, two processing modes are supported: discarding frames and encoding Pskip frames. Only the processing mode of discarding frames is supported during MJPEG encoding.
- **u32EncFrmGaps** determines whether to discard frames or encode Pskip frames evenly. See [Figure 6-20](#).

Figure 6-20 Frame discarding diagram



- This MPI can be called after a VENC channel is created and before the channel is destroyed. If the MPI is called during encoding, this MPI takes effect when the next frame is encoded. If this MPI is called after the GOP mode is switched, this MPI takes effect after the GOP takes effect.



- The Pskip frame is valid only for the VENC channels of which the GOP mode is set to **VENC_GOPMODE_NORMALP**.
- The frame discarding switch is disabled by default after the GOP mode is switched. To use this MPI after mode switching, reconfigure this MPI.
- If the OSD of the current frame is enabled and updated, the current frame cannot be encoded as a Pskip frame.
- The function of encoding the current frame as a Pskip frame and the function of refreshing the Islice in the P frame are exclusive.
- When the function of encoding Pskip frames is configured in the H.265 channel, SAO filtering is disabled in the module. Therefore, you are advised to call [HI_MPI_VENC_SetH265Sao](#) to disable the SAO function.

[Example]

None

[See Also]

None

HI_MPI_VENC_GetFrameLostStrategy

[Description]

Obtains the frame discarding polices when the instantaneous bit rate is above the threshold.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetFrameLostStrategy(VENC_CHN VeChn,  
VENC_PARAM_FRAMELOST_S *pstFrmLostParam);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstFrmLostParam	Frame discarding parameters for a VENC channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a



[Note]

- If **pstFrmLostParam** is null, an error code indicating failure is returned.
- During the switching of the GOP mode, the obtained attributes are the latest configured attributes.

[Example]

None

[See Also]

None

HI_MPI_VENC_SetSuperFrameCfg

[Description]

Sets the processing mode of jumbo frames for the encoding channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetSuperFrameCfg(VENC_CHN VeChn, const  
VENC_SUPERFRAME_CFG_S *pstSuperFrmParam);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstSuperFrmParam	Configuration parameters of jumbo frames for the encoding channel	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_rc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- If no VENC channel is created, an error code indicating failure is returned.
- This MPI is an advanced interface. You can call it as required. The MPI parameters have the following default values: **enSuperFrmMode = SUPERFRM_NONE**,



**u32SuperIFrmBitsThr = 500000, u32SuperPfrmBitsThr = 500000,
u32SuperBfrmBitsThr = 500000, and enRcPriority = RC_PRIORITY_BITRATE.**

- This MPI can be called after an encoding channel is created and before the channel is destroyed.

[Example]

None

[See Also]

None

HI_MPI_VENC_GetSuperFrameCfg

[Description]

Obtains the processing mode of jumbo frames for the encoding channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetSuperFrameCfg(VENC_CHN VeChn, VENC_SUPERFRAME_CFG_S  
*pstSuperFrmParam);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstSuperFrmParam	Configuration parameters of jumbo frames for the encoding channel	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

If **pstSuperFrmParam** is null, an error code indicating failure is returned.

[Example]

None



[See Also]

None

HI_MPI_VENC_SetChnlPriority

[Description]

Sets the channel priority.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetChnlPriority(VENC_CHN VeChn, HI_U32 u32Priority);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
u32Priority	VENC channel priority Value range: [0, 2)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- If **u32Priority** is greater than 1, an error code indicating failure is returned.
- This MPI is an advanced interface. You can call it as required. The default channel priority is 0.
- This MPI can be called after a VENC channel is created and before the channel is destroyed.

[Example]

None

[See Also]

None



HI_MPI_VENC_GetChnlPriority

[Description]

Obtains the channel priority.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetChnlPriority(VENC_CHN VeChn, HI_U32 *pu32Priority);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pu32Priority	VENC channel priority	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpapi.a

[Note]

If **pu32Priority** is null, an error code indicating failure is returned.

[Example]

None

[See Also]

None

HI_MPI_VENC_GetIntraRefresh

[Description]

Obtains the parameter of refreshing the Islice in the P frame.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetIntraRefresh(VENC_CHN VeChn,  
VENC_PARAM_INTRA_REFRESH_S *pstIntraRefresh)
```



[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID	Input
pstIntraRefresh	Parameter used to control the function of refreshing the Islice in the P frame	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI must be called after a VENC channel is created and before the channel is destroyed.
- During the switching of the GOP mode, the obtained attributes are the latest configured attributes.

[Example]

None

[See Also]

None

HI_MPI_VENC_SetIntraRefresh

[Description]

Sets the parameter of refreshing the Islice in the P frame.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetIntraRefresh(VENC_CHN VeChn,  
VENC_PARAM_INTRA_REFRESH_S *pstIntraRefresh)
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID	Input



Parameter	Description	Input/Output
pstIntraRefresh	Parameter used to control the function of refreshing the Islice in the P frame	Input

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI needs to be called to reconfigure the parameter after GOP is changed.
- This MPI needs to be called to reconfigure the parameter after the advanced frame skipping reference parameter is configured. If **bEnablePred** is set to **0**, the advanced frame skipping reference mode is not supported.
- The number of rows to be refreshed (**u32RefreshLineNum**) must be set to an appropriate value to ensure that Islice refresh can be finished within one GOP cycle. Note that the refresh is implemented only in the **BASE_PSLICE_REFBYBASE** frame at the base layer in advanced frame skipping reference mode. **u32RefreshLineNum** needs to meet the formulas in [Table 6-11](#).

Table 6-11 Requirements that **u32RefreshLineNum** needs to meet

Mode	Formula	Remarks
H.264	$u32RefreshLineNum * Gop \geq (u32PicHeight + 15)/16$	If advanced frame skipping reference is not used: Gop = Gop;
H.265	$u32RefreshLineNum * Gop \geq (u32PicHeight + 63)/64$	If advanced frame skipping reference is used: $Gop = (Gop + (u32Base * (u32Enhance+1) - 1)) / (u32Base * (u32Enhance+1))$

- For the MPI that takes effect when the next I frame is encoded, an I frame is generated after parameters are configured by calling this MPI to ensure that parameter settings take effect.
- This MPI is valid only for the VENC channels of which the GOP mode is set to **VENC_GOPMODE_NORMALP**.
- This MPI automatically becomes invalid after the GOP mode is switched. After the GOP mode is switched from non-NORMALP to NORMALP, calling this MPI takes effect after the GOP takes effect.
- This MPI and the method of discarding frames in Pskip mode are mutually exclusive.

[Example]

None



HI_MPI_VENC_SetModParam

[Description]

Sets the module parameters related to encoding.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetModParam(VENC_PARAM_MOD_S *pstModParam)
```

[Parameter]

Parameter	Description	Input/Output
pstModParam	Pointer to the parameters of the encoding module	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI can be called only before a channel is created. If this MPI is called after a channel is created, the error code [HI_ERR_VENC_NOT_PERM](#) is returned.
- This MPI can be called to set the module parameters [hi35xx_venc.ko](#), [hi35xx_h264e.ko](#), [hi35xx_h265e.ko](#), and [hi35xx_jpge.ko](#).

[Example]

None

[See Also]

None

HI_MPI_VENC_GetModParam

[Description]

Obtains the module parameters related to encoding.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetModParam(VENC_PARAM_MOD_S *pstModParam)
```

[Parameter]



Parameter	Description	Input/Output
pstModParam	Pointer to the parameters of the encoding module	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

This MPI can be called to obtain the module parameters **hi35xx_venc.ko**, **hi35xx_h264e.ko**, **hi35xx_h265e.ko**, and **hi35xx_jpge.ko**.

[Example]

None

[See Also]

None

HI_MPI_VENC_SetSSERegion

[Description]

Sets the SSE attributes of an H.265 channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetSSERegion(VENC_CHN VeChn, const VENC_SSE_CFG_S *  
pstSSECfg);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstSSECfg	Parameter of the SSE region	Input

[Return Value]



Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to configure parameters of the SSE region for an H.265 VENC channel.
- The SSE parameters are determined by the following three parameters:
 - **u32Index**: Each channel can configure at most eight SSE regions. The SSE regions are managed in the system based on the indexes 0–7. **u32Index** indicates the SSE index configured by the user. The SSE regions can be overlapped. When overlapping occurs, each SSE region reports its own sum of mean squared errors (MSEs) of all the blocks in the region, and the SSE regions are independent of each other.
 - **bEnable**: This parameter indicates whether the current SSE region is enabled.
 - **stRect**: This parameter specifies the position coordinate and size of the current SSE region. The coordinates of the start point for the SSE region must fall within the picture range and be 64-pixel-aligned. The width and length of the SSE region must be 64-pixel-aligned. The SSE region must fall within the picture range.
- This MPI is an advanced MPI. The eight SSE regions are disabled by default. You can call this MPI to modify the SSE region and obtain the MSE sum of each region from the watermark information.
- This MPI can be called after a VENC channel is created and before the channel is destroyed. If the MPI is called during encoding, the settings take effect when the next frame is encoded.
- Before calling this MPI, you are advised to call [HI_MPI_VENC_GetSSERegion](#) to obtain the SSE configurations of the current VENC channel.

[Example]

```
HI_S32 SetSSE(HI_VOID)
{
    HI_S32 s32Ret = HI_FAILURE;
    VENC_SSE_CFG_S stSseCfg;
    VENC_CHN_ATTR_S stChnAttr;
    HI_S32 index = 0;
    VENC_CHN VeChnId = 0;
    //...omit other thing
    s32Ret = HI_MPI_VENC_GetChnAttr(VeChnId, &stChnAttr);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_GetChnAttr err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }
}
```



```
    }
    s32Ret = HI_MPI_VENC_GetSseCfg(VeChnId, index, &stSseCfg);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_GetSseCfg err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }
    stSseCfg.bEnable = HI_TRUE;
    stSseCfg.u32Index = 1;
    stSseCfg.stRect.s32X = 0;
    stSseCfg.stRect.s32Y = 0;
    stSseCfg.stRect.u32Width = 64;
    stSseCfg.stRect.u32Height = 64;
    s32Ret = HI_MPI_VENC_SetSSERegion(VeChnId, &stSseCfg);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_SetSseCfg err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }
    return HI_SUCC
ESS;
}
```

[See Also]

None

HI_MPI_VENC_GetSSERegion

[Description]

Obtains the SSE attributes of an H.265 channel.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetSSERegion(VENC_CHN VeChn, HI_U32 u32Index,
VENC_SSE_CFG_S *pstSSECfg);
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
u32Index	Index of the SSE region for an H.265 VENC channel	Input
pstSSECfg	SSE parameter setting	Output



[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to obtain the configuration of the SSE region with a specific index for an H.265 VENC channel.
- This MPI can be called after a VENC channel is created and before the channel is destroyed.
- You are advised to call this MPI after creating a channel but before starting encoding. This reduces the number of times of calling this MPI during encoding.

[Example]

See the example of [HI_MPI_VENC_SetSSERegion](#).

[See Also]

None

HI_MPI_VENC_SetVencAdvancedParam

[Description]

Sets the advanced parameters of the VENC.

[Syntax]

```
HI_S32 HI_MPI_VENC_SetVencAdvancedParam(VENC_CHN  
VeChn, VENC\_PARAM\_ADVANCED\_S *pstAdvancedParam)
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
pstAdvancedParam	Advanced parameters of the VENC	Input

[Return Value]



Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI is used to set the advanced parameters of the VENC.
- The advanced parameters of the VENC are determined by the parameter **u32PollWakeUpFrmCnt**. When the channel obtains streams in timeout or block mode, the specified VENC frame wakes up the block interface after **u32PollWakeUpFrmCnt**.
- This MPI is an advanced MPI. The block interface is woken up once every one frame.
- This MPI can be called after a VENC channel is created and before the channel is destroyed. If the MPI is called during encoding, the settings take effect when the next frame is encoded.
- Before calling this MPI, you are advised to call [HI_MPI_VENC_GetVencAdvancedParam](#) to obtain the configurations of the advanced parameters of the current channel.
- When streams are obtained in block mode, the maximum number of buffered frames (**u32MaxStrmCnt**) is configured in [HI_MPI_VENC_SetMaxStreamCnt](#). It is recommended that **u32PollWakeUpFrmCnt** be less than **u32MaxStrmCnt**.

[Example]

None

[See Also]

None

HI_MPI_VENC_GetVencAdvancedParam

[Description]

Obtains the advanced parameters of the VENC.

[Syntax]

```
HI_S32 HI_MPI_VENC_GetVencAdvancedParam(VENC_CHN  
VeChn, VENC\_PARAM\_ADVANCED\_S *pstAdvancedParam)
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input



Parameter	Description	Input/Output
pstAdvancedParam	Advanced parameters of the VENC	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

None

[Example]

None

[See Also]

None

HI_MPI_VENC_EnableAdvSmartP

[Description]

Enables/Disables the advanced SmartP mode.

[Syntax]

```
HI_S32 HI_MPI_VENC_EnableAdvSmartP (VENC_CHN VeChn, HI_BOOL  
bEnableAdvSmartP)
```

[Parameter]

Parameter	Description	Input/Output
VeChn	VENC channel ID Value range: [0, VENC_MAX_CHN_NUM)	Input
bEnableAdvSmartP	Whether to enable the advanced SmartP mode	Input

[Return Value]



Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 6.5 "Error Codes."

[Requirement]

- Header files: hi_comm_venc.h, mpi_venc.h
- Library file: libmpi.a

[Note]

- This MPI can be called after a VENC channel is created and before the channel is destroyed. If the MPI is called during encoding, the settings take effect when the next frame is encoded.
- When calling this MPI ([HI_MPI_VENC_SetChnAttr](#)) is not called to switch the GOP mode), ensure that the GOP mode is **VENC_GOPMODE_SMARTP**. Otherwise, **HI_ERR_VENC_NOT_SUPPORT** is returned.
- When the AdvSmartP function is enabled, one more VB is allocated in the encoder. The VB size is calculated as follows: VB size = align(MaxPicWidth,16) * align(MaxPicHeight,16) * 3/2 + (align(MaxPicWidth,16)/64 + 1) * (align(MaxPicHeight,16)/64 + 1) * 74 * 16.
- Only H.265 encoding is supported.

[Example]

```
HI_S32 StartVenc(HI_VOID)
{
    HI_S32 s32Ret;
    VI_CHN ViChn = 0;
    VENC_CHN VeChn = 0;
    VENC_CHN_ATTR_S stAttr;
    MPP_CHN_S stSrcChn, stDestChn;
    /* set h265 chnnel video encode attribute */
    stAttr.stVeAttr.enType = PT_H265;
    stAttr.stVeAttr.stAttrH265e.u32PicWidth = u32PicWidth;
    stAttr.stVeAttr.stAttrH265e.u32PicHeight = u32PicHeigh;
    stAttr.stVeAttr.stAttrH265e.u32MaxPicWidth = u32MaxPicWidth;
    stAttr.stVeAttr.stAttrH265e.u32MaxPicHeight = u32MaxPicHeigh;
    stAttr.stVeAttr.stAttrH265e.u32Profile = 0;
    stAttr.stVeAttr.stAttrH265e.u32BufSize = u32MaxPicWidth*
u32MaxPicHeigh:
    stAttr.stVeAttr.stAttrH265e.bByFrame = HI_TRUE;
    ..... // omit other video encode assignments here.
    /* set h265 chnnel rate control attribute */
    stAttr.stRcAttr.enRcMode = VENC_RC_MODE_H265CBR;
    stAttr.stRcAttr.stAttrH265Cbr.u32BitRate = 10*1024;
```



```
stAttr.stRcAttr.stAttrH265Cbr.fr32DstFrmRate      = 30;
stAttr.stRcAttr.stAttrH265Cbr.u32SrcFrmRate       = 30;
stAttr.stRcAttr.stAttrH265Cbr.u32Gop              = 30;
stAttr.stRcAttr.stAttrH265Cbr.u32FluctuateLevel   = 1;
stAttr.stRcAttr.stAttrH265Cbr.u32StatTime         = 1;

stAttr.stGopAttr.enGopMode    = VENC_GOPMODE_SMARTP;
stAttr.stGopAttr.stSmartP.u32BgInterval =
stAttr.stRcAttr.stAttrH265Cbr.u32Gop *10;
stAttr.stGopAttr.stSmartP. s32BgQpDelta= 3;
stAttr.stGopAttr.stSmartP. s32ViQpDelta= 2;

..... // omit other rate control assignments here.

s32Ret = HI_MPI_VENC_CreateChn(VeChn, &stAttr);
if (HI_SUCCESS != s32Ret)
{
    printf("HI_MPI_VENC_CreateChn err 0x%x\n",s32Ret);
    return HI_FAILURE;
}

s32Ret = HI_MPI_VENC_EnableAdvSmartP(VeChn, HI_TRUE);
if (HI_SUCCESS != s32Ret)
{
    printf("HI_MPI_VENC_EnableAdvSmartP err 0x%x\n",s32Ret);
    return HI_FAILURE;
}

s32Ret = HI_MPI_VENC_StartRecvPic(VeChn);
if (s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VENC_StartRecvPic err 0x%x\n",s32Ret);
    return HI_FAILURE;
}
..... // omit other code here.

return HI_SUCCESS;
}
```

[See Also]

None



6.4 Data Structures

VENC data structures are defined as follows:

- [**VENC_MAX_CHN_NUM**](#): Defines the maximum number of channels.
- [**RC_TEXTURE_THR_SIZE**](#): Defines the number of thresholds for bit rate control in texture.
- [**H264E_NALU_TYPE_E**](#): Defines the NALU type of an H.264 stream.
- [**H264E_REFSLICE_TYPE_E**](#): Defines the reference frame type in a specific frame skipping reference mode based on the obtained H.264 streams.
- [**H264E_REF_TYPE_E**](#): Defines the frame type and reference attributes of the H.264 frame skipping reference streams.
- [**H265E_REF_TYPE_E**](#): Defines the frame type and reference attributes of the H.265 frame skipping reference streams.
- [**JPEG_E_PACK_TYPE_E**](#): Defines the Pack type of a JPEG stream.
- [**H265E_NALU_TYPE_E**](#): Defines the NALU type of an H.265 stream.
- [**VENC_DATA_TYPE_U**](#): Defines the stream result type.
- [**VENC_PACK_S**](#): Defines a stream packet.
- [**VENC_STREAM_INFO_H264_S**](#): Defines the features of an H.264 stream.
- [**VENC_STREAM_INFO_JPEG_S**](#): Defines the features of a JPEG/MJPEG stream.
- [**VENC_STREAM_INFO_H265_S**](#): Defines the features of an H.265 stream.
- [**VENC_STREAM_ADVANCE_INFO_H264_S**](#): Defines the advanced features of the H.264 stream.
- [**VENC_STREAM_ADVANCE_INFO_JPEG_S**](#): Defines the advanced features of the JPEG/MJPEG stream.
- [**VENC_SSE_INFO_S**](#): Defines the SSE information of the H.264/H.265 protocol.
- [**VENC_STREAM_ADVANCE_INFO_H265_S**](#): Defines the advanced features of the H.265 stream.
- [**VENC_STREAM_S**](#): Defines the stream frame type.
- [**VENC_STREAM_BUF_INFO_S**](#): Defines the stream buffer information.
- [**VENC_ATTR_H264_S**](#): Defines the attribute of the H.264 encoder.
- [**VENC_ATTR_MJPEG_S**](#): Defines the attribute of the MJPEG encoder.
- [**VENC_ATTR_JPEG_S**](#): Defines the attribute of the JPEG snapshot encoder.
- [**VENC_ATTR_H265_S**](#): Defines the attribute of the H.265 encoder.
- [**VENC_ATTR_S**](#): Defines the encoder attributes.
- [**VENC_GOP_MODE_E**](#): Defines the GOP type for H.264/H.265 encoding.
- [**VENC_GOP_NORMALP_S**](#): Defines the structure of the GOP attribute for encoding the P frame that requires only one reference frame.
- [**VENC_GOP_DUALP_S**](#): Defines the structure of the GOP attribute for encoding the P frame that requires two reference frames.
- [**VENC_GOP_SMARTP_S**](#): Defines the structure of the GOP attribute for encoding the smart P frame.
- [**VENC_GOP_BIPREDB_S**](#): Defines the structure of the GOP attribute for encoding the B frame.
- [**VENC_GOP_ATTR_S**](#): Defines the structure of the GOP attribute for the encoder.



- [**VENC_CHN_ATTR_S**](#): Defines the attribute of a VENC channel.
- [**VENC_CHN_STAT_S**](#): Defines the status of a VENC channel.
- [**VENC_PARAM_H264_SLICE_SPLIT_S**](#): Defines the slice attribute of an H.264 channel.
- [**VENC_PARAM_H264_INTER_PRED_S**](#): Defines the inter-prediction attribute of an H.264 channel.
- [**VENC_PARAM_H264_INTRA_PRED_S**](#): Defines the intra-prediction attribute of an H.264 channel.
- [**VENC_PARAM_H264_TRANS_S**](#): Defines the transformation and quantization attribute of an H.264 channel.
- [**VENC_PARAM_H264_ENTROPY_S**](#): Defines the entropy attribute of an H.264 channel.
- [**VENC_PARAM_H264_POC_S**](#): Defines the POC attribute of an H.264 channel.
- [**VENC_PARAM_H264_DBLK_S**](#): Defines the de-blocking attribute of an H.264 channel.
- [**VENC_PARAM_H264_VUI_S**](#): Defines the VUI attribute of an H.264 channel.
- [**VENC_PARAM_H265_VUI_S**](#): Defines the VUI structure of an H.265 VENC channel.
- [**VENC_PARAM_VUI_ASPECT_RATIO_S**](#): Defines the AspectRatio information in the VUI of an H.264/H.265 VENC channel.
- [**VENC_PARAM_VUI_H264_TIME_INFO_S**](#): Defines the Time_Info information in the VUI of an H.264 encoding channel.
- [**VENC_PARAM_VUI_H265_TIME_INFO_S**](#): Defines the Time_Info information in the VUI of an H.265 VENC channel.
- [**VENC_PARAM_VUI_VIDEO_SIGNAL_S**](#): Defines the Video_Signal information in the VUI of an H.264/H.265 VENC channel.
- [**VENC_PARAM_VUI_BITSTREAM_RESTRIC_S**](#): Defines the Bitstream_Restriction structure in the VUI of an H.264/H.265 VENC channel.
- [**VENC_PARAM_JPEG_S**](#): Defines the parameters of the JPEG encoder.
- [**VENC_PARAM_MJPEG_S**](#): Defines the advanced parameters of the MJPEG encoder.
- [**VENC_ROI_CFG_S**](#): Defines an ROI for a VENC channel.
- [**VENC_ROIBG_FRAME_RATE_S**](#): Defines the frame rate of a non-ROI.
- [**VENC_PARAM_REF_S**](#): Defines the advanced frame skipping reference parameters for H.264/H.265 encoding.
- [**VENC_RC_ATTR_S**](#): Defines RC attributes for controlling the bit rate of a VENC channel.
- [**VENC_RC_MODE_E**](#): Defines the mode of the RC for controlling the bit rate of a VENC channel.
- [**VENC_RC_QPMAP_MODE_E**](#): Defines the method of assigning the QP values for CU32 and CU64 when the VENC channel uses the QpMap mode.
- [**VENC_ATTR_H264_CBR_S**](#): Defines the CBR attribute structure of an H.264 encoding channel.
- [**VENC_ATTR_H264_VBR_S**](#): Defines the VBR attribute structure of an H.264 encoding channel.
- [**VENC_ATTR_H264_AVBR_S**](#): Defines the AVBR attribute structure of an H.264 encoding channel.
- [**VENC_ATTR_H264_FIXQP_S**](#): Defines the FixQP attribute of an H.264 channel.



- **VENC_ATTR_H264_QPMAP_S:** Defines the structure of the QpMap attribute for an H.264 encoding channel.
- **VENC_ATTR_MJPEG_FIXQP_S:** Defines the FixQp attribute of an MJPEG channel.
- **VENC_ATTR_MJPEG_CBR_S:** Defines the CBR attribute of an MJPEG channel.
- **VENC_ATTR_MJPEG_VBR_S:** Defines the VBR attribute of an MJPEG channel.
- **VENC_ATTR_H265_CBR_S:** Defines the CBR attribute structure of an H.265 encoding channel.
- **VENC_ATTR_H265_VBR_S:** Defines the VBR attribute structure of an H.265 encoding channel.
- **VENC_ATTR_H265_AVBR_S:** Defines the AVBR attribute structure of an H.265 VENC channel.
- **VENC_ATTR_H265_QPMAP_S:** Defines the structure of the QpMap attribute for an H.265 encoding channel.
- **VENC_SUPERFRM_MODE_E:** Defines the mode of processing jumbo frames when the bit rate is being controlled.
- **VENC_PARAM_H264_VBR_S:** Defines the configurations of the advanced parameters related to the VBR control mode of an H.264 encoding channel.
- **VENC_PARAM_H264_AVBR_S:** Defines the configurations of the advanced parameters related to the AVBR control mode of an H.264 encoding channel.
- **VENC_PARAM_H264_CBR_S:** Defines the configurations of the advanced parameters related to the new CBR control mode of an H.264 encoding channel.
- **VENC_PARAM_MJPEG_CBR_S:** Defines the configurations of the advanced parameters related to the CBR control mode of an MJPEG channel.
- **VENC_PARAM_MJPEG_VBR_S:** Defines the configurations of the advanced parameters related to the VBR control mode of an MJPEG channel.
- **VENC_PARAM_H265_VBR_S:** Defines the configurations of the advanced parameters related to the VBR control mode of an H.265 encoding channel.
- **VENC_PARAM_H265_AVBR_S:** Defines the configurations of the advanced parameters related to the AVBR control mode of an H.265 encoding channel.
- **VENC_PARAM_H265_CBR_S:** Defines the configurations of the advanced parameters related to the CBR control mode of an H.265 encoding channel.
- **VENC_RC_PARAM_S:** Defines the advanced parameters for controlling the bit rate of a VENC channel.
- **VENC_CROP_CFG_S:** Defines the crop parameters of a VENC channel.
- **VENC_FRAME_RATE_S:** Defines the parameters for controlling the frame rate of a VENC channel.
- **VENC_COLOR2GREY_S:** Defines color-to-gray parameters.
- **VENC_JPEG_SNAP_MODE_E:** Defines the snapshot mode of a JPEG VENC channel.
- **VENC_RECV_PIC_PARAM_S:** Defines the number of received frames to be encoded.
- **H264E_IDR_PIC_ID_MODE_E:** Defines the mode of setting the idr_pic_id of an IDR frame or I frame.
- **VENC_H264_IDRPICID_CFG_S:** Defines the parameters for setting the idr_pic_id of an IDR frame or I frame.
- **VENC_PARAM_H265_SLICE_SPLIT_S:** Defines the slice split attribute of an H.265 VENC channel.



- [VENC_PARAM_H265_PU_S](#): Defines the PU attribute of an H.265 VENC channel.
- [VENC_PARAM_H265_TRANS_S](#): Defines the transformation and quantization attribute of an H.265 VENC channel.
- [VENC_PARAM_H265_ENTROPY_S](#): Defines the entropy encoding attribute of an H.265 VENC channel.
- [VENC_PARAM_H265_DBLK_S](#): Defines the deblocking attribute of an H.265 VENC channel.
- [VENC_PARAM_H265_SAO_S](#): Defines the SAO attribute of an H.265 VENC channel.
- [VENC_PARAM_H265_TIMING_S](#): Defines the timing attribute of an H.265 VENC channel.
- [VENC_FRAMELOST_MODE_E](#): Defines the frame discarding mode when the instantaneous bit rate of a VENC channel is above the threshold.
- [VENC_PARAM_FRAMELOST_S](#): Defines the frame discarding policy when the instantaneous bit rate of a VENC channel is above the threshold.
- [VENC_RC_PRIORITY_E](#): Defines the RC priority.
- [VENC_SUPERFRAME_CFG_S](#): Defines the parameters for jumbo frame processing modes.
- [VENC_PARAM_INTRA_REFRESH_S](#): Defines the parameter for controlling I macroblock/Islice refresh in the P frame.
- [VENC_PARAM_MOD_S](#): Defines the module parameters related to encoding.
- [VENC_MODTYPE_E](#): Defines the type of the module parameters related to encoding.
- [VENC_PARAM_MOD_VENC_S](#): Defines the module parameter for **hi35xx_venc.ko**.
- [VENC_PARAM_MOD_H264E_S](#): Defines the module parameter for **hi35xx_h264e.ko**.
- [VENC_PARAM_MOD_H265E_S](#): Defines the module parameter for **hi35xx_h265.ko**.
- [VENC_PARAM_MOD_JPEG_E_S](#): Defines the module parameter for **hi35xx_jpeg.ko**.
- [USER_RC_INFO_S](#): Defines the data structure for bit rate control.
- [USER_FRAME_INFO_S](#): Defines the data structure of the pictures sent by users.
- [VENC_SSE_CFG_S](#): Defines SSE region parameters.
- [VENC_PARAM_ADVANCED_S](#): Defines the advanced parameters of the VENC.

VENC_MAX_CHN_NUM

[Description]

Defines the maximum number of channels.

[Syntax]

```
#define VENC_MAX_CHN_NUM 16
```

[Note]

None

[See Also]

None



RC_TEXTURE_THR_SIZE

[Description]

Defines the number of thresholds for bit rate control in texture.

[Syntax]

```
#define RC_TEXTURE_THR_SIZE 16
```

[Note]

None

[See Also]

None

H264E_NALU_TYPE_E

[Description]

Defines the NALU type of an H.264 stream.

[Syntax]

```
typedef enum hiH264E_NALU_TYPE_E
{
    H264E_NALU_BSLICE = 0,
    H264E_NALU_PSLICE = 1,
    H264E_NALU_ISLICE = 2,
    H264E_NALU_IDRSLICE = 5,
    H264E_NALU_SEI = 6,
    H264E_NALU_SPS = 7,
    H264E_NALU_PPS = 8,
    H264E_NALU_BUTT
} H264E_NALU_TYPE_E;
```

[Member]

Member	Description
H264E_NALU_BSLICE	BSLICE
H264E_NALU_PSLICE	PSLICE
H264E_NALU_ISLICE	PSLICE. The frame type is P frame.
H264E_NALU_IDRSLICE	ISLICE. The frame type is IDR frame.
H264E_NALU_SEI	SEI
H264E_NALU_SPS	SPS
H264E_NALU_PPS	PPS



[Note]

None

[See Also]

None

H264E_REFSLICE_TYPE_E

[Description]

Defines the reference frame type in a specific frame skipping reference mode based on the obtained H.264 streams.

[Syntax]

```
typedef enum hiH264E_REFSLICE_TYPE_E
{
    H264E_REFSLICE_FOR_1X      = 1,
    H264E_REFSLICE_FOR_2X      = 2,
    H264E_REFSLICE_FOR_4X      = 5,
    H264E_REFSLICE_FOR_BUTT
} H264E_REFSLICE_TYPE_E;
```

[Member]

Member	Description
H264E_REFSLICE_FOR_1X	Reference frame in 1x frame skipping reference mode
H264E_REFSLICE_FOR_2X	Reference frame in 2x frame skipping reference mode or reference frame in 4x frame skipping reference mode for 2x frame skipping reference
H264E_REFSLICE_FOR_4X	Reference frame in 4x frame skipping reference mode
H264E_REFSLICE_BUTT	Non-reference mode

[Note]

None

[See Also]

None

H264E_REF_TYPE_E

[Description]

Defines the frame type and reference attributes of the H.264 frame skipping reference streams.

[Syntax]

```
typedef enum hiH264E_REF_TYPE_E
```



```
{  
    BASE_IDRSLICE = 0,                      //IDR frame at the base layer  
    BASE_PSLICE_REFTOIDR,                   //P frame at the base layer, referenced  
    by other frames at the base layer and references only IDR frames  
    BASE_PSLICE_REFBYBASE,                  //P frame at the base layer,  
    referenced by other frames at the base layer  
    BASE_PSLICE_REFBYENHANCE,              //P frame at the base layer,  
    referenced by frames at the enhance layer  
    ENHANCE_PSLICE_REFBYENHANCE,          //P frame at the enhance layer,  
    referenced by other frames at the enhance layer  
    ENHANCE_PSLICE_NOTFORREF,             //P frame at the enhance layer, not  
    referenced  
    ENHANCE_PSLICE_BUTT  
} H264E_REF_TYPE_E;
```

[Member]

Member	Description
BASE_IDRSLICE	IDR frame at the base layer
BASE_PSLICE_REFTOIDR	P frame at the base layer, referenced by other frames at the base layer and only refers to IDR frames
BASE_PSLICE_REFBYBASE	P frame at the base layer, referenced by other frames at the base layer
BASE_PSLICE_REFBYENHANCE	P frame at the base layer, referenced by other frames at the enhance layer
ENHANCE_PSLICE_REFBYENHANCE	P frame at the enhance layer, referenced by other frames at the enhance layer
ENHANCE_PSLICE_NOTFORREF	P frame at the enhance layer, not referenced

[Note]

None

[See Also]

None

H265E_REF_TYPE_E

[Description]

Defines the frame type and reference attributes of the H.265 frame skipping reference streams.

[Syntax]

```
TypeDef enum hiH264E_REF_TYPE_E H265E_REF_TYPE_E;
```



[Member]

See [H264E_REF_TYPE_E](#).

[Note]

None

[See Also]

[H264E_REF_TYPE_E](#)

JPEGE_PACK_TYPE_E

[Description]

Defines the PACK type of a JPEG stream.

[Syntax]

```
typedef enum hiJPEGE_PACK_TYPE_E
{
    JPEGE_PACK_ECS = 5,
    JPEGE_PACK_APP = 6,
    JPEGE_PACK_VDO = 7,
    JPEGE_PACK_PIC = 8,
    JPEGE_PACK_BUTT
} JPEGE_PACK_TYPE_E;
```

[Member]

Member	Description
JPEGE_PACK_ECS	ECS
JPEGE_PACK_APP	APP
JPEGE_PACK_VDO	VDO
JPEGE_PACK_PIC	PIC

[Note]

None

[See Also]

None

H265E_NALU_TYPE_E

[Description]

Defines the NALU type of an H.265 stream.

[Syntax]



```
typedef enum hiH265E_NALU_TYPE_E
{
    H265E_NALU_BSLICE = 0,
    H265E_NALU_PSLICE = 1,
    H265E_NALU_ISLICE = 2,
    H265E_NALU_IDRSLICE = 19,
    H265E_NALU_VPS     = 32,
    H265E_NALU_SPS     = 33,
    H265E_NALU_PPS     = 34,
    H265E_NALU_SEI     = 39,
    H265E_NALU_BUTT
} H265E_NALU_TYPE_E;
```

[Member]

Member	Description
H265E_NALU_BSLICE	BSLICE
H265E_NALU_PSLICE	PSLICE type
H265E_NALU_ISLICE	ISLICE type. The frame type is P frame.
H265E_NALU_IDRSLICE	ISLICE type. The frame type is IDR frame.
H265E_NALU_VPS	VPS type
H265E_NALU_SPS	SPS type
H265E_NALU_PPS	PPS type
H265E_NALU_SEI	SEI type

[Note]

None

[See Also]

None

VENC_DATA_TYPE_U

[Description]

Defines the stream result type.

[Syntax]

```
typedef union hivENC_DATA_TYPE_U
{
    H264E_NALU_TYPE_E   enH264EType;
    JPEGE_PACK_TYPE_E   enJPEGEType;
    H265E_NALU_TYPE_E   enH265EType;
```



```
}VENC_DATA_TYPE_U;
```

[Member]

Member	Description
enH264EType	Type of an H.264 stream packet
enJPEGEType	Type of a JPEG stream packet
enH265EType	Type of the H.265 stream packet

[Note]

None

[See Also]

- [H264E_NALU_TYPE_E](#)
- [JPEGE_PACK_TYPE_E](#)
- [H265E_NALU_TYPE_E](#)

VENC_PACK_INFO_S

[Description]

Defines the structure of other types of stream packet data which is contained in the current stream packet data.

[Syntax]

```
typedef struct hivENC_PACK_INFO_S
{
    VENC_DATA_TYPE_U u32PackType;
    HI_U32 u32PackOffset;
    HI_U32 u32PackLength;
} VENC_PACK_INFO_S;
```

[Member]

Member	Description
u32PackType	Type of other stream packets in the current stream packet data
u32PackOffset	Offset of other types of stream packet data in the current stream packet data
u32PackLength	Size of other types of stream packet data in the current stream packet data

VENC_PACK_S

[Description]



Defines a stream packet.

[Syntax]

```
typedef struct hiVENC_PACK_S
{
    HI_U32             u32PhyAddr;
    HI_U8              *pu8Addr;
    HI_U32             u32Len;
    HI_U64             u64PTS;
    HI_BOOL            bFrameEnd;
    VENC_DATA_TYPE_U   DataType;
    HI_U32             u32Offset;
    HI_U32             u32DataNum;
    VENC_PACK_INFO_S   stPackInfo[8];
} VENC_PACK_S;
```

[Member]

Member	Description
pu8Addr	Initial address of a stream packet
u32PhyAddr	Physical address of a stream packet
u32Len	Length of a stream packet
DataType	Type of a stream. H.264, JPEG, and MPEG-4 packets are supported.
u64PTS	Time stamp, in μ s
bFrameEnd	Flag of ending a frame Value range: HI_TRUE : the last stream packet of a frame HI_FALSE : the stream packet is not the last one of a field
u32Offset	Valid data in the stream packet and offset of the start address pu8Addr for stream packet
u32DataNum	Number of other stream packets contained in the current stream packet (the type of the current stream packet is specified by DataType data)
stPackInfo[8]	Information about other types of stream packet data in the current stream packet data

[Note]

None

[See Also]

[VENC_DATA_TYPE_U](#)



VENC_STREAM_INFO_H264_S

[Description]

Defines the features of an H.264 stream.

[Syntax]

```
typedef struct hiVENC_STREAM_INFO_H264_S
{
    HI_U32 u32PicBytesNum;
    HI_U32 u32Inter16x16MbNum;
    HI_U32 u32Inter8x8MbNum;
    HI_U32 u32Intra16MbNum;
    HI_U32 u32Intra8MbNum;
    HI_U32 u32Intra4MbNum;
    H264E_REFSLICE_TYPE_E enRefSliceType;
    H264E_REF_TYPE_E     enRefType;
    HI_U32 u32UpdateAttrCnt;
    HI_U32 u32StartQp;
    HI_U32 u32MeanQp;
}VENC_STREAM_INFO_H264_S;
```

[Member]

Member	Description
u32PicBytesNum	Number of bytes to be encoded in the current frame
u32Inter16x16MbNum	Number of macroblocks in 16x16 inter-prediction encoding mode to be encoded in the current frame
u32Inter8x8MbNum	Number of macroblocks in 8x8 inter-prediction encoding mode to be encoded in the current frame
u32Intra16MbNum	Number of macroblocks in intra16 prediction encoding mode to be encoded in the current frame
u32Intra8MbNum	Number of macroblocks in intra8 prediction encoding mode to be encoded in the current frame
u32Intra4MbNum	Number of macroblocks in intra4 prediction encoding mode to be encoded in the current frame
enRefSliceType	Encode the frame whose reference frame is in a specific frame skipping reference mode
enRefType	Type of the encoded frame in advanced frame skipping reference mode
u32UpdateAttrCnt	Number of times that channel attributes or parameters (including RC parameter) are configured
u32StartQp	Start QP value of the frame that is being encoded



Member	Description
u32MeanQp	Mean QP value of the frame that is being encoded

[Note]

You can store only the H.264 streams whose reference frames are in a specific frame skipping reference mode. The details are as follows:

- In 1x frame skipping reference mode, you can store only the streams whose **enRefSliceType** value is **H264E_REFSLICE_FOR_1X**.
- In 2x frame skipping reference mode, you can store only the streams whose **enRefSliceType** value is **H264E_REFSLICE_FOR_2X**.
- In 4x frame skipping reference mode, you can store only the streams whose **enRefSliceType** value is **H264E_REFSLICE_FOR_4X** or store the streams whose **enRefSliceType** value is **H264E_REFSLICE_FOR_2X** or **H264E_REFSLICE_FOR_4X**.

[See Also]

- [VENC_DATA_TYPE_U](#)
- [H264E_REFSLICE_TYPE_E](#)

VENC_STREAM_INFO_JPEG_S

[Description]

Defines the features of a JPEG/MJPEG stream.

[Syntax]

```
typedef struct hiVENC_STREAM_INFO_JPEG_S
{
    HI_U32 u32PicBytesNum;
    HI_U32 u32UpdateAttrCnt;
    HI_U32 u32Qfactor;
} VENC_STREAM_INFO_JPEG_S;
```

[Member]

Member	Description
u32PicBytesNum	Size of a JPEG stream, in byte
u32UpdateAttrCnt	Number of times that channel attributes or parameters (including RC parameter) are set
u32Qfactor	Qfactor value of the currently encoded frame

[Note]

None



[See Also]

[VENC_DATA_TYPE_U](#)

VENC_STREAM_INFO_H265_S

[Description]

Defines the features of an H.265 stream.

[Syntax]

```
typedef struct hiVENC_STREAM_INFO_H265_S
{
    HI_U32 u32PicBytesNum;
    HI_U32 u32Inter64x64CuNum;
    HI_U32 u32Inter32x32CuNum;
    HI_U32 u32Inter16x16CuNum;
    HI_U32 u32Inter8x8CuNum;
    HI_U32 u32Intra32x32CuNum;
    HI_U32 u32Intra16x16CuNum;
    HI_U32 u32Intra8x8CuNum;
    HI_U32 u32Intra4x4CuNum;
    H265E_REF_TYPE_E    enRefType;
    HI_U32 u32UpdateAttrCnt;
    HI_U32 u32StartQp;
    HI_U32 u32MeanQp;
} VENC_STREAM_INFO_H265_S;
```

[Member]

Member	Description
u32PicBytesNum	Number of bytes to be encoded in the current frame
u32Inter64x64CuNum	Number of coding units (CUs) to be encoded in inter64x64 prediction mode in the current frame
u32Inter32x32CuNum	Number of CUs to be encoded in inter32x32 prediction mode in the current frame
u32Inter16x16CuNum	Number of CUs to be encoded in inter16x16 prediction mode in the current frame
u32Inter8x8CuNum	Number of CUs to be encoded in inter8x8 prediction mode in the current frame
u32Intra32x32CuNum	Number of CUs to be encoded in intra32x32 prediction mode in the current frame
u32Intra16x16CuNum	Number of CUs to be encoded in intra16x16 prediction mode in the current frame



Member	Description
u32Intra8x8CuNum	Number of CUs to be encoded in intra8x8 prediction mode in the current frame
u32Intra4x4CuNum	Number of CUs to be encoded in intra4x4 prediction mode in the current frame
enRefType	Type of the encoded frame in advanced frame skipping reference mode
u32UpdateAttrCnt	Number of times that channel attributes or parameters (including RC parameter) are configured
u32StartQp	Start QP value of the frame that is being encoded
u32MeanQp	Mean QP value of the frame that is being encoded

[Note]

For details about **enRefType**, see [VENC_STREAM_INFO_H264_S](#).

[See Also]

- [VENC_DATA_TYPE_U](#)
- [H265E_REF_TYPE_E](#)

VENC_STREAM_ADVANCE_INFO_H264_S

[Description]

Defines the advanced features of the H.264 stream.

[Syntax]

```
typedef struct hiVENC_STREAM_ADVANCE_INFO_H264_S
{
    HI_U32 u32ResidualBitNum;
    HI_U32 u32HeadBitNum;
    HI_U32 u32MadiVal;
    HI_U32 u32MadpVal;
    HI_DOUBLE     dPSNRVal;      //PSNR
    HI_U32 u32MseLcuCnt;
    HI_U32 u32MseSum;
    VENC_SSE_INFO_S stSSEInfo[8];
    HI_U32 u32QpHstgrm[52];
} VENC_STREAM_ADVANCE_INFO_H264_S;
```

[Member]

Member	Description
u32ResidualBitNum	Number of residuals (in bit) of the current VENC frame



Member	Description
u32HeadBitNum	Number of pieces of header information (in bit) of the current VENC frame
u32MadiVal	Madi value of the spatial-domain texture complexity of the current VENC frame
u32MadpVal	Madp value of the time-domain motion complexity of the current VENC frame
dPSNRVal	Peak signal to noise ratio (PSNR) of the current VENC frame
u32MseLcuCnt	Number of LCUs in the current VENC frame
u32MseSum	MSE of the current VENC frame
stSSEInfo[8]	SSE of the eight regions in the current VENC frame
u32QpHstgrm[52]	QP histogram of the current VENC frame

[Note]

None

[See Also]

[HI_MPI_VENC_GetStream](#)

VENC_STREAM_ADVANCE_INFO_JPEG_S

[Description]

Defines the advanced features of the JPEG/MJPEG stream.

[Syntax]

```
typedef struct hiVENC_STREAM_ADVANCE_INFO_JPEG_S
{
    HI_U32 u32Reserved;
} VENC_STREAM_ADVANCE_INFO_JPEG_S;
```

[Member]

Member	Description
u32Reserved	Reserved

[Note]

None

[See Also]

[HI_MPI_VENC_GetStream](#)



VENC_SSE_INFO_S

[Description]

Defines the SSE information of the H.264/H.265 protocol.

[Syntax]

```
typedef struct hiVENC_SSE_INFO_S
{
    HI_BOOL bSSEEEn;
    HI_U32 u32SSEVal;
}VENC_SSE_INFO_S;
```

[Member]

Member	Description
bSSEEEn	Region SSE enable
u32SSEVal	Region SSE value

[Note]

None

[See Also]

- [VENC_STREAM_ADVANCE_INFO_H265_S](#)
- [HI_MPI_VENC_GetStream](#)

VENC_STREAM_ADVANCE_INFO_H265_S

[Description]

Defines the advanced features of the H.265 stream.

[Syntax]

```
typedef struct hiVENC_STREAM_ADVANCE_INFO_H265_S
{
    HI_U32 u32ResidualBitNum;
    HI_U32 u32HeadBitNum;
    HI_U32 u32MadiVal;
    HI_U32 u32MadpVal;
    HI_DOUBLE dPSNRVal;
    HI_U32 u32MseLcuCnt;
    HI_U32 u32MseSum;
    VENC_SSE_INFO_S stSSEInfo[8];
    HI_U32 u32QpHstgrm[52];
}VENC_STREAM_ADVANCE_INFO_H265_S;
```

[Member]



Member	Description
u32ResidualBitNum	Number of residuals (in bit) of the current VENC frame
u32HeadBitNum	Number of pieces of header information (in bit) of the current VENC frame
u32MadiVal	Madi value of the spatial-domain texture complexity of the current VENC frame
u32MadpVal	Madp value of the time-domain motion complexity of the current VENC frame
dPSNRVal	PSNR of the current VENC frame
u32MseLcuCnt	Number of LCUs in the current VENC frame
u32MseSum	MSE of the current VENC frame
stSSEInfo[8]	SSE of the eight regions in the current VENC frame
u32QpHstgrm[52]	QP histogram of the current VENC frame

[Note]

None

[See Also]

[HI_MPI_VENC_GetStream](#)

VENC_STREAM_S

[Description]

Defines the stream frame type.

[Syntax]

```
typedef struct hiVENC_STREAM_S
{
    VENC_PACK_S *pstPack;
    HI_U32       u32PackCount;
    HI_U32       u32Seq;
    union
    {
        VENC_STREAM_INFO_H264_S stH264Info;
        VENC_STREAM_INFO_JPEG_S stJpegInfo;
        VENC_STREAM_INFO_H265_S stH265Info;
    };
    union
    {
        VENC_STREAM_ADVANCE_INFO_H264_S stAdvanceH264Info;
        VENC_STREAM_ADVANCE_INFO_JPEG_S stAdvanceJpegInfo;
    };
}
```



```
VENC_STREAM_ADVANCE_INFO_H265_S stAdvanceH265Info;  
};  
}VENC_STREAM_S
```

[Member]

Member	Description
pstPack	Structure of a stream frame
u32PackCount	Number of stream packets per frame
u32Seq	Sequence number of a stream The sequence number of a frame is obtained by frame and the sequence number of a packet is obtained by packet.
stH264Info/stJpegInfo/stH265Info	Features of a stream
stAdvanceH264Info/stAdvanceJpegInfo/stAdvanceH265Info	Members not supported by Hi3519 V100

[Note]

None

[See Also]

- [VENC_PACK_S](#)
- [HI_MPI_VENC_GetStream](#)

VENC_STREAM_BUF_INFO_S

[Description]

Defines the stream buffer information.

[Syntax]

```
typedef struct hiVENC_STREAM_BUF_INFO_S  
{  
    HI_U32    u32PhyAddr;  
    HI_VOID *pUserAddr;  
    HI_U32    u32BufSize;  
} VENC_STREAM_BUF_INFO_S;
```

[Member]

Member	Description
u32PhyAddr	Start physical address for a stream buffer
pUserAddr	Virtual address for a stream buffer



Member	Description
u32BufSize	Stream buffer size

[Note]

None

[See Also]

[HI_MPI_VENC_GetStreamBufInfo](#)

VENC_ATTR_H264_S

[Description]

Defines the attribute structure for H.264 encoding.

[Syntax]

```
typedef struct hivenc_attr_h264_s
{
    HI_U32    u32MaxPicWidth;
    HI_U32    u32MaxPicHeight;
    HI_U32    u32BufSize;
    HI_U32    u32Profile;
    HI_BOOL   bByFrame;
    HI_U32    u32PicWidth;
    HI_U32    u32PicHeight;
}VENC_ATTR_H264_S;
```

[Member]

Member	Description
u32MaxPicWidth	Maximum width of a picture to be encoded, in pixel Value range: [MIN_WIDTH, MAX_WIDTH] The value must be an integral multiple of MIN_ALIGN . Static attribute
u32PicWidth	Width of a picture to be encoded, in pixel Value range: [MIN_WIDTH, MAX_WIDTH] The value must be an integral multiple of MIN_ALIGN .
u32MaxPicHeight	Maximum height of a picture to be encoded, in pixel Value range: [MIN_WIDTH, MAX_HEIGHT] The value must be an integral multiple of MIN_ALIGN . Static attribute
u32PicHeight	Height of a picture to be encoded, in pixel



Member	Description
	<p>Value range: [MIN_HEIGHT, MAX_HEIGHT] The value must be an integral multiple of MIN_ALIGN.</p>
u32BufSize	<p>Stream buffer size, in byte Value range: [Min, Max] The value must be an integral multiple of 64. You are advised to set the buffer size to the maximum size of a picture to be encoded. That is, the recommended value is as follows: u32MaxPicWidth x u32MaxPicHeight x 1.5 bytes Static attribute The module parameter H264eMiniBufMode can be configured when the .ko driver is loaded to select the mode for limiting the value range.</p> <ul style="list-style-type: none">• H264eMiniBufMode = 0<ul style="list-style-type: none">Minimum value: 1/2 of the maximum size of a picture to be encodedMaximum value: There is no limitation but much memory is consumed if the buffer size is large.• H264eMiniBufMode = 1 (the user needs to ensure that the buffer size is appropriate)<ul style="list-style-type: none">Minimum value: 32 x 1024 bytesMaximum value: There is no limitation but much memory is consumed if the buffer size is large.
bByFrame	<p>Obtaining streams by frame/packet Value range: [HI_TRUE, HI_FALSE]</p> <ul style="list-style-type: none">• HI_TRUE: obtain streams by frame• HI_FALSE: obtain streams by packet <p>Static attribute</p>
u32Profile	<p>Encoding level Value range: [0, 3]</p> <p>0: baseline 1: main profile 2: high profile 3: svc-t</p>

[Note]

None

[See Also]

None



VENC_ATTR_MJPEG_S

[Description]

Defines the attribute of the MJPEG encoding.

[Syntax]

```
typedef struct hivENC_ATTR_MJPEG_S
{
    HI_U32    u32MaxPicWidth;
    HI_U32    u32MaxPicHeight;
    HI_U32    u32BufSize;
    HI_BOOL   bByFrame;
    HI_U32    u32PicWidth;
    HI_U32    u32PicHeight;
} VENC_ATTR_MJPEG_S;
```

[Member]

Member	Description
u32MaxPicWidth	Maximum width of a picture to be encoded, in pixel Value range: [MIN_WIDTH, MAX_WIDTH]. The value must be an integral multiple of MIN_ALIGN. Static attribute
u32PicWidth	Width of a picture to be encoded, in pixel Value range: [MIN_WIDTH, MAX_WIDTH] The value must be an integral multiple of MIN_ALIGN.
u32MaxPicHeight	Maximum height of a picture to be encoded, in pixel Value range: [MIN_HEIGHT, MAX_HEIGHT]. The value must be an integral multiple of MIN_ALIGN. Static attribute
u32PicHeight	Height of a picture to be encoded, in pixel Value range: [MIN_HEIGHT, MAX_HEIGHT] The value must be an integral multiple of MIN_ALIGN.
u32BufSize	Buffer size The value must be greater than or equal to the maximum picture width multiplied by the maximum picture height after 16-pixel alignment, and must be an integral multiple of 64. Static attribute The module parameter JpegMinBufMode can be configured when the .ko driver is loaded to select the mode for limiting the value range. <ul style="list-style-type: none">• JpegMinBufMode = 0 Value range: greater than or equal to the product of the 16-



Member	Description
	<p>pixel-aligned maximum picture width and maximum picture height</p> <ul style="list-style-type: none">• JpegeMiniBufMode = 1 (the user needs to ensure that the buffer size is appropriate) <p>Value range: greater than or equal to 32 x 1024 bytes</p> <p>Maximum value: There is no limitation but much memory is consumed if the buffer size is large.</p>
bByFrame	<p>Stream acquisition mode</p> <p>Value range: [HI_TRUE, HI_FALSE]</p> <ul style="list-style-type: none">• HI_TRUE: obtains streams by frame.• HI_FALSE: obtains streams by packet. <p>Static attribute</p> <p>Only the mode of obtaining streams by frame is supported.</p>

[Note]

None

[See Also]

None

VENC_ATTR_JPEG_S

[Description]

Defines the attribute of the JPEG snapshot.

[Syntax]

```
typedef struct hiVENC_ATTR_JPEG_S
{
    HI_U32    u32MaxPicWidth;
    HI_U32    u32MaxPicHeight;
    HI_U32    u32BufSize;
    HI_BOOL   bByFrame;
    HI_U32    u32PicWidth;
    HI_U32    u32PicHeight;
    HI_BOOL   bSupportDCF;
}VENC_ATTR_JPEG_S;
```

[Member]

Member	Description
u32MaxPicWidth	Maximum width of a picture to be encoded, in pixel Value range: [MIN_WIDTH, MAX_WIDTH]



Member	Description
	The value must be an integral multiple of MIN_ALIGN. Static attribute
u32PicWidth	Width of a picture to be encoded Value range: [MIN_WIDTH, MAX_WIDTH] The value must be an integral multiple of MIN_ALIGN.
u32MaxPicHeight	Maximum height of a picture to be encoded, in pixel Value range: [MIN_HEIGHT, MAX_HEIGHT] The value must be an integral multiple of MIN_ALIGN. Static attribute
u32PicHeight	Height of a picture to be encoded Value range: [MIN_HEIGHT, MAX_HEIGHT] The value must be an integral multiple of MIN_ALIGN.
u32BufSize	Buffer size. The value must be an integral multiple of 64. Static attribute The module parameter JpegeMiniBufMode can be configured when the .ko driver is loaded to select the mode for limiting the value range. <ul style="list-style-type: none">• JpegeMiniBufMode = 0 Value range: greater than or equal to the product of the 16-pixel-aligned maximum picture width and maximum picture height• JpegeMiniBufMode = 1 (the user needs to ensure that the buffer size is appropriate) Value range: greater than or equal to 32 x 1024 bytes Maximum value: There is no limitation but much memory is consumed if the buffer size is large.
bByFrame	Obtaining streams by frame/packet Value range: [HI_TRUE, HI_FALSE] <ul style="list-style-type: none">• HI_TRUE: obtains streams by frame.• HI_FALSE: obtains streams by packet. Static attribute Only the mode of obtaining streams by frame is supported
bSupportDCF	Whether the JPEG picture has a thumbnail Static attribute

[Note]

None

[See Also]



None

VENC_ATTR_H265_S

[Description]

Defines the attribute of the H.265 encoder.

[Syntax]

```
typedef struct hiVENC_ATTR_H265_S
{
    HI_U32    u32MaxPicWidth;
    HI_U32    u32MaxPicHeight;
    HI_U32    u32BufSize;
    HI_U32    u32Profile;
    HI_BOOL   bByFrame;
    HI_U32    u32PicWidth;
    HI_U32    u32PicHeight;
}VENC_ATTR_H265_S;
```

[Member]

Member	Description
u32MaxPicWidth	Maximum width of a picture to be encoded, in pixel Value range: [MIN_WIDTH, MAX_WIDTH] The value must be an integral multiple of MIN_ALIGN . Static attribute
u32PicWidth	Width of a picture to be encoded Value range: [MIN_WIDTH, MAX_WIDTH] The value must be an integral multiple of MIN_ALIGN .
u32MaxPicHeight	Maximum height of a picture to be encoded, in pixel Value range: [MIN_HEIGHT, MAX_HEIGHT] The value must be an integral multiple of MIN_ALIGN . Static attribute
u32PicHeight	Height of a picture to be encoded Value range: [MIN_HEIGHT, MAX_HEIGHT] The value must be an integral multiple of MIN_ALIGN .
u32BufSize	Stream buffer size, in byte Value range: [Min, Max] The value must be an integral multiple of 64. Static attribute You are advised to set the buffer size to the maximum size of a picture to be encoded. That is, the recommended value is as follows: u32MaxPicWidth x u32MaxPicHeight x1.5 bytes



Member	Description
	<p>The module parameter H265eMiniBufMode can be configured when the .ko driver is loaded to select the mode for limiting the value range.</p> <ul style="list-style-type: none">• H265eMiniBufMode = 0 Minimum value: 1/2 of the maximum size of a picture to be encoded Maximum value: There is no limitation but much memory is consumed if the buffer size is large.• H265eMiniBufMode = 1 (the user needs to ensure that the buffer size is appropriate) Minimum value: 32 x 1024 bytes Maximum value: There is no limitation but much memory is consumed if the buffer size is large.
bByFrame	<p>Mode for obtaining streams Value range: {HI_TRUE, HI_FALSE} HI_TRUE: obtain streams by frame. HI_FALSE: obtain streams by packet. Static attribute</p>
u32Profile	<p>Encoding level Value: 0 (main profile) Static attribute</p>

[Note]

None

[See Also]

None

VENC_ATTR_S

[Description]

Defines the encoder attributes.

[Syntax]

```
typedef struct hiVENC_ATTR_S
{
    PAYLOAD_TYPE_E enType;
    union
    {
        VENC_ATTR_H264_S stAttrH264e;
        VENC_ATTR_MJPEG_S stAttrMjpege;
        VENC_ATTR_JPEG_S stAttrJpege;
    }
}
```



```
VENC_ATTR_H265_S stAttrH265e;  
};  
}VENC_ATTR_S;
```

[Member]

Member	Description
enType	Protocol type
stAttrH264e/stAttrMjpeg/stAttrJpeg/ /stAttrH265e	Attribute of a protocol-compliant encoder

[Note]

None

[See Also]

[HI_MPI_VENC_CreateChn](#)

VENC_GOP_MODE_E

[Description]

Defines the GOP type for H.264/H.265 encoding.

[Syntax]

```
typedef enum hivENC_GOP_MODE_E  
{  
    VENC_GOPMODE_NORMALP    = 0,  
    VENC_GOPMODE_DUALP      = 1,  
    VENC_GOPMODE_SMARTP     = 2,  
    VENC_GOPMODE_BIPREDB    = 3,  
    VENC_GOPMODE_LOWDELAYB  = 4,  
    VENC_GOPMODE_BUTT,  
}VENC_GOP_MODE_E;
```

[Member]

Member	Description
VENC_GOPMODE_NORMALP	Type of the GOP for encoding the P frame that requires only one reference frame
VENC_GOPMODE_DUALP	Type of the GOP for encoding the P frame that requires two reference frames
VENC_GOPMODE_SMARTP	Type of the GOP for encoding the smart P frame
VENC_GOPMODE_BIPREDB	Type of the GOP for encoding the B frame



Member	Description
VENC_GOPMODE_LOWDELAYB	Type of the GOP for encoding the B frame that requires only the forward reference frame.

[Note]

None

[See Also]

None

VENC_GOP_NORMALP_S

[Description]

Defines the structure of the GOP attribute for encoding the P frame that requires only one reference frame.

[Syntax]

```
typedef struct hivENC_GOP_NORMALP_S
{
    HI_S32 s32IPQpDelta;
}VENC_GOP_NORMALP_S;
```

[Member]

Member	Description
s32IPQpDelta	I/P QP variance Value range: [-10, +30]

[Note]

None

[See Also]

None

VENC_GOP_DUALP_S

[Description]

Defines the structure of the GOP attribute for encoding the P frame that requires two reference frames.

[Syntax]

```
typedef struct hivENC_GOP_DUALP_S
{
    HI_U32 u32SPInterval;
```



```
    HI_S32 s32SPQpDelta;  
    HI_S32 s32IPQpDelta;  
}VENC_GOP_DUALP_S;
```

[Member]

Member	Description
u32SPInterval	Interval of the special P frames (the special P frame indicates the P frame that requires only one reference frame.) Value range: $[0, 1) \cup (1, u32Gop - 1]$ (u32Gop indicates the interval of the I frames)
s32SPQpDelta	QP delta for the special P frame and the P frame that requires two reference frames Value range: $[-10, +30]$
s32IPQpDelta	I/P QP variance Value range: $[-10, +30]$

[Note]

None

[See Also]

None

VENC_GOP_SMARTP_S

[Description]

Defines the structure of the GOP attribute for encoding the smart P frame.

[Syntax]

```
typedef struct hivENC_GOP_SMARTP_S  
{  
    HI_U32 u32BgInterval;  
    HI_S32 s32BgQpDelta;  
    HI_S32 s32ViQpDelta;  
}VENC_GOP_SMARTP_S;
```

[Member]

Member	Description
u32BgInterval	Interval of the long-term reference frame Value range: u32BgInterval must be greater than or equal to u32Gop and must be an integral multiple of u32Gop .
s32BgQpDelta	QP delta for the long-term reference frame and P frame Value range: $[-10, +30]$



Member	Description
s32ViQpDelta	QP delta for the virtual I frame and P frame Value range: [-10, +30]

[Note]

None

[See Also]

None

VENC_GOP_BIPREDB_S

[Description]

Defines the structure of the GOP attribute for encoding the B frame.

[Syntax]

```
typedef struct hivENC_GOP_BIPREDB_S
{
    HI_U32 u32BFrmNum;
    HI_S32 s32BQpDelta;
    HI_S32 s32IPQpDelta;
}VENC_GOP_BIPREDB_S;
```

[Member]

Member	Description
u32BFrmNum	Number of B frames to be encoded Value range: [1, 3]
s32BQpDelta	QP delta for the B frame and P frame Value range: [-10, +30]
s32IPQpDelta	I/P QP variance Value range: [-10, +30]

[Note]

None

[See Also]

None

VENC_GOP_ATTR_S

[Description]



Defines the structure of the GOP attribute for the encoder.

[Syntax]

```
typedef struct hiVENC_GOP_ATTR_S
{
    VENC_GOP_MODE_E enGopMode;
    union
    {
        VENC_GOP_NORMALP_S stNormalP;
        VENC_GOP_DUALP_S   stDualP;
        VENC_GOP_SMARTP_S  stSmartP;
        VENC_GOP_BIPREDB_S stBipredB;
    };
} VENC_GOP_ATTR_S;
```

[Member]

Member	Description
enGopMode	Encoding GOP type
stNormalP	Structure of the GOP attribute for encoding the P frame that requires only one reference frame
stDualP	Structure of the GOP attribute for encoding the P frame that requires two reference frames
stSmartP	Structure of the GOP attribute for encoding the smart P frame
stBipredB	Structure of the GOP attribute for encoding the B frame

[Note]

None

[See Also]

[HI_MPI_VENC_CreateChn](#)

VENC_CHN_ATTR_S

[Description]

Defines the attribute of a VENC channel.

[Syntax]

```
typedef struct hiVENC_CHN_ATTR_S
{
    VENC_ATTR_S      stVeAttr;
    VENC_RC_ATTR_S  stRcAttr;
    VENC_GOP_ATTR_S stGopAttr;
}
```



```
}VENC_CHN_ATTR_S;
```

[Member]

Member	Description
stVeAttr	Encoder attributes
stRcAttr	Bit rate controller attributes
stGopAttr	Structure of the GOP mode

[Note]

None

[See Also]

[HI_MPI_VENC_CreateChn](#)

VENC_CHN_STAT_S

[Description]

Defines the status of a VENC channel.

[Syntax]

```
typedef struct hiVENC_CHN_STAT_S
{
    HI_U32    u32LeftPics;
    HI_U32    u32LeftStreamBytes;
    HI_U32    u32LeftStreamFrames;
    HI_U32    u32CurPacks;
    HI_U32    u32LeftRecvPics;
    HI_U32    u32LeftEncPics;
}VENC_CHN_STAT_S;
```

[Member]

Member	Description
u32LeftPics	Number of pictures to be encoded
u32LeftStreamBytes	Number of remaining bytes in a stream buffer
u32LeftStreamFrames	Number of remaining frames in a stream buffer
u32CurPacks	Number of stream packets in the current frame
u32LeftRecvPics	Number of frames to be received. This member is valid after HI_MPI_VENC_StartRecvPicEx is called.
u32LeftEncPics	Number of frames to be encoded. This member is valid after HI_MPI_VENC_StartRecvPicEx is called.



[Note]

None

[See Also]

[HI_MPI_VENC_Query](#)

VENC_PARAM_H264_SLICE_SPLIT_S

[Description]

Defines the slice structure of an H.264 channel.

[Syntax]

```
typedef struct hivenc_param_h264_slice_split_s
{
    HI_BOOL bSplitEnable;
    HI_U32 u32SplitMode;
    HI_U32 u32SliceSize;
} VENC_PARAM_H264_SLICE_SPLIT_S;
```

[Member]

Member	Description
bSplitEnable	Whether the slice function is enabled
u32SplitMode	Slice mode <ul style="list-style-type: none">• 0: slice a frame by byte.• 1: slice a frame by rows of macroblocks. This member must be set to be less than 2 .
u32SliceSize	When u32SplitMode is set to 0 , it indicates the number of byte per slice Minimum value: 128 Maximum value: min((0xFFFF), u32PicSize/2) where, u32PicSize = picwidth x picheight x 3/2 When u32SplitMode is set to 1 , it indicates the number of rows of macroblocks occupied by each slice. Minimum value: 1 Maximum value: (Picture height + 15)/16

[Note]

None

[See Also]

- [HI_MPI_VENC_SetH264SliceSplit](#)



- [HI_MPI_VENC_GetH264SliceSplit](#)

VENC_PARAM_H264_INTER_PRED_S

[Description]

Defines the inter-prediction structure of an H.264 channel.

[Syntax]

```
typedef struct hiVENC_PARAM_H264_INTER_PRED_S
{
    /* search window */
    HI_U32 u32HWSize;
    HI_U32 u32VWSize;
    HI_BOOL bInter16x16PredEn;
    HI_BOOL bInter16x8PredEn;
    HI_BOOL bInter8x16PredEn;
    HI_BOOL bInter8x8PredEn;
    HI_BOOL bExtedgeEn;
} VENC_PARAM_H264_INTER_PRED_S;
```

[Member]

Member	Description
u32HWSize	Size of a horizontal search window. This member is reserved and not used currently.
u32VWSize	Size of a vertical search window. This member is reserved and not used currently.
bInter16x16PredEn	16x16 inter-prediction. By default, this function is enabled.
bInter16x8PredEn	16x8 inter-prediction enable. By default, this function is enabled. This member is reserved and not used currently.
bInter8x16PredEn	8x16 inter-prediction enable. By default, this function is enabled. This member is reserved and not used currently.
bInter8x8PredEn	8x8 inter-prediction. By default, this function is enabled.
bExtedgeEn	When a search window is larger than a picture, this member indicates whether to enable edge extension of the picture. By default, edge extension is enabled.

[Note]

None

[See Also]

- [HI_MPI_VENC_SetH264InterPred](#)
- [HI_MPI_VENC_GetH264InterPred](#)



VENC_PARAM_H264_INTRA_PRED_S

[Description]

Defines the intra-prediction structure of an H.264 channel.

[Syntax]

```
typedef struct hiVENC_PARAM_H264_INTRA_PRED_S
{
    HI_BOOL bIntra16x16PredEn;
    HI_BOOL bIntraNxNPredEn;
    HI_U32 constrained_intra_pred_flag;
    HI_BOOL bIpcmEn;
} VENC_PARAM_H264_INTRA_PRED_S;
```

[Member]

Member	Description
bIntra16x16PredEn	16x16 intra-prediction enable. By default, this function is enabled. Value: <ul style="list-style-type: none">• 0: disabled• 1: enabled
bIntraNxNPredEn	NxN intra-prediction enable. By default, this function is enabled. Value: <ul style="list-style-type: none">• 0: disabled• 1: enabled
constrained_intra_pred_flag;	Default value: 0 Value: 0 or 1
bIpcmEn	IP camera prediction enable. The default value varies depending on chip type. Value: 0 or 1

[Note]

For the specific meanings of the members, see the H.264 protocol.

[See Also]

- [HI_MPI_VENC_SetH264IntraPred](#)
- [HI_MPI_VENC_GetH264IntraPred](#)

VENC_PARAM_H264_TRANS_S

[Description]



Defines the transformation and quantization structure of an H.264 channel.

[Syntax]

```
typedef struct hiVENC_PARAM_H264_TRANS_S
{
    HI_U32 u32IntraTransMode;
    HI_U32 u32InterTransMode;
    HI_BOOL bScalingListValid;
    HI_U8 InterScalingList8X8[64];
    HI_U8 IntraScalingList8X8[64];
    HI_S32 chroma_qp_index_offset;
} VENC_PARAM_H264_TRANS_S;
```

[Member]

Member	Description
u32IntraTransMode	Conversion mode for intra-prediction <ul style="list-style-type: none">• 0: supports 4x4 and 8x8 transformation and high profile.• 1: supports 4x4 transformation and baseline, main, and high profiles.• 2: supports 8x8 transformation and high profile. The system sets this member based on the channel protocol type.
u32InterTransMode	Transformation mode for inter-prediction <ul style="list-style-type: none">• 0: supports 4x4 and 8x8 transformation and high profile.• 1: supports 4x4 transformation and baseline, main, and high profiles.• 2: supports 8x8 transformation and high profile. The system sets this member based on the channel protocol type.
bScalingListValid	InterScalingList8x8 and IntraScalingList8x8 are valid only in the H.264 high profile. Value: <ul style="list-style-type: none">• 0: invalid• 1: valid This member can only be set to 0 .
InterScalingList8x8	A quantization table for 8x8 inter-prediction. You can use your own quantization table in the high profile. Value range: [1, 255] This member is reserved and not used currently.
IntraScalingList8x8	A quantization table for 8x8 intra-prediction. You can use your own quantization table in the high profile. This member is reserved and not used currently. Value range: [1, 255]



Member	Description
chroma_qp_index_offset	For details, see the H.264 protocol. Default value: 0 Value range: [-12, +12]

[Note]

- For the specific meanings of the members, see the H.264 protocol.
- Hi3516C V300 does not support quantization tables, and therefore does not support the configurations of **bScalingListValid**, **InterScalingList8X8[64]**, and **IntraScalingList8X8[64]**.

[See Also]

- [HI_MPI_VENC_SetH264Trans](#)
- [HI_MPI_VENC_GetH264Trans](#)

VENC_PARAM_H264_ENTROPY_S

[Description]

Defines the entropy structure of an H.264 channel.

[Syntax]

```
typedef struct hivENC_PARAM_H264_ENTROPY_S
{
    HI_U32 u32EntropyEncModeI;
    HI_U32 u32EntropyEncModeP;
    HI_U32 u32EntropyEncModeB;
    HI_U32 cabac_stuff_en;
    HI_U32 cabac_init_idc;
}VENC_PARAM_H264_ENTROPY_S;
```

[Member]

Member	Description
u32EntropyEncModeI	Entropy encoding mode for the I frame <ul style="list-style-type: none">• 0: CAVLC• 1: CABAC• Other values: undefined The baseline profile does not support CABAC.
u32EntropyEncModeP	Entropy encoding mode for the P frame <ul style="list-style-type: none">• 0: CAVLC• 1: CABAC• Other values: null



Member	Description
	The baseline profile does not support CABAC.
u32EntropyEncModeB	Entropy encoding mode for the B frame <ul style="list-style-type: none">• 0: CAVLC• 1: CABAC• Other values: undefined The baseline profile does not support CABAC and the B frame.
cabac_stuff_en	For details, see the H.264 protocol. By default, this member is set to 0. Value: 0 or 1
cabac_init_idc	For details, see the H.264 protocol. The value range is 0–2. The default value is 0.

[Note]

None

[See Also]

- [HI_MPI_VENC_SetH264Entropy](#)
- [HI_MPI_VENC_GetH264Entropy](#)

VENC_PARAM_H264_POC_S

[Description]

Defines the POC structure of an H.264 channel.

[Syntax]

```
typedef struct hivenc_param_h264_poc_s
{
    HI_U32 pic_order_cnt_type;
}VENC_PARAM_H264_POC_S;
```

[Member]

Member	Description
pic_order_cnt_type	The value range is 0–2, and the default value is 2. For details, see the H.264 protocol.

[Note]

None

[See Also]



- [HI_MPI_VENC_SetH264Poc](#)
- [HI_MPI_VENC_GetH264Poc](#)

VENC_PARAM_H264_DBLK_S

[Description]

Defines the de-blocking structure of an H.264 channel.

[Syntax]

```
typedef struct hiVENC_PARAM_H264_DBLK_S
{
    HI_U32 disable_deblocking_filter_idc;
    HI_S32 slice_alpha_c0_offset_div2;
    HI_S32 slice_beta_offset_div2;
}VENC_PARAM_H264_DBLK_S;
```

[Member]

Member	Description
disable_deblocking_filter_idc	The value range is [0, 2], and the default value is 0. For details, see the H.264 protocol.
slice_alpha_c0_offset_div2	The value range is [-6, +6], and the default value is 5. For details, see the H.264 protocol.
slice_beta_offset_div2	The value range is [-6, +6], and the default value is 5. For details, see the H.264 protocol.

[Note]

None

[See Also]

- [HI_MPI_VENC_SetH264Dbblk](#)
- [HI_MPI_VENC_GetH264Dbblk](#)

VENC_PARAM_H264_VUI_S

[Description]

Defines the VUI structure of an H.264 channel.

[Syntax]

```
typedef struct hiVENC_PARAM_H264_VUI_S
{
    VENC_PARAM_VUI_ASPECT_RATIO_S      stVuiAspectRatio;
    VENC_PARAM_VUI_H264_TIME_INFO_S    stVuiTimeInfo;
    VENC_PARAM_VUI_VIDEO_SIGNAL_S     stVuiVideoSignal;
    VENC_PARAM_VUI_BITSTREAM_RESTRIC_S stVuiBitstreamRestric;
```



```
}VENC_PARAM_H264_VUI_S;
```

[Member]

Member	Description
stVuiAspectRatio	For details, see the H.264 protocol.
stVuiTimeInfo	For details, see the H.264 protocol.
stVuiVideoSignal	For details, see the H.264 protocol.
stVuiBitstreamRestric	For details, see the H.264 protocol.

[Note]

None

[See Also]

- [HI_MPI_VENC_SetH264Vui](#)
- [HI_MPI_VENC_GetH264Vui](#)

VENC_PARAM_H265_VUI_S

[Description]

Defines the VUI structure of an H.265 VENC channel.

[Syntax]

```
typedef struct hivENC_PARAM_H265_VUI_S
{
    VENC_PARAM_VUI_ASPECT_RATIO_S      stVuiAspectRatio;
    VENC_PARAM_VUI_H265_TIME_INFO_S    stVuiTimeInfo;
    VENC_PARAM_VUI_VIDEO_SIGNAL_S     stVuiVideoSignal;
    VENC_PARAM_VUI_BITSTREAM_RESTRIC_S stVuiBitstreamRestric;
}VENC_PARAM_H265_VUI_S;
```

[Member]

Member	Description
stVuiAspectRatio	For details, see the H.265 protocol.
stVuiTimeInfo	For details, see the H.265 protocol.
stVuiVideoSignal	For details, see the H.265 protocol.
stVuiBitstreamRestric	For details, see the H.265 protocol.

[Note]

None



[See Also]

- [HI_MPI_VENC_SetH265Vui](#)
- [HI_MPI_VENC_GetH265Vui](#)

VENC_PARAM_VUI_ASPECT_RATIO_S

[Description]

Defines the AspectRatio information in the VUI of an H.264/H.265 VENC channel.

[Syntax]

```
typedef struct hiVENC_PARAM_VUI_ASPECT_RATIO_S
{
    HI_U8          aspect_ratio_info_present_flag;
    HI_U8          aspect_ratio_idc;
    HI_U8          overscan_info_present_flag;
    HI_U8          overscan_appropriate_flag;
    HI_U16         sar_width;
    HI_U16         sar_height ;
} VENC_PARAM_VUI_ASPECT_RATIO_S;
```

[Member]

Member	Description
aspect_ratio_info_present_flag	For details, see the H.264/H.265 protocol. The default value is 0. Value: 0 or 1
aspect_ratio_idc	For details, see the H.264/H.265 protocol. The default value is 1. Value range: 0–255 excluding 17–254
overscan_info_present_flag	For details, see the H.264/H.265 protocol. The default value is 0. Value: 0 or 1
overscan_appropriate_flag	For details, see the H.264/H.265 protocol. The default value is 0. Value: 0 or 1
sar_width	For details, see the H.264/H.265 protocol. The default value is 1. Value range: (0, 65535]. sar_width and sar_height are relatively prime.
sar_height	For details, see the H.264/H.265 protocol. The default value is 1. Value range: (0, 65535]. sar_height and sar_width are relatively prime.



[Note]

Note

[See Also]

- [VENC_PARAM_H264_VUI_S](#)
- [VENC_PARAM_H265_VUI_S](#)

VENC_PARAM_VUI_H264_TIME_INFO_S

[Description]

Defines the TIME_INFO information in the VUI of an H.264 encoding channel.

[Syntax]

```
typedef struct hivENC_PARAM_H264_VUI_TIME_INFO_S
{
    HI_U8          timing_info_present_flag;
    HI_U8          fixed_frame_rate_flag;
    HI_U32         num_units_in_tick;
    HI_U32         time_scale;
}VENC_PARAM_VUI_H264_TIME_INFO_S;
```

[Member]

Member	Description
timing_info_present_flag	For details, see the H.264 protocol. The default value is 0. Value: 0 or 1
num_units_in_tick	For details, see the H.264 protocol. The default value is 1. Value range: > 0
time_scale	For details, see the H.264 protocol. The default value is 60. Value range: > 0
fixed_frame_rate_flag;	For details, see the H.264 protocol. The default value is 1. Value: 0 or 1

[Note]

Note

[See Also]

[VENC_PARAM_H264_VUI_S](#)

VENC_PARAM_VUI_H265_TIME_INFO_S

[Description]



Defines the Time_Info information in the VUI of an H.265 VENC channel.

[Syntax]

```
typedef struct hiVENC_PARAM_H265_TIME_INFO_S
{
    HI_U32 timing_info_present_flag;
    HI_U32 num_units_in_tick;
    HI_U32 time_scale;
    HI_U32 num_ticks_poc_diff_one_minus1;
} VENC_PARAM_VUI_H265_TIME_INFO_S;
```

[Member]

Member	Description
timing_info_present_flag	For details, see the H.265 protocol. The default value is 0. Value: 0 or 1
num_units_in_tick	For details, see the H.265 protocol. The default value is 1. Value range: > 0
time_scale	For details, see the H.265 protocol. The default value is 60. Value range: > 0
num_ticks_poc_diff_one_minus1	For details, see the H.265 protocol. The default value is 1. Value range: [0, $2^{32} - 2$]

[Note]

None

[See Also]

[VENC_PARAM_H265_VUI_S](#)

VENC_PARAM_VUI_VIDEO_SIGNAL_S

[Description]

Defines the VIDEO_SIGNAL information in the VUI of an H.264/H.265 VENC channel.

[Syntax]

```
typedef struct hiVENC_PARAM_VUI_VIDEO_SIGNAL_S
{
    HI_U8     video_signal_type_present_flag;
    HI_U8     video_format;
    HI_U8     video_full_range_flag;
```



```
    HI_U8      colour_description_present_flag;
    HI_U8      colour_primaries;
    HI_U8      transfer_characteristics;
    HI_U8      matrix_coefficients;
}VENC_PARAM_VUI_VIDEO_SIGNAL_S;
```

[Member]

Member	Description
video_signal_type_present_flag	For details, see the H.264/H.265 protocol. The default value is 1. Value: 0 or 1
video_format	For details, see the H.264/H.265 protocol. The default value is 5. Value range: H.264:[0, 7], H.265: [0, 5]
video_full_range_flag	For details, see the H.264/H.265 protocol. The default value is 1. Value: 0 or 1
colour_description_present_flag	For details, see the H.264/H.265 protocol. The default value is 1. Value: 0 or 1
colour_primaries	For details, see the H.264/H.265 protocol. The default value is 1. Value range: [0, 255]
transfer_characteristics	For details, see the H.264/H.265 protocol. The default value is 1. Value range: [0, 255]
matrix_coefficients	For details, see the H.264/H.265 protocol. The default value is 1. Value range: [0, 255]

[Note]

Note

[See Also]

- [VENC_PARAM_H264_VUI_S](#)
- [VENC_PARAM_H265_VUI_S](#)

VENC_PARAM_VUI_BITSTREAM_RESTRIC_S

[Description]

Defines the Bitstream_Restriction structure in the VUI of an H.264/H.265 VENC channel.



[Syntax]

```
typedef struct hiVENC_PARAM_VUI_BITSTREAM_RESTRICT_S
{
    HI_U8 bitstream_restriction_flag ;
} VENC_PARAM_VUI_BITSTREAM_RESTRICT_S;
```

[Member]

Member	Description
bitstream_restriction_flag	For details, see the H.264/H.265 protocol. The default value is 0. Value: 0 or 1

[Note]

Note

[See Also]

- [VENC_PARAM_H264_VUI_S](#)
- [VENC_PARAM_H265_VUI_S](#)

VENC_PARAM_JPEG_S

[Description]

Defines the parameters of the JPEG encoder.

[Syntax]

```
typedef struct hiVENC_PARAM_JPEG_S
{
    HI_U32 u32Qfactor;
    HI_U8 u8YQt[64];
    HI_U8 u8CbQt[64];
    HI_U8 u8CrQt[64];
    HI_U32 u32MCUPerECS;
} VENC_PARAM_JPEG_S;
```

[Member]

Member	Description
u32Qfactor	The value range is [1, 99], and the default value is 90. For details, see the RFC2435 protocol.
u8YQt	Y quantization table Value range: [1, 255]
u8CbQt	Cb quantization table



Member	Description
	Value range: [1, 255]
u8CrQt	Cr quantization table Value range: [1, 255]
u32MCUPerECS	This member indicates the number of MCUs per ECS. By default, the value is set to 0 , indicating no ECS division. u32MCUPerECS: [0, (picwidth + 15) >> 4 x (picheight + 15) >> 4 x 2]

[Note]

None

[See Also]

- [HI_MPI_VENC_SetJpegParam](#)
- [HI_MPI_VENC_GetJpegParam](#)

VENC_PARAM_MJPEG_S

[Description]

Defines the advanced parameters of the MJPEG encoder.

[Syntax]

```
typedef struct hivenc_param_mjpeg_s
{
    HI_U8    u8YQt[64];
    HI_U8    u8CbQt[64];
    HI_U8    u8CrQt[64];
    HI_U32   u32MCUPerECS;
} VENC_PARAM_MJPEG_S;
```

[Member]

Member	Description
u8YQt	Y quantization table Value range: [1, 255]
u8CbQt	Cb quantization table Value range: [1, 255]
u8CrQt	Cr quantization table Value range: [1, 255]
u32MCUPerECS	Number of MCUs in each ECS. The default value 0 indicates that there are no ECSs. u32MCUPerECS: [0, (picwidth + 15) >> 4 x (picheight + 15) >> 4 x 2]



[Note]

None

[See Also]

- [HI_MPI_VENC_SetMjpegParam](#)
- [HI_MPI_VENC_GetMjpegParam](#)

VENC_ROI_CFG_S

[Description]

Defines an ROI.

[Syntax]

```
typedef struct hivENC_ROI_CFG_S
{
    HI_U32 u32Index;
    HI_BOOL bEnable;
    HI_BOOL bAbsQp;
    HI_S32 s32Qp;
    RECT_S stRect;
}VENC_ROI_CFG_S;
```

[Member]

Member	Description
u32Index	Index of an ROI. The system supports indexes ranging from 0 to 7 .
bEnable	Whether to enable this ROI
bAbsQp	QP mode of an ROI <ul style="list-style-type: none">• HI_FALSE: relative QP• HI_TRUE: absolute QP
s32Qp	QP value. When the QP mode is set to HI_FALSE , this member indicates the QP offset, and the value range is $[-51, +51]$. When the QP mode is set to HI_TRUE , this member indicates the macroblock QP value, and the value range is $[0, 51]$.
stRect	Region of an ROI The value of s32X , s32Y , u32Width , or u32Height must be a multiple of 16.

[Note]

None



[See Also]

- [HI_MPI_VENC_SetRoiCfg](#)
- [HI_MPI_VENC_GetRoiCfg](#)

VENC_ROIBG_FRAME_RATE_S

[Description]

Defines the frame rate of a non-ROI.

[Syntax]

```
typedef struct hivENC_ROIBG_FRAME_RATE_S
{
    HI_S32 s32SrcFrmRate;
    HI_S32 s32DstFrmRate;
}VENC_ROIBG_FRAME_RATE_S;
```

[Member]

Member	Description
s32SrcFrmRate	Source frame rate of a non-ROI
s32DstFrmRate	Target frame rate of a non-ROI

[Note]

None

[See Also]

- [HI_MPI_VENC_SetRoiBgFrameRate](#)
- [HI_MPI_VENC_GetRoiBgFrameRate](#)

VENC_PARAM_REF_S

[Description]

Defines the advanced frame skipping reference parameters for H.264/H.265 encoding.

[Syntax]

```
typedef struct hivENC_PARAM_REF_S
{
    HI_U32    u32Base;
    HI_U32    u32Enhance;
    HI_BOOL   bEnablePred;
}VENC_PARAM_REF_S;
```

[Member]



Member	Description
u32Base	Base layer period Value range: (0, +∞)
u32Enhance	Enhance layer period Value range: [0, 255]
bEnablePred	Whether some frames at the base layer are referenced by other frames at the base layer. When bEnablePred is HI_FALSE , all frames at the base layer reference IDR frames.

[Note]

None

[See Also]

None

VENC_RC_ATTR_S

[Description]

Defines RC attributes for controlling the bit rate of a VENC channel.

[Syntax]

```
typedef struct hiVENC_RC_ATTR_S
{
    VENC_RC_MODE_E          enRcMode;
    union
    {
        VENC_ATTR_H264_CBR_S    stAttrH264Cbr;
        VENC_ATTR_H264_VBR_S    stAttrH264Vbr;
        VENC_ATTR_H264_AVBR_S   stAttrH264AVbr;
        VENC_ATTR_H264_FIXQP_S  stAttrH264FixQp;
        VENC_ATTR_H264_QPMAP_S  stAttrH264QpMap;
        VENC_ATTR_MJPEG_CBR_S   stAttrMjpegeCbr;
        VENC_ATTR_MJPEG_VBR_S   stAttrMjpegeVbr;
        VENC_ATTR_MJPEG_FIXQP_S stAttrMjpegeFixQp;
        VENC_ATTR_H265_CBR_S    stAttrH265Cbr;
        VENC_ATTR_H265_VBR_S    stAttrH265Vbr;
        VENC_ATTR_H265_AVBR_S   stAttrH265AVbr;
        VENC_ATTR_H265_FIXQP_S  stAttrH265FixQp;
        VENC_ATTR_H265_QPMAP_S  stAttrH265QpMap;
    };
    HI_VOID*      pRcAttr ;
}VENC_RC_ATTR_S;
```



[Member]

Member	Description
enRcMode	RC mode
stAttrH264Cbr	CBR mode attribute of an H.264 channel
stAttrH264Vbr	VBR mode attribute of an H.264 channel
stAttrH264AVbr	AVBR mode attribute of an H.264 channel
stAttrH264FixQp	FixQp mode attribute of an H.264 channel
stAttrH264QpMap	QpMap mode attribute of an H.264 channel Hi3519 V100 does not support this mode.
stAttrMjpegeFixQp	FixQp mode attribute of an MJPEG channel
stAttrMjpegeCbr	CBR mode attribute of an MJPEG channel
stAttrMjpegeVbr	VBR mode attribute of an MJPEG channel
stAttrH265Cbr	CBR mode attribute of an H.265 encoding channel
stAttrH265Vbr	VBR mode attribute of an H.265 encoding channel
stAttrH265AVbr	AVBR mode attribute of an H.265 encoding channel
stAttrH265FixQp	FixQp mode attribute of an H.265 encoding channel
stAttrH265QpMap	QpMap mode attribute of an H.265 encoding channel Hi3519 V100 does not support this member.

[Note]

None

[See Also]

[HI_MPI_VENC_CreateChn](#)

VENC_RC_MODE_E

[Description]

Defines the mode of the RC for controlling the bit rate of a VENC channel.

[Syntax]

```
typedef enum hivenc_rc_mode_e
{
    VENC_RC_MODE_H264CBR = 1,
    VENC_RC_MODE_H264VBR,
    VENC_RC_MODE_H264AVBR,
    VENC_RC_MODE_H264FIXQP,
    VENC_RC_MODE_H264QPMAP,
```



```
VENC_RC_MODE_MJPEGCBR,  
VENC_RC_MODE_MJPEGVBR,  
VENC_RC_MODE_MJPEGABR,  
VENC_RC_MODE_MJPEGFIXQP,  
VENC_RC_MODE_H265CBR,  
VENC_RC_MODE_H265VBR,  
VENC_RC_MODE_H265AVBR,  
VENC_RC_MODE_H265FIXQP,  
VENC_RC_MODE_H265QPMAP,  
VENC_RC_MODE_BUTT,  
}VENC_RC_MODE_E;
```

[Member]

Member	Description
VENC_RC_MODE_H264CBR	H.264 CBR mode
VENC_RC_MODE_H264VBR	H.264 VBR mode
VENC_RC_MODE_H264AVBR	H.264 AVBR mode
VENC_RC_MODE_H264FIXQP	H264 FixQp mode
VENC_RC_MODE_H264QPMAP	H.264 QpMap mode Hi3519 V100 does not support this member.
VENC_RC_MODE_MJPEGCBR	MJPEG CBR mode
VENC_RC_MODE_MJPEGVBR	MJPEG VBR mode
VENC_RC_MODE_MJPEGFIXQP	MJPEG FixQp mode
VENC_RC_MODE_H265CBR	H.265 CBR mode
VENC_RC_MODE_H265VBR	H.265 VBR mode
VENC_RC_MODE_H265AVBR	H.265 AVBR mode
VENC_RC_MODE_H265FIXQP	H.265 FixQp mode
VENC_RC_MODE_H265QPMAP	H.265 QpMap mode Hi3519 V100 does not support this member.

[Note]

None

[See Also]

[HI_MPI_VENC_CreateChn](#)



VENC_RC_QPMAP_MODE_E

[Description]

Defines the method of assigning the QP values for CU32 and CU64 when the VENC channel uses the QpMap mode.

[Syntax]

```
typedef enum hiVENC_RC_QPMAP_MODE_E
{
    VENC_RC_QPMAP_MODE_MEANQP= 0,
    VENC_RC_QPMAP_MODE_MINQP ,
    VENC_RC_QPMAP_MODE_MAXQP,
    VENC_RC_QPMAP_MODE_BUTT,
}VENC_RC_QPMAP_MODE_E;
```

[Member]

Member	Description
VENC_RC_QPMAP_MODE_MEANQP	The QP value of CU32 is the average QP value of four 16 x 16 blocks. The QP value of CU64 is the average QP value of sixteen 16 x 16 blocks.
VENC_RC_QPMAP_MODE_MINQP	The QP value of CU32 is the minimum QP value of four 16 x 16 blocks. The QP value of CU64 is the minimum QP value of sixteen 16 x 16 blocks.
VENC_RC_QPMAP_MODE_MAXQP	The QP value of CU32 is the maximum QP value of four 16 x 16 blocks. The QP value of CU64 is the maximum QP value of sixteen 16 x 16 blocks.

[Note]

None

[See Also]

[HI_MPI_VENC_CreateChn](#)

VENC_ATTR_H264_CBR_S

[Description]

Defines the CBR attribute structure of an H.264 encoding channel.

[Syntax]

```
typedef struct hiVENC_ATTR_H264_CBR_S
{
```



```
    HI_U32      u32Gop;
    HI_U32      u32StatTime;
    HI_U32      u32SrcFrmRate;
    HI_FR32     fr32DstFrmRate;
    HI_U32      u32BitRate;
    HI_U32      u32FluctuateLevel;
} VENC_ATTR_H264_CBR_S;
```

[Member]

Member	Description
u32Gop	H264 GOP value Value range: [1, 65536]
u32StatTime	CBR statistics time, in second Value range: [1, 60]
u32SrcFrmRate	VI frame rate, in frame/s Value range: [1, 240]
fr32DstFrmRate	Output frame rate of the encoder, in frame/s Value range: [1/16, u32SrcFrmRate]
u32BitRate	Average bit rate, in kbit/s Value range for Hi3519 V100/Hi3519 V101: [2, 102400] Value range for Hi3516C V300: [2, 30720]
u32FluctuateLevel	Fluctuation level of the maximum bit rate relative to the average bit rate. u32FluctuateLevel is reserved and not used currently. Value range: [1, 5] The fluctuation level 1 is recommended

[Note]

- When the source frame rate **SrcFrmRate** is set:
 - If the VENC frame rate control uses the frame adding mode, the actual encoding frame rate is the output frame rate of the VENC. In this mode, **SrcFrmRate** should be set to the output frame rate of the VENC.
 - If the VENC frame rate control uses the frame reduction mode or frame rate control is not implemented, **SrcFrmRate** should be set to the actual frame rate at which TimeRef is generated. The RC needs to calculate the actual frame rate and control the bit rate based on the value of **SrcFrmRate**.
 - If the source of the picture to be encoded is VI, **SrcFrmRate** is set to the actual output frame rate of the VI because TimeRef is generated when the VI outputs frames.
 - If the VI and VPSS are in online mode, **SrcFrmRate** is set to the actual output frame rate of the VPSS because TimeRef is generated when the VPSS outputs frames.



- If the VI and VPSS are in offline mode, the frame rate of the VPSS is controlled by group, and the source frame rate of the VPSS group must be set to the same value as **SrcFrmRate**. In this case, **SrcFrmRate** is set to the actual output frame rate of the VI because TimeRef is generated when the VI outputs frames.

Assume that the output frame rate of the VI is 30:

- If the VENC does not implement frame rate control, **SrcFrmRate** should be set to **30**.
- If the VENC implements frame rate control, **SrcFrmRate** should be set to the output frame rate in frame adding mode and be set to **30** in frame reduction mode.
- If the VENC uses the frame reduction mode and the RC controls the frame rate, the final frame rate is the smaller value between the two frame rates controlled by the VENC and RC. For example, if the input frame rate and output frame rate of the VENC are **30** and **6**, respectively and those of the RC are **30** and **10**, respectively, the final output frame rate is **6**.
- When **DstFrmRate** is set, its data structure is defined as the fractional type HI_FR32 that is the same as HI_U32. That is, the upper 16 bits indicate the denominator and the lower 16 bits indicate the numerator.
 - To set **DstFrmRate** to an integer, set the upper 16 bits to zeros. For example, if you set **SrcFrmRate** to **25** and **DstFrmRate** to **12**, 12 frames will be selected from 25 frames for encoding, and the other 13 frames are discarded.
 - If you set **SrcFrmRate** to **25** and **DstFrmRate** to **15/2**, the encoder will encode 15 frames in two seconds.
- **DstFrmRate** can be set as follows:
 - If you want the integral frame rate 25, set **DstFrmRate** to **25**.
 - If you want the fractional frame rate 15/2, set **DstFrmRate** to **[15 + (2 << 16)]**.
- When both the VENC and RC are used to control the frame rate, the target frame rate of the VENC is less than that of the RC. For example, if the source frame rate and target frame rate of the VENC are set to 30 fps and 6 fps respectively and the source frame rate and target frame rate of the RC are set to 30 fps and 10 fps respectively, the frame rate cannot be controlled accurately. It is recommended that the target frame rate of the VENC be greater than or equal to that of the RC. In this way, the expected target frame rate is obtained.
- When the VPSS is bound only to the VENC and the automatic mode is used, the required VPSS resources are reduced if the frame rate is controlled by the using the VENC.

[See Also]

[HI_MPI_VENC_CreateChn](#)

VENC_ATTR_H264_VBR_S

[Description]

Defines the VBR attribute structure of an H.264 encoding channel.

[Syntax]

```
typedef struct hivENC_ATTR_H264_VBR_S
{
    HI_U32      u32Gop;
    HI_U32      u32StatTime;
    HI_U32      u32SrcFrmRate;
```



```
    HI_U32      fr32DstFrmRate;
    HI_U32      u32MaxBitRate;
    HI_U32      u32MaxQp;
    HI_U32      u32MinQp;
    HI_U32      u32MinIQp;
}VENC_ATTR_H264_VBR_S;
```

[Member]

Member	Description
u32Gop	H264 GOP value Value range: [1, 65536]
u32StatTime	VBR statistics time, in second Value range: [1, 60]
u32SrcFrmRate	VI frame rate, in frame/s Value range: [1, 240]
fr32DstFrmRate	Output frame rate of the encoder, in frame/s Value range: [1/16, u32SrcFrmRate]
u32MaxBitRate	Maximum output bit rate of the encoder, in frame/s Value range for Hi3519 V100/Hi3519 V101: [2, 102400] Value range for Hi3516C V300: [2, 30720]
u32MaxQp	Maximum QP supported by the encoder Value range: [u32MinQp, 51]
u32MinQp	Minimum QP supported by the encoder Value range: [0, 51]
u32MinIQp	Minimum QP value for an I frame picture that is supported by the encoder. This member is used to control the maximum number of bits in an I frame. Value range: [u32MinQp, u32MaxQp]

[Note]

For details, see the descriptions of **u32SrcFrmRate** and **fr32DstFrmRate** in [VENC_ATTR_H264_CBR_S](#).

[See Also]

[HI_MPI_VENC_CreateChn](#)

VENC_ATTR_H264_AVBR_S

[Description]

Defines the AVBR attribute structure of an H.264 encoding channel.



[Syntax]

```
typedef struct hivENC_ATTR_H264_AVBR_S
{
    HI_U32      u32Gop;
    HI_U32      u32StatTime;
    HI_U32      u32SrcFrmRate;
    HI_FR32     fr32DstFrmRate;
    HI_U32      u32MaxBitRate;
}VENC_ATTR_H264_AVBR_S;
```

[Member]

Member	Description
u32Gop	H.264 GOP value Value range: [1, 65536]
u32StatTime	AVBR statistical time, in second Value range: [1, 60]
u32SrcFrmRate	VI frame rate, in frame/s Value range: [1, 240]
fr32DstFrmRate	Output frame rate of the encoder, in frame/s Value range: (0, u32SrcFrmRate]
u32MaxBitRate	Maximum output bit rate of the encoder, in kbit/s Value range for Hi3519 V100/Hi3519 V101: [2, 102400] Value range for Hi3516C V300: [2, 30720]

[Note]

For details, see the descriptions of **u32SrcFrmRate** and **fr32DstFrmRate** in [VENC_ATTR_H264_CBR_S](#).

[See Also]

[HI_MPI_VENC_CreateChn](#)

VENC_ATTR_H264_FIXQP_S

[Description]

Defines the FixQP attribute of an H.264 channel.

[Syntax]

```
typedef struct hivENC_ATTR_H264_FIXQP_S
{
    HI_U32      u32Gop;
    HI_U32      u32SrcFrmRate;
```



```
    HI_FR32      fr32DstFrmRate;
    HI_U32       u32IQp;
    HI_U32       u32PQp;
    HI_U32       u32BQp;
} VENC_ATTR_H264_FIXQP_S;
```

[Member]

Member	Description
u32Gop	H264 GOP value Value range: [1, 65536]
u32SrcFrmRate	VI frame rate, in frame/s Value range: [1, 240]
fr32DstFrmRate	Output frame rate of the encoder, in frame/s Value range: [1/16, u32SrcFrmRate]
u32IQp	QP values of all the macroblocks of I frames Value range: [0, 51]
u32PQp	QP values of all the macroblocks of P frames Value range: [0, 51]
u32BQp	QP values of all the macroblocks of B frames Value range: [0, 51]

[Note]

For details, see the descriptions of **u32SrcFrmRate** and **fr32DstFrmRate** in **VENC_ATTR_H264_CBR_S**.

[See Also]

[HI_MPI_VENC_CreateChn](#)

VENC_ATTR_H264_QPMAP_S

[Description]

Defines the structure of the QpMap attribute for an H.264 encoding channel.

[Syntax]

```
typedef struct hivENC_ATTR_H264_QPMAP_S
{
    HI_U32       u32Gop;
    HI_U32       u32StatTime;
    HI_U32       u32SrcFrmRate;
    HI_FR32      fr32DstFrmRate;
    VENC_RC_QPMAP_MODE_E enQpMapMode;
```



```
    HI_BOOL      bQpMapAbsQp;
    HI_U32       u32Reserved;
} VENC_ATTR_H264_QPMAP_S;
```

[Member]

Member	Description
u32Gop	H.264 GOP value Value range: [1, 65536]
u32StatTime	QpMap statistical time (in second) Value range: [1, 60]
u32SrcFrmRate	VI frame rate, in frame/s Value range: [1, 240]
fr32DstFrmRate	Output frame rate of the encoder, in frame/s Value range: [1/16, u32ViFrmRate]
enQpMapMode	Value assignment method of the QP value for CU32 or CU64
bQpMapAbsQp	QP mode of the CU32 or CU64 <ul style="list-style-type: none">• HI_FALSE: relative QP• HI_TRUE: absolute QP
u32Reserved;	Reserved

[Note]

For details, see the descriptions of **u32SrcFrmRate** and **fr32DstFrmRate** in [VENC_ATTR_H264_CBR_S](#).

[See Also]

[HI_MPI_VENC_CreateChn](#)

VENC_ATTR_MJPEG_FIXQP_S

[Description]

Defines the FixQp attribute of an MJPEG channel.

[Syntax]

```
typedef struct hiVENC_ATTR_MJPEG_FIXQP_S
{
    HI_U32       u32SrcFrmRate;
    HI_FR32      fr32DstFrmRate;
    HI_U32       u32Qfactor;
} VENC_ATTR_MJPEG_FIXQP_S;
```



[Member]

Member	Description
u32SrcFrmRate	VI frame rate, in frame/s Value range: [1, 240]
fr32DstFrmRate	Output frame rate of the encoder, in frame/s Value range: [1/16, u32ViFrmRate]
u32Qfactor	Qfactor for MJPEG encoding Value range: [1, 99]

[Note]

For details, see the descriptions of **u32SrcFrmRate** and **fr32DstFrmRate** in [VENC_ATTR_H264_CBR_S](#).

[See Also]

[HI_MPI_VENC_CreateChn](#)

VENC_ATTR_MJPEG_CBR_S

[Description]

Defines the CBR attribute of an MJPEG channel.

[Syntax]

```
typedef struct hiVENC_ATTR_MJPEG_CBR_S
{
    HI_U32      u32StatTime;
    HI_U32      u32SrcFrmRate;
    HI_FR32     fr32DstFrmRate;
    HI_U32      u32BitRate;
    HI_U32      u32FluctuateLevel;
} VENC_ATTR_MJPEG_CBR_S;
```

[Member]

Member	Description
u32StatTime	CBR statistics time, in second Value range: [1, 60]
u32SrcFrmRate	Input frame rate of the encoder, in frame/s Value range: [1, 240]
fr32DstFrmRate	Output frame rate of the encoder, in frame/s Value range: [1/16, u32SrcFrmRate]
u32BitRate	Average bit rate, in kbit/s



Member	Description
	Value range: [2, 102400]
u32FluctuateLevel	Fluctuation level of the maximum bit rate relative to the average bit rate. u32FluctuateLevel is reserved and not used currently. Value range: [1, 5] The fluctuation level 1 is recommended.

[Note]

For details, see the descriptions of **u32SrcFrmRate** and **fr32DstFrmRate** in [VENC_ATTR_H264_CBR_S](#).

[See Also]

[HI_MPI_VENC_CreateChn](#)

VENC_ATTR_MJPEG_VBR_S

[Description]

Defines the VBR attribute of an MJPEG channel.

[Syntax]

```
typedef struct hivENC_ATTR_MJPEG_VBR_S
{
    HI_U32      u32StatTime;
    HI_U32      u32SrcFrmRate;
    HI_FR32     fr32DstFrmRate;
    HI_U32      u32MaxBitRate;
    HI_U32      u32MaxQfactor;
    HI_U32      u32MinQfactor;
} VENC_ATTR_MJPEG_VBR_S;
```

[Member]

Member	Description
u32StatTime	VBR statistics time, in second Value range: [1, 60]
u32SrcFrmRate	Input frame rate of the encoder, in frame/s Value range: [1, 240]
fr32DstFrmRate	Output frame rate of the encoder, in frame/s Value range: [1/16, u32SrcFrmRate]
u32MaxBitRate	Maximum bit rate, in kbit/s Value range: [2, 102400]
u32MaxQfactor	Maximum quantization factor



Member	Description
	Value range: [1, 99]
u32MinQfactor	Minimum quantization factor Value range: [1, u32MaxQfactor]

[Note]

For details, see the descriptions of **u32SrcFrmRate** and **fr32DstFrmRate** in [VENC_ATTR_H264_CBR_S](#).

[See Also]

[HI_MPI_VENC_CreateChn](#)

VENC_ATTR_H265_CBR_S

[Description]

Defines the CBR attribute structure of an H.265 encoding channel.

[Syntax]

```
typedef struct hIVENC_ATTR_H265_CBR_S
{
    HI_U32      u32Gop;
    HI_U32      u32StatTime;
    HI_U32      u32SrcFrmRate;
    HI_FR32     fr32DstFrmRate;
    HI_U32      u32BitRate;
    HI_U32      u32FluctuateLevel;
} VENC_ATTR_H265_CBR_S;
```

[Member]

Member	Description
u32Gop	H.265 GOP value Value range: [1, 65536]
u32StatTime	CBR statistics time, in second Value range: [1, 60]
u32SrcFrmRate	VI frame rate, in frame/s Value range: [1, 240]
fr32DstFrmRate	Output frame rate of the encoder, in frame/s Value range: [1/16, u32SrcFrmRate]



Member	Description
u32BitRate	Average bit rate, in kbit/s Value range for Hi3519 V100/Hi3519 V101: [2, 102400] Value range for Hi3516C V300: [2, 30720]
u32FluctuateLevel	Fluctuation level of the maximum bit rate relative to the average bit rate. This member is reserved and not used currently. Value range: [1, 5] The fluctuation level 1 is recommended.

[Note]

For details, see the descriptions of **u32SrcFrmRate** and **fr32DstFrmRate** in [VENC_ATTR_H264_CBR_S](#).

[See Also]

[HI_MPI_VENC_CreateChn](#)

VENC_ATTR_H265_VBR_S

[Description]

Defines the VBR attribute structure of an H.265 encoding channel.

[Syntax]

```
typedef struct hiVENC_ATTR_H265_VBR_S
{
    HI_U32      u32Gop;
    HI_U32      u32StatTime;
    HI_U32      u32SrcFrmRate;
    HI_FR32     fr32DstFrmRate;
    HI_U32      u32MaxBitRate;
    HI_U32      u32MaxQp;
    HI_U32      u32MinQp;
    HI_U32      u32MinIQP;
} VENC_ATTR_H265_VBR_S;
```

[Member]

Member	Description
u32Gop	H.265 GOP value Value range: [1, 65536]
u32StatTime	VBR statistical time, in second Value range: [1, 60]



Member	Description
u32SrcFrmRate	VI frame rate, in frame/s Value range: [1, 240]
fr32DstFrmRate	Output frame rate of the encoder, in frame/s Value range: [1/16, u32SrcFrmRate]
u32MaxBitRate	Maximum output bit rate of the encoder, in kbit/s Value range for Hi3519 V100/Hi3519 V101: [2, 102400] Value range for Hi3516C V300: [2, 30720]
u32MaxQp	Maximum QP supported by the encoder Value range: [u32MinQp, 51]
u32MinQp	Minimum QP supported by the encoder Value range: [0, 51]
u32MinIQp	Minimum QP value for an I frame picture that is supported by the encoder. This member is used to control the maximum number of bits in an I frame. Value range: [u32MinQp, u32MaxQp]

[Note]

For details, see the descriptions of **u32SrcFrmRate** and **fr32DstFrmRate** in [VENC_ATTR_H264_CBR_S](#).

[See Also]

[HI_MPI_VENC_CreateChn](#)

VENC_ATTR_H265_AVBR_S

[Description]

Defines the AVBR attribute structure of an H.265 encoding channel.

[Syntax]

```
typedef struct hivENC_ATTR_H265_AVBR_S
{
    HI_U32      u32Gop;
    HI_U32      u32StatTime;
    HI_U32      u32SrcFrmRate;
    HI_FR32     fr32DstFrmRate;
    HI_U32      u32MaxBitRate;
} VENC_ATTR_H265_AVBR_S;
```

[Member]



Member	Description
u32Gop	H.265 GOP value Value range: [1, 65536]
u32StatTime	AVBR statistical time, in second Value range: [1, 60]
u32SrcFrmRate	VI frame rate, in fps Value range: [1, 240]
fr32DstFrmRate	Output frame rate of the encoder, in frame/s Value range: (0, u32SrcFrmRate]
u32MaxBitRate	Maximum output bit rate of the encoder, in kbit/s Value range for Hi3519 V100/Hi3519 V101: [2, 102400] Value range for Hi3516C V300: [2, 30720]

[Note]

For details, see the descriptions of **u32SrcFrmRate** and **fr32DstFrmRate** in [VENC_ATTR_H264_CBR_S](#).

[See Also]

[HI_MPI_VENC_CreateChn](#)

VENC_ATTR_H265_FIXQP_S

[Description]

Defines the FixQp attribute of an H.265 VENC channel.

[Syntax]

```
typedef struct hIVENC_ATTR_H265_FIXQP_S
{
    HI_U32      u32Gop;
    HI_U32      u32SrcFrmRate;
    HI_FR32     fr32DstFrmRate;
    HI_U32      u32IQp;
    HI_U32      u32PQp;
    HI_U32      u32BQp;
} VENC_ATTR_H265_FIXQP_S;
```

[Member]

Member	Description
u32Gop	H.265 GOP value Value range: [1, 65536]



Member	Description
u32SrcFrmRate	VI frame rate, in frame/s Value range: [1, 240]
fr32DstFrmRate	Output frame rate of the encoder, in frame/s Value range: [1/16, u32SrcFrmRate]
u32IQp	QP values of all the macroblocks of I frames Value range: [0, 51]
u32PQp	QP values of all the macroblocks of P frames Value range: [0, 51]
u32BQp	QP values of all the macroblocks of B frames Value range: [0, 51]

[Note]

For details, see the descriptions of **u32SrcFrmRate** and **fr32DstFrmRate** in [VENC_ATTR_H264_CBR_S](#).

[See Also]

[HI_MPI_VENC_CreateChn](#)

VENC_ATTR_H265_QPMAP_S

[Description]

Defines the structure of the QpMap attribute for an H.265 encoding channel.

[Syntax]

```
typedef struct hivENC_ATTR_H265_QPMAP_S
{
    HI_U32      u32Gop;
    HI_U32      u32StatTime;
    HI_U32      u32SrcFrmRate;
    HI_FR32     fr32DstFrmRate;
    VENC_RC_QPMAP_MODE_E enQpMapMode;
    HI_BOOL     bQpMapAbsQp;
    HI_U32      u32Reserved;
} VENC_ATTR_H265_QPMAP_S;
```

[Member]

Member	Description
u32Gop	H.265 GOP value Value range: [1, 65536]



Member	Description
u32StatTime	QpMap statistical time (in second) Value range: [1, 60]
u32SrcFrmRate	VI frame rate, in frame/s Value range: [1, 240]
fr32DstFrmRate	Output frame rate of the encoder, in frame/s Value range: [1/16, u32SrcFrmRate]
enQpMapMode	Value assignment method of the QP value for CU32 or CU64
bQpMapAbsQp	QP mode of the CU32 or CU64 <ul style="list-style-type: none">• HI_FALSE: relative QP• HI_TRUE: absolute QP
u32Reserved	Reserved

[Note]

For details, see the descriptions of **u32SrcFrmRate** and **fr32DstFrmRate** in [VENC_ATTR_H265_CBR_S](#).

[See Also]

[HI_MPI_VENC_CreateChn](#)

VENC_SUPERFRM_MODE_E

[Description]

Defines the mode of processing jumbo frames when the bit rate is being controlled.

[Syntax]

```
typedef enum hiRC_SUPERFRM_MODE_E
{
    SUPERFRM_NONE=0,
    SUPERFRM_DISCARD,
    SUPERFRM_REENCODE,
    SUPERFRM_BUTT
}VENC_SUPERFRM_MODE_E;
```

[Member]

Member	Description
SUPERFRM_NONE	No special measure is taken.
SUPERFRM_DISCARD	Jumbo frames are discarded.
SUPERFRM_REENCODE	Jumbo frames are re-encoded.



[Note]

None

[See Also]

[HI_MPI_VENC_SetSuperFrameCfg](#)

VENC_PARAM_H264_VBR_S

[Description]

Defines the configurations of the advanced parameters related to the VBR control mode of an H.264 channel.

[Syntax]

```
typedef struct hiVENC_PARAM_H264_VBR_S
{
    HI_S32    s32ChangePos;
    HI_U32    u32MinIprop;
    HI_U32    u32MaxIprop;
    HI_S32    s32MaxReEncodeTimes;
    HI_BOOL   bQpMapEn;
} VENC_PARAM_H264_VBR_S;
```

[Member]

Member	Description
s32ChangePos	Ratio of the bit rate when the QP starts to be adjusted in VBR mode relative to the maximum bit rate Value range: [50, 100]
u32MinIprop	Minimum ratio of the I/P frame bit rates Value range: [1, 100]
u32MaxIprop	Maximum ratio of the I/P frame bit rate Value range: [u32MinIprop, 100]
s32MaxReEncodeTimes	Number of times that each frame is re-encoded. The value 0 indicates that the frame is not re-encoded. Value range: [0, 3]
bQpMapEn	Whether to enable the QpMap function in VBR mode Value range: HI_TRUE or HI_FALSE

[Note]

Hi3519 V100 does not support texture-level QpMap.

[See Also]



- [HI_MPI_VENC_GetRcParam](#)
- [HI_MPI_VENC_SetRcParam](#)

VENC_PARAM_H264_AVBR_S

[Description]

Defines the configuration of the advanced parameters related to the AVBR control mode of an H.264 encoding channel.

[Syntax]

```
typedef struct hivenc_param_h264_avbr_s
{
    HI_S32    s32ChangePos;
    HI_U32    u32MinIprop;
    HI_U32    u32MaxIprop;
    HI_S32    s32MaxReEncodeTimes;
    HI_BOOL   bQpMapEn;
    HI_S32    s32MinStillPercent;
    HI_U32    u32MaxStillQP;
    HI_U32    u32MinStillPSNR;
    HI_U32    u32MaxQp;
    HI_U32    u32MinQp;
    HI_U32    u32MaxIQp;
    HI_U32    u32MinIQp;
} VENC_PARAM_H264_AVBR_S;
```

[Member]

Member	Description
s32ChangePos	Ratio of the bit rate when the QP starts to be adjusted in AVBR mode relative to the maximum bit rate Value range: [50, 100] Default value: 90
u32MinIprop	Minimum ratio of the I frame bit rate to the P frame bit rate Value range: [1, 100] Default value: 1
u32MaxIprop	Maximum ratio of the I frame bit rate to the P frame bit rate Value range: [u32MinIprop, 100] Default value: 100
s32MaxReEncodeTimes	Number of times that each frame is re-encoded. The value 0 indicates that the frame is not re-encoded. Value range: [0, 3] Default value: 2



Member	Description
bQpMapEn	Whether to enable the QpMap function in AVBR mode Value range: HI_TRUE or HI_FALSE Default value: HI_FALSE
s32MinStillPercent	Minimum percentage of the target bit rate in the static scenario. If this member is set to 100 , the target bit rate will not be proactively reduced in AVBR mode when the scenario is judged to be a static one, and the effect of the AVBR mode is consistent with that of the VBR mode. Value range: [20, 100] Default value: 25
u32MaxStillQP	Maximum QP value of the I frame in the static scenario Value range: [MinIQp, MaxIQp] Default value: 35
u32MinStillPSNR	Invalid member currently, which can only be set to 0
u32MaxQp	Maximum QP value of the P frame and B frame Value range: [MinQp, 51] Default value: 51
u32MinQp	Minimum QP value of the P frame and B frame Value range: [0, 51] Default value: 16
u32MaxIQp	Maximum QP of an I frame Value range: [MinIQp, 51] Default value: 51
u32MinIQP	Minimum QP of an I frame Value range: [0, 51] Default value: 16

[Note]

None

[See Also]

- [HI_MPI_VENC_GetRcParam](#)
- [HI_MPI_VENC_SetRcParam](#)

VENC_PARAM_H264_CBR_S

[Description]

Defines the configurations of the advanced parameters related to the new CBR control mode of an H.264 channel.



[Syntax]

```
typedef struct hiVENC_PARAM_H264_CBR_S
{
    HI_U32    u32MinIprop;
    HI_U32    u32MaxIprop;
    HI_U32    u32MaxQp;
    HI_U32    u32MinQp;
    HI_U32    u32MaxIQp;
    HI_U32    u32MinIQp;
    HI_S32    s32QualityLevel;
    HI_S32    s32MaxReEncodeTimes;
    HI_BOOL   bQpMapEn;
}VENC_PARAM_H264_CBR_S;
```

[Member]

Member	Description
u32MinIprop	Minimum ratio of I frames to P frames Value range: [1, 100]
u32MaxIprop	Maximum ratio of I frames to P frames Value range: [u32MinIprop, 100]
u32MaxQp	Maximum frame QP value for controlling the picture quality Value range: [u32MinQp, 51]
u32MinQp	Minimum frame QP value for controlling bit rate fluctuation Value range: [0, 51]
s32QualityLevel	Quality level A smaller value indicates better quality. Value range: [1, 5]
s32MaxReEncodeTimes	Number of times that each frame is re-encoded The value 0 indicates that the frame is not re-encoded. Value range: [0, 3]
u32MinIQp	Minimum QP of an I frame for controlling the maximum number of bits of an I frame Value range: [0, 51]
u32MaxIQp	Maximum QP of an I frame for controlling the minimum number of bits of an I frame Value range: [u32MinIQp, 51]
bQpMapEn	Whether to enable the QpMap function in CBR mode Value range: HI_TRUE or HI_FALSE



[Note]

- Hi3519 V100 does not support texture-level QpMap.
- The **u32MaxIQp** parameter of Hi3519 V100 needs to be separately configured.
When **u32MaxIQp** falls within [u32MinIQp, 51], the maximum QP value of the I frame is **u32MaxIQp** and that of other frames is **u32MaxQp**.
When **u32MaxIQp** is 0xFFFFFFFF, the maximum QP value of all frames is **u32MaxQp**.
- The value range of **u32MaxIQp** of Hi3519 V101 and Hi3516C V300 is [u32MinIQp, 51].

[See Also]

- [HI_MPI_VENC_GetRcParam](#)
- [HI_MPI_VENC_SetRcParam](#)

VENC_PARAM_MJPEG_CBR_S

[Description]

Defines the configurations of the advanced parameters related to the CBR control mode of an MJPEG channel.

[Syntax]

```
typedef struct hiVENC_PARAM_MJPEG_CBR_S
{
    HI_U32 u32MaxQfactor;
    HI_U32 u32MinQfactor;
} VENC_PARAM_MJPEG_CBR_S;
```

[Member]

Member	Description
u32MaxQfactor	Maximum frame Qfactor value for controlling the picture quality Value range: [u32MinQfactor, 99]
u32MinQfactor	Minimum frame Qfactor for controlling the picture quality Value range: [1, 99]

[Note]

None

[See Also]

- [HI_MPI_VENC_GetRcParam](#)
- [HI_MPI_VENC_SetRcParam](#)

VENC_PARAM_MJPEG_VBR_S

[Description]



Defines the configurations of the advanced parameters related to the VBR control mode of an MJPEG channel.

[Syntax]

```
typedef struct hiVENC_PARAM_MJPEG_VBR_S
{
    HI_S32 s32ChangePos;
} VENC_PARAM_MJPEG_VBR_S;
```

[Member]

Member	Description
s32ChangePos	Ratio of the bit rate when the Qfactor starts to be adjusted in VBR mode relative to the maximum bit rate Value range: [50, 100]

[Note]

None

[See Also]

- [HI_MPI_VENC_GetRcParam](#)
- [HI_MPI_VENC_SetRcParam](#)

VENC_PARAM_H265_VBR_S

[Description]

Defines the configurations of the advanced parameters related to the VBR control mode of an H.265 encoding channel.

[Syntax]

```
typedef struct hiVENC_PARAM_H265_VBR_S
{
    HI_S32 s32ChangePos;
    HI_U32 u32MinIprop;
    HI_U32 u32MaxIprop;
    HI_S32 s32MaxReEncodeTimes;
    HI_BOOL bQpMapEn;
    VENC_RC_QPMAP_MODE_E enQpMapMode;
} VENC_PARAM_H265_VBR_S;
```

[Member]



Member	Description
s32ChangePos	Ratio of the bit rate when the QP starts to be adjusted in VBR mode relative to the maximum bit rate Value range: [50, 100]
u32MinIprop	Minimum ratio of the I frame bit rate to the P frame bit rate Value range: [1, 100]
u32MaxIprop	Maximum ratio of the I frame bit rate to the P frame bit rate Value range: [u32MinIprop, 100]
s32MaxReEncodeTimes	Number of times that each frame is re-encoded. The value 0 indicates that the frame is not re-encoded. Value range: [0, 3]
bQpMapEn	Whether to enable the QpMap function in VBR mode Value range: HI_TRUE or HI_FALSE
enQpMapMode	Method of assigning the QP values for CU32 and CU64 in QpMap mode

[Note]

Hi3519 V100 does not support texture-level QpMap.

[See Also]

- [HI_MPI_VENC_GetRcParam](#)
- [HI_MPI_VENC_SetRcParam](#)

VENC_PARAM_H265_AVBR_S

[Description]

Defines the configurations of the advanced parameters related to the AVBR control mode of an H.265 encoding channel.

[Syntax]

```
typedef struct hivenc_param_h265_avbr_s
{
    HI_S32    s32ChangePos;
    HI_U32    u32MinIprop;
    HI_U32    u32MaxIprop;
    HI_S32    s32MaxReEncodeTimes;
    HI_BOOL   bQpMapEn;
    VENC_RC_QPMAP_MODE_E enQpMapMode;
    HI_S32    s32MinStillPercent;
    HI_U32    u32MaxStillQP;
    HI_U32    u32MinStillPSNR;
    HI_U32    u32MaxQp;
```



```
    HI_U32  u32MinQp;  
    HI_U32  u32MaxIQp;  
    HI_U32  u32MinIQP;  
};VENC_PARAM_H265_AVBR_S;
```

[Member]

Member	Description
s32ChangePos	Ratio of the bit rate when the QP starts to be adjusted relative to the maximum bit rate Value range: [50, 100] Default value: 90
u32MinIprop	Minimum ratio of the I frame bit rate to the P frame bit rate Value range: [1, 100] Default value: 1
u32MaxIprop	Maximum ratio of the I frame bit rate to the P frame bit rate Value range: [u32MinIprop, 100] Default value: 100
s32MaxReEncodeTimes	Number of times that each frame is re-encoded. The value 0 indicates that the frame is not re-encoded. Value range: [0, 3] Default value: 2
bQpMapEn	Whether to enable the QpMap function in AVBR mode Value range: HI_TRUE or HI_FALSE Default value: HI_FALSE
enQpMapMode	Method of assigning the QP values for CU32 and CU64 in QpMap mode
s32MinStillPercent	Minimum percentage of the target bit rate in the static scenario. If this member is set to 100 , the target bit rate will not be proactively reduced in AVBR mode when the scenario is judged to be a static one, and the effect of the AVBR mode is consistent with that of the VBR mode. Value range: [20, 100] Default value: 25
u32MaxStillQP	Maximum QP value of the I frame in the static scenario Value range: [MinIQp, MaxIQp] Default value: 35
u32MinStillPSNR	Invalid member currently, which can only be set to 0
u32MaxQp	Maximum QP of a P frame Value range: [MinQp, 51] Default value: 51



Member	Description
u32MinQp	Minimum QP of a P frame Value range: [0, 51] Default value: 16
u32MaxIQp	Maximum QP of an I frame Value range: [MinIQp, 51] Default value: 51
u32MinIQP	Minimum QP of an I frame Value range: [0, 51] Default value: 16

[Note]

None

[See Also]

- [HI_MPI_VENC_GetRcParam](#)
- [HI_MPI_VENC_SetRcParam](#)

VENC_PARAM_H265_CBR_S

[Description]

Defines the configurations of the advanced parameters related to the CBR control mode of an H.265 encoding channel.

[Syntax]

```
typedef struct hiVENC_PARAM_H265_CBR_S
{
    HI_U32    u32MinIprop;
    HI_U32    u32MaxIprop;
    HI_U32    u32MaxQp;
    HI_U32    u32MinQp;
    HI_U32    u32MaxIQp;
    HI_U32    u32MinIQp;
    HI_S32    s32QualityLevel;
    HI_S32    s32MaxReEncodeTimes;
    HI_BOOL   bQpMapEn;
    VENC_RC_QPMAP_MODE_E enQpMapMode;
} VENC_PARAM_H265_CBR_S;
```

[Member]



Member	Description
u32MinIprop	Minimum ratio of I frames to P frames Value range: [1, 100]
u32MaxIprop	Maximum ratio of I frames to P frames Value range: [u32MinIprop, 100]
u32MaxQp	Maximum frame QP value for controlling the picture quality Value range: [u32MinQp, 51]
u32MinQp	Minimum frame QP value for controlling bit rate fluctuation Value range: [0, 51]
s32QualityLevel	Quality level. A smaller value indicates better quality Value range: [1, 5]
s32MaxReEncodeTimes	Number of times that each frame is re-encoded. The value 0 indicates that the frame is not re-encoded. Value range: [0, 3]
u32MinIQp	Minimum QP of an I frame for controlling the maximum number of bits of an I frame Value range: [0, 51]
u32MaxIQp	Maximum QP of an I frame for controlling the minimum number of bits of an I frame Value range: [u32MinIQp, 51]
bQpMapEn	Whether to enable or disable the QpMap function in CBR mode Value range: HI_TRUE or HI_FALSE
enQpMapMode	Method of assigning the QP values for CU32 and CU64 when the QpMap table is enabled

[Note]

- Hi3519 V100 does not support texture-level QpMap.
- The **u32MaxIQp** parameter of Hi3519 V100 needs to be separately configured.
 - When **u32MaxIQp** falls within [u32MinIQp, 51], the maximum QP value of the I frame is **u32MaxIQp** and that of other frames is **u32MaxQp**.
 - When **u32MaxIQp** is 0xFFFFFFFF, the maximum QP value of all frames is **u32MaxQp**.
- The value range of **u32MaxIQp** of Hi3519 V101 and Hi3516C V300 is [u32MinIQp, 51].

[See Also]

- [HI_MPI_VENC_GetRcParam](#)
- [HI_MPI_VENC_SetRcParam](#)



VENC_RC_PARAM_S

[Description]

Defines the advanced parameters for controlling the bit rate of a VENC channel.

[Syntax]

```
typedef struct hiVENC_RC_PARAM_S
{
    HI_U32 u32ThrdI[RC_TEXTURE_THR_SIZE];
    HI_U32 u32ThrdP[RC_TEXTURE_THR_SIZE];
    HI_U32 u32ThrdB[RC_TEXTURE_THR_SIZE];
    HI_U32 u32QpDeltaLevelI[RC_TEXTURE_THR_SIZE];
    HI_U32 u32QpDeltaLevelP[RC_TEXTURE_THR_SIZE];
    HI_U32 u32QpDeltaLevelB[RC_TEXTURE_THR_SIZE];
    HI_U32 u32RowQpDelta;
    HI_S32 s32FirstFrameStartQp;
    union
    {
        VENC_PARAM_H264_CBR_S stParamH264Cbr;
        VENC_PARAM_H264_VBR_S stParamH264Vbr;
        VENC_PARAM_H264_AVBR_S stParamH264AVbr;
        VENC_PARAM_MJPEG_CBR_S stParamMjpegCbr;
        VENC_PARAM_MJPEG_VBR_S stParamMjpegVbr;
        VENC_PARAM_H265_CBR_S stParamH265Cbr;
        VENC_PARAM_H265_VBR_S stParamH265Vbr;
        VENC_PARAM_H265_AVBR_S stParamH265AVbr;
    };
    HI_VOID* pRcParam;
}VENC_RC_PARAM_S;
```

[Member]

Member	Description
u32ThrdI	Mad threshold for controlling the macroblock-level bit rate of I frames Value range: [0, 255]
u32ThrdP	Mad threshold for controlling the macroblock-level bit rate of P frames Value range: [0, 255]
u32ThrdB	Mad threshold for controlling the macroblock-level bit rate of B frames Value range: [0, 255]
u32QpDeltaLevelI	Mad QP delta level for controlling the macroblock-level bit rate of I frames



Member	Description
	Value range: [0, 3]
u32QpDeltaLevelP	Mad QP delta level for controlling the macroblock-level bit rate of P frames Value range: [0, 3]
u32QpDeltaLevelB	Mad QP delta level for controlling the macroblock-level bit rate of B frames Value range: [0, 3]
u32RowQpDelta	Fluctuation amplitude of the start QP value of each macroblock row relative to the start QP value of the frame during macroblock-level bit rate control Value range: [0, 10]
s32FirstFrameStartQp	Start QP value of the first frame, valid in CBR mode, VBR mode and AVBR mode. Value range: [MinIQP, MaxIQP] and -1 for the CBR mode [MinIQP, MaxQp] and -1 for the VBR mode [MinIQP, MaxIQp] and -1 for the AVBR mode If s32FirstFrameStartQp is -1, the start QP value of the first frame is internally calculated by the encoder. If s32FirstFrameStartQp is set to any other valid value, the user specifies this valid value as the start QP value of the first frame.
stParamH264Cbr	Advanced parameter of the CBR control mode for an H.264 channel
stParamH264VBR	Advanced parameter of the VBR control mode for an H.264 channel
stParamH264AVbr	Advanced parameter of the AVBR control mode for an H.264 channel
stParamMjpegCbr	Advanced parameter of the CBR control mode for an MJPEG channel
stParamMjpegVbr	Advanced parameter of the VBR control mode of an MJPEG channel
stParamH265Cbr	Advanced parameter of the CBR control mode of an H.265 channel
stParamH265Vbr	Advanced parameter of the VBR control mode of an H.265 channel
stParamH265AVbr	Advanced parameter of the AVBR control mode of an H.265 channel
pRcParam	Reserved

[Note]

None

[See Also]



- [HI_MPI_VENC_SetRcParam](#)
- [HI_MPI_VENC_GetRcParam](#)

VENC_CROP_CFG_S

[Description]

Defines the crop parameters of a VENC channel.

[Syntax]

```
typedef struct hiVENC_CROP_CFG_S
{
    HI_BOOL bEnable;      /* Crop region enable */
    RECT_S stRect;        /* Crop region, note: s32X must be multi of 16 */
}VENC_CROP_CFG_S;
```

[Member]

Member	Description
bEnable	Crop enable Value range: [HI_FALSE, HI_TRUE] HI_TRUE: enabled HI_FALSE: disabled
stRect	Crop region stRect.s32X : The parameter value must be 16-pixel-aligned. stRect.s32Y : The parameter value must be 2-pixel-aligned. stRect.u32Width and s32Rect.u32Height meet the width and height requirements of a VENC channel.

[Note]

None

[See Also]

- [HI_MPI_VENC_SetCrop](#)
- [HI_MPI_VENC_GetCrop](#)

VENC_FRAME_RATE_S

[Description]

Defines the parameters for controlling the frame rate of a VENC channel.

[Syntax]

```
typedef struct hiVENC_FRAME_RATE_S
{
    HI_S32 s32SrcFrmRate;
```



```
    HI_S32 s32DstFrmRate;  
} VENC_FRAME_RATE_S;
```

[Member]

Member	Description
s32SrcFrmRate	Input frame rate of a VENC channel, in frame/s Value range: -1 or [1, 240]
s32DstFrmRate	Output frame rate of a VENC channel, in frame/s Value range for JPEG: -1 or [1, s32SrcFrmRate] Value range for H.264/H.265/MJPEG: -1 or [1, 240]

[Note]

None

[See Also]

- [HI_MPI_VENC_SetFrameRate](#)
- [HI_MPI_VENC_GetFrameRate](#)

VENC_COLOR2GREY_S

[Description]

Defines the color-to-gray function of a VENC channel.

[Syntax]

```
typedef struct hivENC_COLOR2GREY_S  
{  
    HI_BOOL bColor2Grey;           /* Whether to enable Color2Grey. */  
}VENC_COLOR2GREY_S;
```

[Member]

Member	Description
bColor2Grey	Color-to-gray enable for a VENC channel

[Note]

None

[See Also]

- [HI_MPI_VENC_SetColor2Grey](#)
- [HI_MPI_VENC_GetColor2Grey](#)



VENC_JPEG_SNAP_MODE_E

[Description]

Defines the snapshot mode of a JPEG VENC channel.

[Syntax]

```
typedef enum hiVENC_JPEG_SNAP_MODE_E
{
    JPEG_SNAP_ALL      = 0,
    JPEG_SNAP_FLASH   = 1,
    JPEG_SNAP_BUTT,
} VENC_JPEG_SNAP_MODE_E;
```

[Member]

Member	Description
JPEG_SNAP_ALL	Snapshot mode in which all captured pictures are encoded. This mode is the default snapshot mode of a JPEG channel.
JPEG_SNAP_FLASH	Snapshot mode in which only the pictures are captured when the camera flash is on are encoded. This mode is available only when the front-end camera flash works.

[Note]

None

[See Also]

- [HI_MPI_VENC_SetJpegSnapMode](#)
- [HI_MPI_VENC_GetJpegSnapMode](#)

VENC_RECV_PIC_PARAM_S

[Description]

Defines the number of received frames to be encoded.

[Syntax]

```
typedef struct hiVENC_RECV_PIC_PARAM_S
{
    HI_S32 s32RecvPicNum;
} VENC_RECV_PIC_PARAM_S;
```

[Member]

Member	Description
s32RecvPicNum	Number of received frames to be encoded by the encoding channel



[Note]

None

[See Also]

[HI_MPI_VENC_StartRecvPicEx](#)

H264E_IDR_PIC_ID_MODE_E

[Description]

Defines the mode of setting the idr_pic_id of an IDR frame or I frame.

[Syntax]

```
typedef enum hiH264E_IDR_PIC_ID_MODE_E
{
    H264E_IDR_PIC_ID_MODE_AUTO = 0,
    H264E_IDR_PIC_ID_MODE_USR,
    H264E_IDR_PIC_ID_MODE_BUTT,
} H264E_IDR_PIC_ID_MODE_E;
```

[Member]

Member	Description
H264E_IDR_PIC_ID_MODE_AUTO	Automatic mode
H264E_IDR_PIC_ID_MODE_USR	User-defined mode

[Note]

None

[See Also]

[HI_MPI_VENC_SetH264IdrPicId](#)

VENC_H264_IDRPICID_CFG_S

[Description]

Defines the parameters for setting the idr_pic_id of an IDR frame or I frame.

[Syntax]

```
typedef struct hiVENC_H264_IDRPICID_CFG_S
{
    H264E_IDR_PIC_ID_MODE_E enH264eIdrPicIdMode;
    HI_U32 u32H264eIdrPicId;
} VENC_H264_IDRPICID_CFG_S;
```

[Member]



Member	Description
enH264IdrPicIdMode	Mode of setting the idr_pic_id
u32H264IdrPicId	idr_pic_id value Value range: [0, 65535]

[Note]

None

[See Also]

[HI_MPI_VENC_SetH264IdrPicId](#)

VENC_PARAM_H265_SLICE_SPLIT_S

[Description]

Defines the slice split attribute of an H.265 VENC channel.

[Syntax]

```
typedef struct hiVENC_PARAM_H265_SLICE_SPLIT_S
{
    HI_BOOL bSplitEnable;
    HI_U32 u32SplitMode;
    HI_U32 u32SliceSize;
    HI_U32 loop_filter_across_slices_enabled_flag;
} VENC_PARAM_H265_SLICE_SPLIT_S;
```

[Member]

Member	Description
bSplitEnable	Slice split enable
u32SplitMode	Slice split mode 0: by byte 1: by LCU line The value that is greater than or equal to 2 is not supported.
u32SliceSize	Number of bytes in each slice when u32SplitMode is 0 Minimum value: 128 Maximum value: min((0xFFFF), u32PicSize/2) The picture size is calculated as follows: u32PicSize = picwidth x picheight x 3/2 Number of LCU lines occupied by each slice when u32SplitMode is 1 Minimum value: 1 Maximum value: (Picture height + 63)/64



Member	Description
loop_filter_across_slice_s_enabled_flag	Whether filtering is performed on the left and top borders of slices

[Note]

None

[See Also]

- [HI_MPI_VENC_SetH265SliceSplit](#)
- [HI_MPI_VENC_GetH265SliceSplit](#)

VENC_PARAM_H265_PU_S

[Description]

Defines the PU attribute of an H.265 VENC channel.

[Syntax]

```
typedef struct hivENC_PARAM_H265_PU_S
{
    HI_BOOL bPu32x32En;
    HI_BOOL bPu16x16En;
    HI_BOOL bPu8x8En;
    HI_BOOL bPu4x4En;
    HI_U32 constrained_intra_pred_flag;
    HI_U32 strong_intra_smoothing_enabled_flag;
    HI_U32 pcm_enabled_flag;
    HI_U32 pcm_loop_filter_disabled_flag;
    HI_U32 u32MaxNumMergeCand;
}VENC_PARAM_H265_PU_S;
```

[Member]

Member	Description
bPu32x32En	PU32x32 enable Value range: 0 or 1 0: disabled 1: enabled (default)
bPu16x16En	PU16x16 enable Value range: 0 or 1 0: disabled 1: enabled (default)



Member	Description
bPu8x8En	PU8x8 enable Value range: 0 or 1 0: disabled 1: enabled (default)
bPu4x4En	PU4x4 enable Value range: 0 or 1 0: disabled 1: enabled (default)
constrained_intra_pred_flag	Value range: 0 (default) or 1
strong_intra_smoothing_enable_d_flag	Value range: 0 or 1 (default)
pcm_enabled_flag	Value range: 0 (default) or 1 The default value varies according to the chip type.
pcm_loop_filter_disabled_flag	Value range: 0 or 1 (default) The default value varies according to the chip type. This member is valid only when pcm_enabled_flag is 1. pcm_loop_filter_disabled_flag must be 1 for the Hi3519 V100/Hi3519 V101/Hi3516C V300.
u32MaxNumMergeCand	Value range: [2, 5] Default value: 2 The default value varies according to the chip type. u32MaxNumMergeCand must be 2 for the Hi3519 V100/Hi3519 V101/Hi3516C V300.

[Note]

None

[See Also]

- [HI_MPI_VENC_SetH265PredUnit](#)
- [HI_MPI_VENC_GetH265PredUnit](#)

VENC_PARAM_H265_TRANS_S

[Description]

Defines the transformation and quantization attribute of an H.265 VENC channel.

[Syntax]

```
typedef struct hivenc_param_h265_trans_s
{
```



```
HI_U32 transquant_bypass_enabled_flag;
HI_U32 transform_skip_enabled_flag;
HI_S32 cb_qp_offset;
HI_S32 cr_qp_offset;

HI_BOOL bScalingListTu4Valid;
HI_U8   InterScalingList4X4[2][16];
HI_U8   IntraScalingList4X4[2][16];

HI_BOOL bScalingListTu8Valid;
HI_U8   InterScalingList8X8[2][64];
HI_U8   IntraScalingList8X8[2][64];

HI_BOOL bScalingListTu16Valid;
HI_U8  IntraTu16Dc[2];
HI_U8  InterTu16Dc[2];
HI_U8  InterScalingList16X16[2][64];
HI_U8  IntraScalingList16X16[2][64];

HI_BOOL bScalingListTu32Valid;
HI_U8  IntraTu32Dc;
HI_U8  InterTu32Dc;
HI_U8  InterScalingList32X32[64];
HI_U8  IntraScalingList32X32[64];
} VENC_PARAM_H265_TRANS_S;
```

[Member]

Member	Description
transquant_bypass_enabled_flag	Value range: 0 (default) or 1 The default value varies according to the chip type. The value must be 0 for the Hi3519 V100/Hi3519 V101/Hi3516C V300.
transform_skip_enabled_flag	Value range: 0 (default) or 1 The default value varies according to the chip type. The value must be 0 for the Hi3519 V100/Hi3519 V101/Hi3516C V300.
cb_qp_offset	Value range: [-12, +12] Default value: 0 The default value varies according to the chip type.
cr_qp_offset	Value range: [-12, +12] Default value: 0 The default value varies according to the chip type.



Member	Description
bScalingListTu4Valid	Value range: [0, 1] Default value: 0 The default value varies according to the chip type.
InterScalingList4X4[2][16]	Value range: [1, 31] The default value varies according to the chip type.
IntraScalingList4X4[2][16]	Value range: [1, 31] The default value varies according to the chip type.
bScalingListTu8Valid	Value range: [0, 1] Default value: 0 The default value varies according to the chip type.
InterScalingList8X8[2][64]	Value range: [1, 31] The default value varies according to the chip type.
IntraScalingList8X8[2][64]	Value range: [1, 31] The default value varies according to the chip type.
bScalingListTu16Valid	Value range: [0, 1] Default value: 0 The default value varies according to the chip type.
IntraTu16Dc[2]	Value range: [1, 31] Default value: 6 The default value varies according to the chip type.
InterTu16Dc[2]	Value range: [1, 31] The default value varies according to the chip type.
InterScalingList16X16 [2][64]	Value range: [1, 31] The default value varies according to the chip type.
IntraScalingList16X16 [2][64]	Value range: [1, 31] The default value varies according to the chip type.
bScalingListTu32Valid	Value range: [0, 1] Default value: 0 The default value varies according to the chip type.
IntraTu32Dc	Value range: [1, 31] The default value varies according to the chip type.
InterTu32Dc	Value range: [1, 31] Default value: 6 The default value varies according to the chip type.



Member	Description
InterScalingList32X32 [64]	Value range: [1, 31] The default value varies according to the chip type.
IntraScalingList32X32 [64]	Value range: [1, 31] The default value varies according to the chip type.

[Note]

None

[See Also]

- [HI_MPI_VENC_SetH265Trans](#)
- [HI_MPI_VENC_GetH265Trans](#)

VENC_PARAM_H265_ENTROPY_S

[Description]

Defines the entropy encoding attribute of an H.265 VENC channel.

[Syntax]

```
typedef struct hiVENC_PARAM_H265_ENTROPY_S
{
    HI_U32 cabac_init_flag;
} VENC_PARAM_H265_ENTROPY_S;
```

[Member]

Member	Description
cabac_init_flag	Value range: 0 or 1 (default)

[Note]

None

[See Also]

- [HI_MPI_VENC_SetH265Entropy](#)
- [HI_MPI_VENC_GetH265Entropy](#)

VENC_PARAM_H265_DBLK_S

[Description]

Defines the deblocking attribute of an H.265 VENC channel.

[Syntax]

```
typedef struct hiVENC_PARAM_H265_DBLK_S
```



```
{  
    HI_U32 slice_deblocking_filter_disabled_flag;  
    HI_S32 slice_beta_offset_div2;  
    HI_S32 slice_tc_offset_div2;  
} VENC_PARAM_H265_DBLK_S;
```

[Member]

Member	Description
slice_deblocking_filter_disabled_flag	Value range: 0 (default) or 1
slice_tc_offset_div2	Value range: [-6, +6] Default value: 0
slice_beta_offset_div2	Value range: [-6, +6] Default value: 0

[Note]

None

[See Also]

- [HI_MPI_VENC_SetH265Dblk](#)
- [HI_MPI_VENC_GetH265Dblk](#)

VENC_PARAM_H265_SAO_S

[Description]

Defines the SAO attribute of an H.265 VENC channel.

[Syntax]

```
typedef struct hivENC_PARAM_H265_SAO_S  
{  
    HI_U32 slice_sao_luma_flag;  
    HI_U32 slice_sao_chroma_flag;  
} VENC_PARAM_H265_SAO_S;
```

[Member]

Member	Description
slice_sao_luma_flag	Value range: 0 or 1 (default)
slice_sao_chroma_flag	Value range: 0 or 1 (default)

[Note]

None



[See Also]

- [HI_MPI_VENC_SetH265Sao](#)
- [HI_MPI_VENC_GetH265Sao](#)

VENC_PARAM_H265_TIMING_S

[Description]

Defines the timing attribute of an H.265 VENC channel.

[Syntax]

```
typedef struct hivENC_PARAM_H265_TIMING_S
{
    HI_S32 timing_info_present_flag;
    HI_S32 num_units_in_tick;
    HI_S32 time_scale;
    HI_U32 num_ticks_poc_diff_one;
}VENC_PARAM_H265_TIMING_S;
```

[Member]

Member	Description
timing_info_present_flag	For details, see the H.265 protocol. Value range: 0 (default) or 1
num_units_in_tick	For details, see the H.265 protocol. Value range: greater than 0 Default value: 1
time_scale	For details, see the H.265 protocol. Value range: greater than 0 Default value: 60
num_ticks_poc_diff_one	For details, see the H.265 protocol. Value range: [0, 4294967294] Default value: 1

[Note]

None

[See Also]

- [HI_MPI_VENC_SetH265Timing](#)
- [HI_MPI_VENC_GetH265Timing](#)

VENC_FRAMELOST_MODE_E

[Description]



Defines the frame discarding mode when the instantaneous bit rate of a VENC channel is above the threshold.

[Syntax]

```
typedef enum hiVENC_FRAMELOST_MODE_E
{
    FRMLOST_NORMAL=0,
    FRMLOST_PSKIP,
    FRMLOST_BUTT,
} VENC_FRAMELOST_MODE_E;
```

[Member]

Member	Description
FRMLOST_NORMAL	When the instantaneous bit rate is above the threshold, frames are discarded.
FRMLOST_PSKIP	When the instantaneous bit rate is above the threshold, Pskip frames are encoded.

[Note]

None

[See Also]

- [HI_MPI_VENC_SetFrameLostStrategy](#)
- [HI_MPI_VENC_GetFrameLostStrategy](#)

VENC_PARAM_FRAMELOST_S

[Description]

Defines the frame discarding policy when the instantaneous bit rate of a VENC channel is above the threshold.

[Syntax]

```
typedef struct hiVENC_PARAM_FRAMELOST_S
{
    HI_BOOL bFrmLostOpen;
    HI_U32 u32FrmLostBpsThr;
    VENC_FRAMELOST_MODE_E enFrmLostMode;
    HI_U32 u32EncFrmGaps;
} VENC_PARAM_FRAMELOST_S;
```

[Member]



Member	Description
bFrmLostOpen	Frame discarding enable when the instantaneous bit rate exceeds the threshold
u32FrmLostBpsThr	Frame discarding threshold (in bit/s) Value range: [64k, 163840k]
enFrmLostMode	Frame discarding mode when the instantaneous bit rate is above the threshold
u32EncFrmGaps	Interval of discarding frames. The default value is 0. Value range: [0,65535]

[Note]

None

[See Also]

[HI_MPI_VENC_SetFrameLostStrategy](#)

VENC_RC_PRIORITY_E

[Description]

Defines the RC priority.

[Syntax]

```
typedef enum hiVENC_RC_PRIORITY_E
{
    VENC_RC_PRIORITY_BITRATE_FIRST = 1,
    VENC_RC_PRIORITY_FRAMEBITS_FIRST,
    VENC_RC_PRIORITY_BUTT,
} VENC_RC_PRIORITY_E;
```

[Member]

Member	Description
VENC_RC_PRIORITY_BITRATE_FIRST	The target bit rate takes priority.
VENC_RC_PRIORITY_FRAMEBITS_FIRST	The jumbo frame threshold takes priority.

[Note]

None

[See Also]

None



VENC_SUPERFRAME_CFG_S

[Description]

Defines the parameters for jumbo frame processing modes.

[Syntax]

```
typedef struct hiVENC_SUPERFRAME_CFG_S
{
    VENC_SUPERFRM_MODE_E enSuperFrmMode;
    HI_U32 u32SuperIFrmBitsThr;
    HI_U32 u32SuperPfrmBitsThr;
    HI_U32 u32SuperBfrmBitsThr;
    VENC_RC_PRIORITY_E enRcPriority;
} VENC_SUPERFRAME_CFG_S;
```

[Member]

Member	Description
enSuperFrmMode	Jumbo frame processing mode, The default value is SUPERFRM_NONE.
u32SuperIFrmBitsThr	Jumbo I frame threshold. The default value is 500000. Value range: greater than or equal to 0
u32SuperPfrmBitsThr	Jumbo P frame threshold. The default value is 500000. Value range: greater than or equal to 0
u32SuperBfrmBitsThr	Jumbo B frame threshold. The default value is 500000. Value range: greater than or equal to 0
enRcPriority	Bit rate control priority. The default value is VENC_RC_PRIORITY_BITRATE_FIRST.

[Note]

None

[See Also]

[HI_MPI_VENC_SetSuperFrameCfg](#)

VENC_PARAM_INTRA_REFRESH_S

[Description]

Defines the parameter for controlling I macroblock/Islice refresh in the P frame.

[Syntax]

```
typedef struct hiVENC_PARAM_INTRA_REFRESH_S
{
```



```
    HI_BOOL bRefreshEnable;
    HI_BOOL bISliceEnable;
    HI_U32 u32RefreshLineNum;
    HI_U32 u32ReqIQp;
}VENC_PARAM_INTRA_REFRESH_S;
```

[Member]

Member	Description
bRefreshEnable	I macroblock refresh enable 0 (default): disabled 1: enabled
bISliceEnable	Enable for converting the first refreshed slice into the Islice, valid only when bRefreshEnable is enabled 0 (default): disabled 1: enabled
u32RefreshLineNum	Number of rows to be refreshed during each I macroblock refresh. This member can be used to control the refresh speed and stream smooth degree. A larger value indicates faster refresh and lower stream smooth degree. A smaller value indicates slower refresh and higher stream smooth degree.
u32ReqIQp	QP value of the I frame. In intra-frame refresh mode, IDR frames may be required. The QP value of the I frame can be configured to control the quality of the inserted IDR frame. The higher the quality of the IDR frame, the larger the frame size; the lower the quality of the IDR frame, the smaller the frame size. Value range: [0, 51]

[Note]

None

[See Also]

- [HI_MPI_VENC_GetIntraRefresh](#)
- [HI_MPI_VENC_SetIntraRefresh](#)

VENC_PARAM_MOD_S

[Description]

Defines the module parameters related to encoding.

[Syntax]

```
typedef struct hiVENC_MODPARAM_S
{
    VENC_MODTYPE_E enVencModType;
    union
```



```
{  
    VENC_PARAM_MOD_VENC_S stVencModParam;  
    VENC_PARAM_MOD_H264E_S stH264eModParam;  
    VENC_PARAM_MOD_H265E_S stH265eModParam;  
    VENC_PARAM_MOD_JPEG_E_S stJpegeModParam;  
};  
} VENC_PARAM_MOD_S;
```

[Member]

Member	Description
enVencModType	Type of the module parameters that are set or obtained
stVencModParam/ stH264eModParam/ stH265eModParam/ stJpegeModParam	Structure of the module parameter for hi35xx_venc.ko , hi35xx_h264e.ko , hi35xx_h265.ko , or hi35xx_jpege.ko

[Note]

None

[See Also]

None

VENC_MODTYPE_E

[Description]

Defines the type of the module parameters related to encoding.

[Syntax]

```
typedef enum hiVENC_MODTYPE_E  
{  
    MODTYPE_VENC = 1,  
    MODTYPE_H264E,  
    MODTYPE_H265E,  
    MODTYPE_JPEG_E,  
    MODTYPE_BUTT  
} VENC_MODTYPE_E;
```

[Member]

Member	Description
MODTYPE_VENC	Type of the module parameter for hi35xx_venc.ko



Member	Description
MODTYPE_H264E	Type of the module parameter for hi35xx_h264e.ko
MODTYPE_H265E	Type of the module parameter for hi35xx_h265e.ko
MODTYPE_JPEGE	Type of the module parameter for hi35xx_jpege.ko

[Note]

None

[See Also]

[VENC_PARAM_MOD_S](#)

VENC_PARAM_MOD_VENC_S

[Description]

Defines the module parameter for **hi35xx_venc.ko**.

[Syntax]

```
typedef struct hiVENC_PARAM_MOD_VENC_S
{
    HI_U32 u32VencBufferCache;
    HI_U32 u32FrameBufRecycle;
} VENC_PARAM_MOD_VENC_S;
```

[Member]

Member	Description
u32VencBufferCache	Module parameter VencBufferCache for hi35xx_venc.ko 0: The stream buffer cache is disabled. 1: The stream buffer cache is enabled. Default value: 0
u32FrameBufRecycle	Module parameter FrameBufRecycle for hi35xx_venc.ko 0: The function of recycling the VBs for encoding frames is disabled. 1: The function of recycling the VBs for encoding frames is enabled. Default value: 0

[Note]

None



[See Also]

[VENC_PARAM_MOD_S](#)

VENC_PARAM_MOD_H264E_S

[Description]

Defines the module parameter for **hi35xx_h264e.ko**.

[Syntax]

```
typedef struct hiVENC_PARAM_MOD_H264E_S
{
    HI_U32    u32OneStreamBuffer;
    HI_U32    u32H264eVBSource;
    HI_U32    u32H264eRcnEqualRef;
    HI_U32    u32H264eMiniBufMode;
    HI_U32    u32H264ePowerSaveEn;
} VENC_PARAM_MOD_H264E_S;
```

[Member]

Member	Description
u32OneStreamBuffer	Module parameter H264eOneStreamBuffer for hi35xx_h264e.ko 0: multi-packet mode 1: single-packet mode Default value: 0
u32H264eVBSource	Module parameter H264eVBSource for hi35xx_h264e.ko 1: private VB 2: user VB Default value: 1 The user VB is not supported currently.
u32H264eRcnEqualRef	Module parameter H264eRcnEqualRef for hi35xx_h264e.ko 0: The reference frame does not share the space for storing the luminance data with the reconstruction frame. 1: The reference frame shares the space for storing the luminance data with the reconstruction frame. This member is not supported currently.



Member	Description
u32H264eMiniBufMode	Module parameter H264eMiniBufMode for hi35xx_h264e.ko 0: The stream buffers are allocated based on the resolution. 1: The lower limit for the stream buffer size is 32 KB. Ensure that the size of the allocated stream buffer is appropriate. Default value: 0
u32H264ePowerSaveEn	Module parameter H264ePowerSaveEn for hi35xx_h264e.ko 0: The low-power mode is disabled. 1: The low-power mode is enabled. Default value: 1

[Note]

None

[See Also]

[VENC_PARAM_MOD_S](#)

VENC_PARAM_MOD_H265E_S

[Description]

Defines the module parameter for **hi35xx_h265.ko**.

[Syntax]

```
typedef struct hiVENC_PARAM_MOD_H265E_S
{
    HI_U32 u32OneStreamBuffer;
    HI_U32 u32H265eMiniBufMode;
    HI_U32 u32H265ePowerSaveEn;
} VENC_PARAM_MOD_H265E_S;
```

[Member]

Member	Description
u32OneStreamBuffer	Module parameter H265eOneStreamBuffer for hi35xx_h265.ko 0: multi-packet mode 1: single-packet mode Default value: 0



Member	Description
u32H265eMiniBufMode	Module parameter H265eMiniBufMode for hi35xx_h265.ko 0: The stream buffers are allocated based on the resolution. 1: The lower limit for the stream buffer size is 32 KB. Ensure that the size of the allocated stream buffer is appropriate. Default value: 0
u32H265ePowerSaveEn	Module parameter H265ePowerSaveEn for hi35xx_h265e.ko 0: The low-power mode is disabled. 1: The low-power mode is enabled. Default value: 1

[Note]

None

[See Also]

[VENC_PARAM_MOD_S](#)

VENC_PARAM_MOD_JPEGE_S

[Description]

Defines the module parameter for **hi35xx_jpeg.ko**.

[Syntax]

```
typedef struct hiVENC_PARAM_MOD_JPEGE_S
{
    HI_U32 u32OneStreamBuffer;
    HI_U32 u32JpegeMiniBufMode;
} VENC_PARAM_MOD_JPEGE_S;
```

[Member]

Member	Description
u32OneStreamBuffer	Module parameter JpegeOneStreamBuffer for hi35xx_jpeg.ko . 0: multi-packet mode 1: single-packet mode Default value: 0



Member	Description
u32JpegeMiniBufMode	Module parameter JpegeMiniBufMode for hi35xx_jpeg.ko . 0: The stream buffers are allocated based on the resolution. 1: The lower limit for the stream buffer size is 32 KB. Ensure that the size of the allocated stream buffer is appropriate. Default value: 0

[Note]

None

[See Also]

[VENC_PARAM_MOD_S](#)

USER_RC_INFO_S

[Description]

Defines the data structure for user bit rate control information.

[Syntax]

```
typedef struct hiUSER_RC_INFO_S
{
    HI_U32 u32BlkStartQp;
    HI_U32 u32QpMapPhyAddr;
} USER_RC_INFO_S;
```

[Member]

Member	Description
u32BlkStartQp	QP value of the first 16 x 16 block in QpMap mode
u32QpMapPhyAddr	Physical address of the QP table in QpMap mode

[Note]

None

[See Also]

[HI_MPI_VENC_SendFrameEx](#)

USER_FRAME_INFO_S

[Description]



Defines the data structure of the pictures sent by users.

[Syntax]

```
typedef struct hiUSER_FRAME_INFO_S
{
    VIDEO_FRAME_INFO_S stUserFrame;
    USER_RC_INFO_S     stUserRcInfo;
} USER_FRAME_INFO_S;
```

[Member]

Member	Description
stUserFrame	Data structure of the pictures sent by users
stUserRcInfo	Data structure for the user bit rate control information

[Note]

None

[See Also]

[HI_MPI_VENC_SendFrameEx](#)

VENC_SSE_CFG_S

[Description]

Defines SSE region parameters.

[Syntax]

```
typedef struct hivenc_sse_cfg_s
{
    HI_U32    u32Index;
    HI_BOOL   bEnable;
    RECT_S    stRect;
} VENC_SSE_CFG_S;
```

[Member]

Member	Description
u32Index	Index of an SSE region. The supported index range is [0, 7].
bEnable	Whether to enable an SSE region
stRect	SSE region

[Note]



None

[See Also]

- [HI_MPI_VENC_SetSSERegion](#)
- [HI_MPI_VENC_GetSSERegion](#)

VENC_PARAM_ADVANCED_S

[Description]

Defines the advanced parameters of the VENC.

[Syntax]

```
typedef struct hiVENC_PARAM_ADVANCED_S
{
    HI_U32 u32PollWakeUpFrmCnt;
} VENC_PARAM_ADVANCED_S;
```

[Member]

Member	Description
u32PollWakeUpFrmCnt	When the channel obtains streams in timeout or block mode, the specified VENC frame wakes up the block interface after u32PollWakeUpFrmCnt . Value range: greater than 0 Default value: 1

[Note]

None

[See Also]

- [HI_MPI_VENC_SetVencAdvancedParam](#)
- [HI_MPI_VENC_GetVencAdvancedParam](#)

6.5 Error Codes

[Table 6-12](#) describes the error codes for the VENC APIs.

Table 6-12 Error codes for the VENC APIs

Error Code	Macro Definition	Description
0xA0088002	HI_ERR_VENC_INVALID_CHNID	The channel ID is invalid.
0xA0088003	HI_ERR_VENC_ILLEGAL_PARAM	The parameter is invalid.
0xA0088004	HI_ERR_VENC_EXIST	The device, channel or resource to be created or



Error Code	Macro Definition	Description
		applied for exists.
0xA0088005	HI_ERR_VENC_UNEXIST	The device, channel or resource to be used or destroyed does not exist.
0xA0088006	HI_ERR_VENC_NULL_PTR	The parameter pointer is null.
0xA0088007	HI_ERR_VENC_NOT_CONFIG	No parameter is set before use.
0xA0088008	HI_ERR_VENC_NOT_SUPPORT	The parameter or function is not supported.
0xA0088009	HI_ERR_VENC_NOT_PERM	The operation, for example, modifying static parameters, is forbidden.
0xA008800C	HI_ERR_VENC_NOMEM	The memory fails to be allocated due to some causes such as insufficient system memory.
0xA008800D	HI_ERR_VENC_NOBUF	The buffer fails to be allocated due to some causes such as oversize of the data buffer applied for.
0xA008800E	HI_ERR_VENC_BUF_EMPTY	The buffer is empty.
0xA008800F	HI_ERR_VENC_BUF_FULL	The buffer is full.
0xA0088010	HI_ERR_VENC_SYS_NOTREADY	The system is not initialized or the corresponding module is not loaded.
0xA0088012	HI_ERR_VENC_BUSY	The VENC system is busy.



Contents

7 VDA	7-1
7.1 Overview	7-1
7.2 Functions	7-1
7.2.1 Concepts	7-1
7.2.2 Creating VDA Channels	7-3
7.2.3 Input Sources	7-5
7.2.4 Processing VDA Results	7-6
7.3 API Reference	7-7
7.4 Data Structures	7-21
7.5 Error Codes	7-38



Figures

Figure 7-1 Memory allocation for the macroblock SAD.....7-6



Tables

Table 7-1 Error codes for VDA APIs.....	7-38
--	-------------



7 VDA

7.1 Overview

The video detection analysis (VDA) module obtains video detection results by detecting the video luminance variance. The VDA module supports the motion detection (MD) mode and occlusion detection (OD) mode. Accordingly, the video detection results are classified into MD results and OD results. The details are as follows:

- In MD mode, the VDA module checks the video motion status received by the VDA channel, and outputs MD results. The results include the sum of absolute difference (SAD) for each macroblock of each frame, object (OBJ) region information, and number of alarm pixels for the entire frame.
- In OD mode, the VDA module detects whether the video received by the VDA channel is occluded, and outputs OD results.

You can create a VDA channel, bind the VI source, and start the VDA function. Each channel can be set to an operating mode and channel attributes are set based on the operating mode. For details, see the description of [HI_MPI_VDA_CreateChn](#).

NOTE

Only the Hi3516A supports the VDA function.

7.2 Functions

7.2.1 Concepts

- VDA interval

Pictures are received from the input source at intervals to control the frame rate. The interval is adjusted based on the actual frame rate of the input source. For example, if the VI frame rate is 25 frames per second, it is recommended that five frames are received each second, and the interval is set to 4 frames. Unless otherwise specified, the frame described in this chapter is the picture that is received from the input source, and transmitted to the VDA channel.

- Macroblock

A picture is divided into blocks (in pixels), for example, 8x8 or 16x16 blocks. Each block is called a macroblock.

- Macroblock size



Only 8x8 and 16x16 macroblocks are supported currently.

- Macroblock SAD

The macroblock SAD is the sum of the absolute luminance differences between the macroblocks of two frames. The greater the macroblock SAD, the greater the luminance difference.

- Bits of the macroblock SAD

The Hi35xx supports 8-bit or 16-bit macroblock SAD. That is, the maximum bits of the detected macroblock SAD are 16 bits. In 16-bit mode, the actual macroblock SAD is output, and high bandwidth is required; in 8-bit mode, upper eight bits of the actual macroblock SAD are output, and low bandwidth is required. You need to select the 8-bit or 16-bit mode as required.

- SAD threshold

The Hi3516A processes macroblocks by 4x4 macroblocks. The SAD threshold is used for internal processing of the Hi3516A. That is, the SAD threshold is used to check whether a macroblock is a motion one. If the macroblock SAD is greater than or equal to the SAD threshold, the macroblock is considered as a motion one. Based on the motion macroblocks, the MD OBJ region information, and number of alarm pixels for the entire frame, or OD information is obtained. The preceding macroblocks are 4x4 macroblocks processed in the Hi3516A. The SAD threshold is based on 4x4 macroblocks, which is independent of the configured macroblock size.

- Reference frame mode

The reference frame mode includes static reference frame mode, dynamic reference frame mode, and user reference frame mode. The reference frame mode determines the way in which reference frames are generated during VDA processing.

The user reference frame mode is not supported currently, and the dynamic reference frame mode is recommended.

- VDA algorithm

The VDA algorithm includes the frame reference algorithm (VDA_ALG_REF) and background algorithm (VDA_ALG_BG). The VDA algorithm is specified when a channel is created, and the background algorithm is recommended.

- Frame reference algorithm

This algorithm indicates that a picture is used as the reference frame to obtain VDA results

The reference frame is obtained in either of the following ways:

- Uses a fixed picture as the reference frame in static reference frame mode.

- Uses the previous frame as the reference frame in dynamic reference frame mode

- Background algorithm

This algorithm indicates that the background picture of the current video is generated during VDA processing, and the background picture is used as the reference frame to obtain VDA results

The background picture is obtained in either of the following ways:

- Uses a fixed frame as the background picture in static reference frame mode.

- Checks the still parts of the video, and obtains the background picture through internal processing in dynamic reference frame mode.

- Background refresh weight

The configured background update weight takes effect only when the background algorithm and dynamic reference frame mode are used. During VDA processing, still



pictures are abstracted. The pixels of these pictures and background are blended. The formula is as follows:

Pixels of the new background = (Blending weight of still pictures x Pixels of still pictures + (256 – Blending weight of still pictures) x Pixels of the old background)/256

The blending weight of still pictures is the background refresh weight. The greater the weight, the fast the background is refreshed. The recommended weight is 128.

- VDA result

The VDA results are classified into MD results and OD results based on the operating mode of the channel. For details, see the description of [VDA_DATA_S](#).

7.2.2 Creating VDA Channels

You can start the VDA function by creating a VDA channel, binding the channel and the input source, and calling related MPI to receive pictures. Only one operating mode can be set for a channel. The width, height, and attributes of a channel are configured based on its operating mode. The channel in MD mode is an MD channel, and the channel in OD mode is an OD channel. This section describes the configuration items of the two types of channels, and configuration methods.

Same Configuration Items

- VDA algorithm (enVdaAlg)
The frame reference algorithm and background algorithm are supported.
- Reference frame mode (enRefMode)
The static reference frame mode and dynamic reference frame mode are supported.
- VDA interval (u32VdaIntvl)
You need to adjust the VDA interval to ensure optimum VDA performance. The maximum VDA performance is D1@ 15 fps or CIF@ 30 fps.
- Macroblock size (enMbSize)
The macroblock sizes of 8x8 and 16x16 are supported.
- SAD output bits (enMbSadBits)
The 8-bit and 16-bit modes are supported.
- Background refresh weight (u32BgUpSrcWgt)
This item takes effect only when the background algorithm is used.

Configuration Items for the MD Channel

- Number of MD result buffers (u32MdBufNum)
The MD channel outputs MD results such as the macroblock SADs of each frame. The results of each frame occupy a large memory, and a result queue is generated. You can specify the depth of the result queue, that is, the number of MD results buffers.
- SAD threshold (u32SadTh)
The MD channel processes a frame, and output the MD results of the frame. All motion-related information is determined based on the SAD threshold.
- Maximum number of output OBJ regions (u32ObjNumMax)
The Hi3516A checks whether a macroblock is moved based on its macroblock SAD and the user-defined SAD threshold. By using the edge detecting algorithm, OBJ regions are detected, and the coordinates of the upper left and lower right corners of each OBJ region are output. The number of detected OBJ regions varies according to frames. You



can set u32ObjNumMax to control the maximum number of output OBJ regions. The recommended value is 128.

Configuration Items for the OD Channel

- Number of OD regions (u32RgnNum)

The OD allows you to divide a frame into multiple regions for occlusion detection. A maximum of four regions is supported for each frame. You can set the number of OD regions and set the attributes (astOdRgnAttr) of each region. The region array index ranges from 0 to (u32RgnNum – 1). The OD channel processes only one region of a frame each time.

- Region range (stRect)

The region range specifies the position and size of a region in a frame. Regions can be occluded by each other. For details about restrictions, see the description of [VDA_OD_RGN_ATTR_S](#).

- Region SAD threshold (u32SadTh)

The functions of the region SAD threshold and MD SAD threshold are the same. The difference is that the region SAD threshold is valid only for the current region. You must set same or different SAD thresholds for each region.

- Region area alarm threshold (u32AreaTh)

Based on the region SAD threshold, the VDA module detects the maximum OBJ region after processing a frame. When the percentage of the OBJ region in the current region area is greater than or equal to the region area alarm threshold, the current region is occluded. The region area alarm threshold is a percentage relative to the region area, and ranges from 0 to 100. The recommended value is 70%.

- Region occlusion count alarm threshold (u32OccCntTh)

No alarm information is output when the VDA module detects that a region is occluded once. That is, the VDA module outputs the occlusion alarm information only after detecting that this region is occluded for several times. The number of occlusion times is the region occlusion count alarm threshold. When the number of occlusion times is greater than or equal to the threshold, alarm information is output, and the VDA module stops detecting the region.

- Uncover count (u32UnOccCntTh)

The uncover count is called the error value. The region occlusion count stops until the uncover count reaches the configured error value, regardless of whether the VDA module detects that the region is not occluded. If the occlusion count is much greater than the uncover count, the VDA module considers that the region is occluded, and notifies you of the alarm information.

Configuration Methods

- Configuring the SAD threshold

When the picture noise is large, the SAD threshold is large accordingly.

If you configure the SAD threshold for the first time, do as follows:

- Create an MD channel, set the macroblock size to 8x8, set the output bit to 16 bits, and output the macroblock SAD.
- Perform occlusion operations, choose an intermediate value from the output SADs, and divide the intermediate value by 4 to obtain an approximate SAD threshold.



- Create an OD channel, and adjust the approximate SAD threshold to obtain the optimum SAD threshold. You are advised to respectively increase and decrease the approximate SAD threshold by 16 for five times.
- Configuring the VDA interval

The following is an example using the application scenario of VI D1@ 30 fps, one MD channel, and one OD channel (one region).
In this case, it is recommended that you set both the MD interval and OD interval to 5.
The performance of the MD channel or OD channel is calculated as follows:
 $30/(5 + 1)=5$ fps
Therefore, the total performance is 10 fps, which is allowed by the VDA performance.
- Configuring the number of MD result buffers (u32MdBufNum)

When the number of MD result buffers is large, the occupied memory is high. This indicates that the VDA threshold is also high.
When the number of MD result buffers is small, the occupied memory is low, and the frequency of obtaining results is high. Otherwise, VDA processing is blocked.
The recommended value is 8.
- Configuring the macroblock size (enMbSize) and output bits (enMbSadBits)

If the macroblock size set to 8x8, the SAD precision and required bandwidth is high; if the macroblock size is set to 16x16, the SAD precision and required bandwidth are low.
In 8-bit output mode, the SAD precision and required bandwidth are low; in 16-bit output mode, the SAD precision and bandwidth are high.
You need to balance the SAD precision against the bandwidth as required.
- Configuring u32OccCntTh and u32UnOccCntTh

The u32OccCntTh, input source frame rate, and VDA interval determine the region occlusion duration for outputting the occlusion alarm information.
In general, VI frame rate is set to 30 fps, VDA interval is set to 5, and u32OccCntTh is set to 20 (u32UnOccCntTh is ignored).
In this case, the region occlusion duration is calculated as follows:
 $20/(30/(5 + 1)) = 4$ s
This indicates that when the lens is occluded by an object, the occlusion alarm information is output about four seconds later. You can adjust the value of u32OccCntTh based on the VI frame rate, and VDA interval to obtain the expected region occlusion duration.
u32UnOccCntTh is an auxiliary parameter. You are advised to set the value of u32UnOccCntTh to 1/10 of the value of u32OccCntTh or smaller. In the preceding example, u32OccCntTh is set to 20. You can set the value of u32UnOccCntTh to 2, 1, or 0.

7.2.3 Input Sources

You can select the following bound input sources:

- VI output
- Decoding output
- Bypass channel output of the VPSS
- Other VPSS channel output in user mode



You can send pictures to a VDA channel by calling [HI_MPI_VDA_SendPic](#) without binding.

Note that the Hi3516A does not support compressed input pictures.

7.2.4 Processing VDA Results

After you create a VDA channel, bind the channel and input source, and enable the channel to receive pictures, the related hardware is started for VDA processing. This section describes the results generated in the MD channel and OD channels.

MD Channel

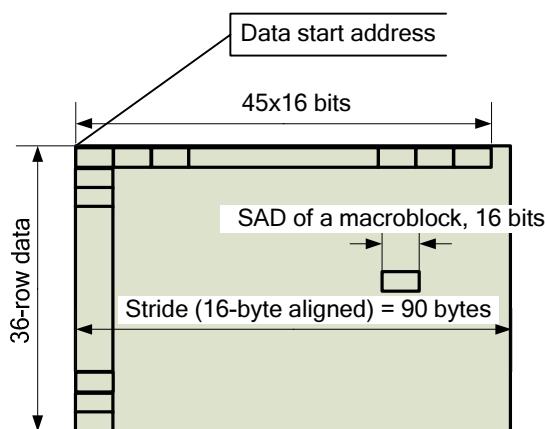
- MD results

For details, see the description of [VDA_MD_DATA_S](#).

- Macroblock SAD

The SAD thresholds of all macroblocks in the entire frame are output. For details, see the description of [VDA_MB_SAD_DATA_S](#). The SAD threshold consists of three variables: data start address, stride, and output bits. Assume that the macroblock size is 16x16, output bits are 16 bits, and the MD channel size is 720x576. The width and height of the MD channel is 45x36 in macroblocks. That is, there are 45x36 macroblocks. [Figure 7-1](#) shows the memory allocation for the macroblock SAD. For details about related read and write operations, see related samples.

Figure 7-1 Memory allocation for the macroblock SAD



- OBJ region information

The OBJ region is a rectangular motion part of the current frame that is detected by the hardware based on the SAD threshold. For details, see the description of [VDA_OBJ_DATA_S](#). For details about related read and write operations, see related samples.

- Number of the alarm pixels in an entire frame

It is the number of motion pixels in an entire frame. During the spray test, the Hi3516A detects many but small OBJ regions. In this case, the number of the alarm pixels in an entire frame serves as an auxiliary parameter.



OD Channel

Only one alarm is provided for the OD channel. This alarm is used to check whether the specified region is occluded.

- Region alarm information

For details, see the description of [VDA_OD_DATA_S](#). The region alarm information includes the number of regions, and Boolean array for identifying whether the regions are occluded. The value HI_TRUE indicates that a region is occluded and the value HI_FALSE indicates that a region is not occluded. For example, if you set three regions A, B, and C, and region B is occluded, the following message is displayed:

u32RgnNum = 3;abRgnAlarm[0] = 0, abRgnAlarm[1] = 1, abRgnAlarm[2] = 0,
abRgnAlarm[3] = invalid

7.3 API Reference

The VDA provides the following MPIs:

- [HI_MPI_VDA_CreateChn](#): Creates a VDA channel.
- [HI_MPI_VDA_DestroyChn](#): Destroys a VDA channel.
- [HI_MPI_VDA_SetChnAttr](#): Sets the attributes of a VDA channel.
- [HI_MPI_VDA_GetChnAttr](#): Obtains the attributes of a VDA channel.
- [HI_MPI_VDA_StartRecvPic](#): Starts to receive pictures.
- [HI_MPI_VDA_StopRecvPic](#): Stops receiving pictures.
- [HI_MPI_VDA_GetData](#): Obtains the VDA results
- [HI_MPI_VDA_ReleaseData](#): Release the buffer for storing the VDA results
- [HI_MPI_VDA_ResetOdRegion](#): Performs the region occlusion detection again.
- [HI_MPI_VDA_Query](#): Queries the status of a VDA channel.
- [HI_MPI_VDA_UpdateRef](#): Updates the reference picture.
- [HI_MPI_VDA_GetFd](#): Obtains the device file descriptor (FD) corresponding to a VDA channel.
- [HI_MPI_VDA_SendPic](#): Sends pictures to a VDA channel.

HI_MPI_VDA_CreateChn

[Description]

Creates a VDA channel.

[Syntax]

```
HI_S32 HI_MPI_VDA_CreateChn(VDA_CHN VdaChn, const VDA_CHN_ATTR_S  
*pstAttr);
```

[Parameter]



Parameter	Description	Input/Output
VdaChn	VDA channel ID Value range: [0, VDA_CHN_NUM_MAX)	Input
pstAttr	VDA channel attributes	Input

[Return Value]

Return Value	Description
0	Success.
Other values	For details, see section 7.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vda.h, mpi_vda.h
- Library file: libmpi.a

[Note]

- Only one operating mode can be set for a VDA channel.
- A maximum of **VDA_CHN_NUM_MAX** channels can be created.
- If the VDA algorithm is set to frame reference algorithm, the background weight must also be set for checking parameters.
- The VDA function is enabled only after the input source is bound and [**HI_MPI_VDA_StartRecvPic**](#) is called.

[Example]

```
HI_S32 s32Ret = HI_SUCCESS;
    VDA_CHN VdaChn;
    VDA_CHN_ATTR_S stVdaChnAttr;
    MPP_CHN_S stSrcChn;
    MPP_CHN_S stDestChn;

    VdaChn = 0;

    stVdaChnAttr.enWorkMode = VDA_WORK_MODE_MD;
    stVdaChnAttr.u32Width = 720;
    stVdaChnAttr.u32Height = 576;
    stVdaChnAttr.unAttr.stMdAttr.enVdaAlg = VDA_ALG_BG;
    stVdaChnAttr.unAttr.stMdAttr.enMbSize = VDA_MB_16PIXEL;
    stVdaChnAttr.unAttr.stMdAttr.enMbSadBits = VDA_MB_SAD_8BIT;
    stVdaChnAttr.unAttr.stMdAttr.enRefMode = VDA_REF_MODE_DYNAMIC;
    stVdaChnAttr.unAttr.stMdAttr.u32VdaIntvl = 12;
    stVdaChnAttr.unAttr.stMdAttr.u32BgUpSrcWgt = 128;
```



```
    stVdaChnAttr.unAttr.stMdAttr.u32MdBufNum = 8;
    stVdaChnAttr.unAttr.stMdAttr.u32ObjNumMax = 128;
    stVdaChnAttr.unAttr.stMdAttr.u32SadTh = 40;
    /*Create a VDA channel.*/
    s32Ret = HI_MPI_VDA_CreateChn(VdaChn, &stVdaChnAttr);
    if(s32Ret != HI_SUCCESS)
    {
        return s32Ret;
    }

    /*Bind the VDA channel and VI source.*/
    stSrcChn.enModId = HI_ID_VIU;
    stSrcChn.s32ChnId = 0;
    stDestChn.enModId = HI_ID_VDA;
    stDestChn.s32ChnId = VdaChn;
    s32Ret = HI_MPI_SYS_Bind(&stSrcChn, &stDestChn);
    if(s32Ret != HI_SUCCESS)
    {
        return s32Ret;
    }
    /*Enable the channel to receive pictures.*/
    s32Ret = HI_MPI_VDA_StartRecvPic(VdaChn);
    if(s32Ret != HI_SUCCESS)
    {
        return s32Ret;
    }

    /*Stop receiving pictures.*/
    s32Ret = HI_MPI_VDA_StopRecvPic(VdaChn);
    if(s32Ret != HI_SUCCESS)
    {
        return s32Ret;
    }
    /*Unbind the VDA channel and VI source.*/
    s32Ret = HI_MPI_SYS_UnBind(&stSrcChn, &stDestChn);
    if(s32Ret != HI_SUCCESS)
    {
        return s32Ret;
    }
    s32Ret = HI_MPI_VDA_DestroyChn(VdaChn);
    if(s32Ret != HI_SUCCESS)
    {
        return s32Ret;
    }
```



}

[See Also]

[HI_MPI_VDA_DestroyChn](#)

HI_MPI_VDA_DestroyChn

[Description]

Destroys a VDA channel.

[Syntax]

```
HI_S32 HI_MPI_VDA_DestroyChn(VDA_CHN VdaChn);
```

[Parameter]

Parameter	Description	Input/Output
VdaChn	VDA channel ID Value range: [0, VDA_CHN_NUM_MAX)	Input

[Return Value]

Return Value	Description
0	Success.
Other values	For details, see section 7.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vda.h, mpi_vda.h
- Library file: libmpi.a

[Note]

- Before calling this MPI, you must create a VDA channel.
- Before calling this MPI, you must call [HI_MPI_VDA_StopRecvPic](#) to stop receiving pictures.

[Example]

For details, see the sample of [HI_MPI_VDA_CreateChn](#).

[See Also]

None

HI_MPI_VDA_SetChnAttr

[Description]

Sets the attributes of a VDA channel.



[Syntax]

```
HI_S32 HI_MPI_VDA_SetChnAttr(VDA_CHN VdaChn, const VDA_CHN_ATTR_S  
*pstAttr);
```

[Parameter]

Parameter	Description	Input/Output
VdaChn	VDA channel ID Value range: [0, VDA_CHN_NUM_MAX)	Input
pstAttr	VDA channel attributes	Input

[Return Value]

Return Value	Description
0	Success.
Other values	For details, see section 7.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vda.h, mpi_vda.h
- Library file: libmpi.a

[Note]

- Only dynamic attributes can be modified. For details about restrictions, see the description of [VDA_CHN_ATTR_S](#).
- Before calling this MPI, you must create a VDA channel.
- You are advised to obtain the attributes of a VDA channel before setting attributes.
- The pointer to input channel attributes cannot be null.

[Example]

```
HI_S32 s32Ret = HI_SUCCESS;  
VDA_CHN VdaChn;  
VDA_CHN_ATTR_S stVdaChnAttrIn;  
VdaChn = 0;  
s32Ret = HI_MPI_VDA_GetChnAttr(VdaChn, &stVdaChnAttrIn);  
if(s32Ret != HI_SUCCESS)  
{  
    return s32Ret;  
}  
  
/*Change the VDA interval.*/  
stVdaChnAttrIn.unAttr.stMdAttr.u32VdaIntvl = 7;  
s32Ret = HI_MPI_VDA_SetChnAttr(VdaChn, &stVdaChnAttrIn);
```



```
if(s32Ret != HI_SUCCESS)
{
    return s32Ret;
}
```

[See Also]

[HI_MPI_VDA_SetChnAttr](#)

HI_MPI_VDA_SetChnAttr

[Description]

Obtains the attributes of a VDA channel.

[Syntax]

```
HI_S32 HI_MPI_VDA_SetChnAttr(VDA_CHN VdaChn, VDA_CHN_ATTR_S *pstAttr);
```

[Parameter]

Parameter	Description	Input/Output
VdaChn	VDA channel ID Value range: [0, VDA_CHN_NUM_MAX)	Input
pstAttr	VDA channel attributes	Output

[Return Value]

Return Value	Description
0	Success.
Other values	For details, see section 7.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vda.h, mpi_vda.h
- Library file: libmpia.a

[Note]

- Before calling this MPI, you must create a VDA channel.
- The pointer to output channel attributes cannot be null.

[Example]

For details, see the sample of [HI_MPI_VDA_SetChnAttr](#).

[See Also]

None



HI_MPI_VDA_StartRecvPic

[Description]

Starts to receive pictures.

[Syntax]

```
HI_S32 HI_MPI_VDA_StartRecvPic(VDA_CHN VdaChn);
```

[Parameter]

Parameter	Description	Input/Output
VdaChn	VDA channel ID Value range: [0, VDA_CHN_NUM_MAX)	Input

[Return Value]

Return Value	Description
0	Success.
Other values	For details, see section 7.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vda.h, mpi_vda.h
- Library file: libmpi.a

[Note]

- Before calling this MPI, you must create a VDA channel.
- You can call this MPI regardless of whether the input source is bound.
- This MPI can be called repeatedly.

[Example]

For details, see the sample of [HI_MPI_VDA_CreateChn](#).

[See Also]

None

HI_MPI_VDA_StopRecvPic

[Description]

Stops receiving pictures.

[Syntax]

```
HI_S32 HI_MPI_VDA_StopRecvPic(VDA_CHN VdaChn);
```

[Parameter]



Parameter	Description	Input/Output
VdaChn	VDA channel ID Value range: [0, VDA_CHN_NUM_MAX)	Input

[Return Value]

Return Value	Description
0	Success.
Other values	For details, see section 7.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vda.h, mpi_vda.h
- Library file: libmpi.a

[Note]

- Before calling this MPI, you must create a VDA channel.
- You can call this MPI regardless of whether the input source is unbound.
- This MPI can be called repeatedly.

[Example]

For details, see the sample of [HI_MPI_VDA_CreateChn](#).

[See Also]

None

HI_MPI_VDA_GetData

[Description]

Obtains the VDA results. If the VDA channel works in MD mode, MD results are obtained; if the VDA channel works in OD mode, OD results are obtained.

[Syntax]

```
HI_S32 HI_MPI_VDA_GetData(VDA_CHN VdaChn, VDA_DATA_S *pstVdaData, HI_S32 s32MilliSec);
```

[Parameter]

Parameter	Description	Input/Output
VdaChn	VDA channel ID Value range: [0, VDA_CHN_NUM_MAX)	Input
pstVdaData	VDA results	Output
s32MilliSec	Wait timeout period (in ms)	Input



[Return Value]

Return Value	Description
0	Success.
Other values	For details, see section 7.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vda.h, mpi_vda.h
- Library file: libmpi.a

[Note]

- Before calling this MPI, you must create a VDA channel.
- The VDA results can be obtained in block mode.
- The pointer to output results cannot be null.
- When **s32MilliSec** is **0**, data is obtained in non-block mode.
- When **s32MilliSec** is **-1**, data is obtained in block mode.
- When **s32MilliSec** is greater than 0, a code indicating success is returned if data is received within **s32MilliSec** ms. If the wait time is longer than **s32MilliSec** ms, a timeout occurs and an error is returned.
- When the MD channel works in block mode and the MD result buffer is empty, you must wait until the MD processing is complete.
- When the MD channel works in non-block mode and the MD result buffer is empty, an error is returned.
- When VDA results are obtained from the MD channel, the time stamp in the results cannot be changed.
- VDA results can be obtained repeatedly from the OD channel in block mode or non-block mode.

[Example]

For details, see the related sample.

[See Also]

None

HI_MPI_VDA_ReleaseData

[Description]

Release the buffer for storing the VDA results

[Syntax]

```
HI_S32 HI_MPI_VDA_ReleaseData(VDA_CHN VdaChn, const VDA_DATA_S*  
pstVdaData);
```

[Parameter]



Parameter	Description	Input/Output
VdaChn	VDA channel ID Value range: [0, VDA_CHN_NUM_MAX)	Input
pstVdaData	VDA results	Input

[Return Value]

Return Value	Description
0	Success.
Other values	For details, see section 7.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vda.h, mpi_vda.h
- Library file: libmpi.a

[Note]

- Before calling this MPI, you must create a VDA channel.
- The pointer to input results cannot be null.
- For the MD channel, you must release the VDA result buffer after VDA results are used. If the buffer is not released in time, the buffer is fully occupied, blocking the MD channel.
- Each time VDA results are obtained from the MD channel, a release operation must be performed.
- Each time VDA results are obtained from the OD channel, you can determine whether to perform a release operation.
- It is recommended that you do not destroy the channel when no code is returned after this MPI is called. Otherwise, the channel is deleted successfully, but an error code is returned, indicating that the channel does not exist.

[Example]

For details, see the related sample.

[See Also]

[HI_MPI_VDA_GetData](#)

HI_MPI_VDA_ResetOdRegion

[Description]

Performs the region occlusion detection again. This MPI is valid only for the OD channel. Assume that four regions are set for an OD channel: A, B, C, and D. If region A is occluded, the VDA module stops checking region A, and outputs the region occlusion alarm information. The regions B, C, and D are still being checked.



After receiving the region occlusion alarm information, you can enable region A to work by calling the MPI and entering the channel ID and the array index 0 of region A. Then region A continues to be checked, which has no effect on other regions.

[Syntax]

```
HI_S32 HI_MPI_VDA_ResetOdRegion(VDA_CHN VdaChn, HI_S32 s32RgnIndex);
```

[Parameter]

Parameter	Description	Input/Output
VdaChn	VDA channel ID Value range: [0, VDA_CHN_NUM_MAX)	Input
s32RgnIndex	Region array index Value range: [0, VDA_OD_RGN_NUM_MAX)	Input

[Return Value]

Return Value	Description
0	Success.
Other values	For details, see section 7.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vda.h, mpi_vda.h
- Library file: libmpapi.a

[Note]

- Before calling this MPI, you must create a VDA channel.
- This MPI can be called repeatedly.
- After receiving the region occlusion alarm information, you need to call this MPI to enable the region to work.

[Example]

For details, see the related sample.

[See Also]

None

HI_MPI_VDA_Query

[Description]

Queries the status of a VDA channel. The status information includes the number of pictures in the channel picture queue, picture receiving status, and number of results in the result buffer.

[Syntax]



```
HI_S32 HI_MPI_VDA_Query(VDA_CHN VdaChn, VDA_CHN_STAT_S *pstChnStat);
```

[Parameter]

Parameter	Description	Input/Output
VdaChn	VDA channel ID Value range: [0, VDA_CHN_NUM_MAX)	Input
pstChnStat	Channel status	Output

[Return Value]

Return Value	Description
0	Success.
Other values	For details, see section 7.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vda.h, mpi_vda.h
- Library file: libmpi.a

[Note]

- Before calling this MPI, you must create a VDA channel.
- The pointer to output channel status cannot be null.

[Example]

For details, see the related sample.

[See Also]

None

HI_MPI_VDA_UpdateRef

[Description]

Updates the reference frame. This MPI is valid only when the user reference frame mode is selected.

[Syntax]

```
HI_S32 HI_MPI_VDA_UpdateRef(VDA_CHN VdaChn, const VIDEO_FRAME_INFO_S  
*pstRefFrame);
```

[Parameter]



Parameter	Description	Input/Output
VdaChn	VDA channel ID Value range: [0, VDA_CHN_NUM_MAX)	Input
pstRefFrame	Reference picture information	Input

[Return Value]

Return Value	Description
0	Success.
Other values	For details, see section 7.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vda.h, mpi_vda.h
- Library file: libmpi.a

[Note]

In user reference frame mode, you must call this MPI to set reference frames first. Otherwise, an error occurs when [HI_MPI_VDA_StartRecvPic\(\)](#) is called, and the VDA cannot receive pictures properly.

[Example]

None

[See Also]

None

HI_MPI_VDA_GetFd

[Description]

Obtains the device FD corresponding to a VDA channel.

[Syntax]

```
HI_S32 HI_MPI_VDA_GetFd(VDA_CHN VdaChn);
```

[Parameter]

Parameter	Description	Input/Output
VdaChn	VDA channel ID Value range: [0, VDA_CHN_NUM_MAX)	Input

[Return Value]



Return Value	Description
Positive value	Valid return value
Non-positive value	Invalid return value

[Error Code]

For details, see section [7.5 "Error Codes."](#)

[Requirement]

- Header files: hi_comm_vda.h, mpi_vda.h
- Library file: libmpi.a

[Note]

None

[Example]

None

[See Also]

None

HI_MPI_VDA_SendPic

[Description]

Sends pictures to a VDA channel.

[Syntax]

```
HI_S32 HI_MPI_VDA_SendPic(VDA_CHN VdaChn, const VIDEO_FRAME_INFO_S
    *pstUserFrame, HI_S32 s32MilliSec)
```

[Parameter]

Parameter	Description	Input/Output
VdaChn	VDA channel ID Value range: [0, VDA_CHN_NUM_MAX)	Input
pstUserFrame	Pointer to the information about the pictures to be sent	Input
u32MilliSec	Wait timeout period, in ms	Input

[Return Value]

Return Value	Description
0	Success.



Return Value	Description
Other values	Timeout or an error occurs. For details, see section 7.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vda.h, mpi_vda.h
- Library file: libmpi.a

[Note]

- Before calling this MPI, you must create a VDA channel.
- The channel must be receiving data.
- The **pstUserFrame** pointer cannot be null.
- When **s32MilliSec** is **0**, pictures are sent by calling this MPI in non-block mode. If pictures fail to be sent, an error is returned. When **s32MilliSec** is **-1**, the MPI is blocked until the VDA receives pictures. When **s32MilliSec** is greater than **0**, a code indicating success is returned when pictures are received in **s32MilliSec** ms. If the wait time is longer than **s32MilliSec** ms, timeout occurs and an error is returned.
- As the timeout period is in the unit of 10 ms, the maximum wait time must be a multiple of 10 ms.

[Example]

None

[See Also]

None

7.4 Data Structures

The VDA data structures are as follows:

- **VDA_CHN**: Defines a VDA channel.
- **VDA_CHN_NUM_MAX**: Defines the maximum number of VDA channels.
- **VDA_MAX_WIDTH**: Defines the maximum width of a VDA channel.
- **VDA_MAX_HEIGHT**: Defines the maximum height of a VDA channel.
- **VDA_MIN_WIDTH**: Defines the minimum width of a VDA channel.
- **VDA_MIN_HEIGHT**: Defines the minimum height of a VDA channel.
- **VDA_OD_RGN_NUM_MAX**: Defines the maximum number of OD regions.
- **VDA_OBJ_S**: Defines the OBJ region information.
- **VDA_REF_MODE_E**: Defines the reference frame mode.
- **VDA_ALG_E**: Defines the VDA algorithm.
- **VDA_MB_SIZE_E**: Defines the macroblock size.
- **VDA_MB_SADBITS_E**: Defines the precision of the MD SAD.
- **VDA_OD_RGN_ATTR_S**: Defines the attributes of an occlusion region.



- [VDA_MD_ATTR_S](#): Defines the MD attributes.
- [VDA_OD_ATTR_S](#): Defines the OD attributes.
- [VDA_WORK_MODE_E](#): Defines the VDA operating mode.
- [VDA_WORK_MODE_ATTR_U](#): Defines the operating mode attributes.
- [VDA_CHN_ATTR_S](#): Defines the channel attributes.
- [VDA_MB_SAD_DATA_S](#): Defines the SAD information about a macroblock.
- [VDA_OBJ_DATA_S](#): Defines the OBJ region results.
- [VDA_MD_DATA_S](#): Defines the MD results.
- [VDA_OD_DATA_S](#): Defines the OD results.
- [VDA_DATA_U](#): Defines the VDA result union.
- [VDA_DATA_S](#): Defines the VDA results
- [VDA_CHN_STAT_S](#): Defines the status of a VDA channel.

VDA_CHN

[Description]

Defines a VDA channel.

[Syntax]

```
typedef HI_S32 VDA_CHN;
```

[Member]

None

[Note]

None

[See Also]

None

VDA_CHN_NUM_MAX

[Description]

Defines the maximum number of VDA channels.

[Syntax]

```
#define VDA_CHN_NUM_MAX 32
```

[Member]

None

[Note]

None

[See Also]

None



VDA_MAX_WIDTH

[Description]

Defines the maximum width of a VDA channel.

[Syntax]

```
#define VDA_MAX_WIDTH 960
```

[Member]

None

[Note]

None

[See Also]

None

VDA_MAX_HEIGHT

[Description]

Defines the maximum height of a VDA channel.

[Syntax]

```
#define VDA_MAX_HEIGHT 960
```

[Member]

None

[Note]

None

[See Also]

None

VDA_MIN_WIDTH

[Description]

Defines the minimum width of a VDA channel.

[Syntax]

```
#define VDA_MIN_WIDTH 32
```

[Member]

None

[Note]

None

[See Also]



None

VDA_MIN_HEIGHT

[Description]

Defines the minimum height of a VDA channel.

[Syntax]

```
#define VDA_MIN_HEIGHT 32
```

[Member]

None

[Note]

None

[See Also]

None

VDA_OD_RGN_NUM_MAX

[Description]

Defines the maximum number of OD regions.

[Syntax]

```
#define VDA_OD_RGN_NUM_MAX 4
```

[Member]

None

[Note]

None

[See Also]

None

VDA_OBJ_S

[Description]

Defines the OBJ region information.

[Syntax]

```
typedef struct hiVDA_OBJ_S
{
    HI_U16 u16Left;
    HI_U16 u16Top;
    HI_U16 u16Right;
    HI_U16 u16Bottom;
```



```
}VDA_OBJ_S;
```

[Member]

Member	Description
u16Left	Horizontal coordinate of the upper left corner of an OBJ region
u16Top	Vertical coordinate of the upper left corner of an OBJ region
u16Right	Horizontal coordinate of the lower right corner of an OBJ region
u16Bottom	Vertical coordinate of the lower right corner of an OBJ region

[Note]

None

[See Also]

None

VDA_REF_MODE_E

[Description]

Defines the reference frame mode.

[Syntax]

```
typedef enum hiVDA_REF_MODE_E
{
    VDA_REF_MODE_DYNAMIC = 0,
    VDA_REF_MODE_STATIC,
    VDA_REF_MODE_USER,
    VDA_REF_MODE_BUTT
}VDA_REF_MODE_E;
```

[Member]

Member	Description
VDA_REF_MODE_DYNAMIC	Dynamic reference frame mode
VDA_REF_MODE_STATIC	Static reference frame mode
VDA_REF_MODE_USER	User reference frame mode (not supported)

[Note]

None

[See Also]

None



VDA_ALG_E

[Description]

Defines the VDA algorithm.

[Syntax]

```
typedef enum hiVDA_ALG_E
{
    VDA_ALG_BG = 0,
    VDA_ALG_REF,
    VDA_ALG_BUTT
} VDA_ALG_E;
```

[Member]

Member	Description
VDA_ALG_BG	Background algorithm
VDA_ALG_REF	Frame reference algorithm

[Note]

None

[See Also]

None

VDA_MB_SIZE_E

[Description]

Defines the macroblock size

[Syntax]

```
typedef enum hiVDA_MB_SIZE_E
{
    VDA_MB_8PIXEL,      /* 8*8 */
    VDA_MB_16PIXEL,     /* 16*16 */
    VDA_MB_BUTT
} VDA_MB_SIZE_E;
```

[Member]

Member	Description
VDA_MB_8PIXEL	Macroblock size of 8x8
VDA_MB_16PIXEL	Macroblock size of 16x16



[Note]

None

[See Also]

None

VDA_MB_SADBITS_E

[Description]

Defines the precision of the MD SAD.

[Syntax]

```
typedef enum hiVDA_MB_SADBITS_E
{
    VDA_MB_SAD_8BIT = 0, /*SAD precision of 8 bits*/
    VDA_MB_SAD_16BIT,    /*SAD precision of 16 bits*/
    VDA_MB_SAD_BUTT      /*reserved*/
} VDA_MB_SADBITS_E;
```

[Member]

Member	Description
VDA_MB_SAD_8BIT	8 bits
VDA_MB_SAD_16BIT	16 bits

[Note]

- The output value is based on the configured macroblock size.
- In 8-bit output mode, the upper eight bits among 16 bits are output.

[See Also]

None

VDA_OD_RGN_ATTR_S

[Description]

Defines the attributes of an occlusion region.

[Syntax]

```
typedef struct hiVDA_OD_RGN_ATTR_S
{
    RECT_S stRect;
    HI_U32 u32SadTh;
    HI_U32 u32AreaTh;
    HI_U32 u32OccCntTh;
```



```
    HI_U32 u32UnOccCntTh;  
}VDA_OD_RGN_ATTR_S;
```

[Member]

Member	Description
stRect	Region range X: value range of [0, VDA_MAX_WIDTH), 16-pixel alignment Y: value range of [0, VDA_MAX_HEIGHT) W: value range of [16, VDA_MAX_WIDTH), 16-pixel alignment H: value range of [16, VDA_MAX_HEIGHT), 16-pixel alignment $X + W \leq$ Channel width $Y + H \leq$ Channel height Static attribute
u32SadTh	Region macroblock alarm threshold Value range: [0, 4080] Dynamic attribute
u32AreaTh	Region area alarm threshold Value range: [0, 100] Dynamic attribute
u32OccCntTh	Region occlusion count alarm threshold Value range: [1, 126] Dynamic attribute
u32UnOccCntTh	Uncover count Value range: [0, 256] Dynamic attribute

[Note]

None

[See Also]

None

VDA_MD_ATTR_S

[Description]

Defines the MD attributes.

[Syntax]

```
typedef struct hiVDA_MD_ATTR_S  
{  
    VDA_ALG_E        enVdaAlg;
```



```
VDA_MB_SIZE_E    enMbSize;
VDA_MB_SADBITS_E enMbSadBits;
VDA_REF_MODE_E   enRefMode;
HI_U32           u32MdBufNum;
HI_U32           u32VdaIntvl;
HI_U32           u32BgUpSrcWgt;
HI_U32           u32SadTh;
HI_U32           u32ObjNumMax;
}VDA_MD_ATTR_S;
```

[Member]

Member	Description
enVdaAlg	VDA algorithm Static attribute
enMbSize	Macroblock size Static attribute
enMbSadBits	SAD output precision Static attribute
enRefMode	Reference picture mode Static attribute
u32MdBufNum	Number of MD result buffers Value range: [1, 16] Static attribute
u32VdaIntvl	VDA interval Value range: [0, 256], in frames Dynamic attribute
u32BgUpSrcWgt	Weight of source pictures when the background is updated. The total weight is 256. Value range: [1, 255] The value 128 is recommended Dynamic attribute
u32SadTh	SAD alarm threshold Dynamic attribute
u32ObjNumMax	Number of output OBJ regions Value range: [1, 128] Dynamic attribute

[Note]

None



[See Also]

None

VDA_OD_ATTR_S

[Description]

Defines the OD attributes.

[Syntax]

```
typedef struct hiVDA_OD_ATTR_S
{
    HI_U32             u32RgnNum;
    VDA_OD_RGN_ATTR_S astOdRgnAttr[VDA_OD_RGN_NUM_MAX];
    VDA_ALG_E          enVdaAlg;
    VDA_MB_SIZE_E      enMbSize;
    VDA_MB_SADBITS_E   enMbSadBits;
    VDA_REF_MODE_E     enRefMode;
    HI_U32             u32VdaIntvl;
    HI_U32             u32BgUpSrcWgt;
}VDA_OD_ATTR_S;
```

[Member]

Member	Description
u32RgnNum	Number of regions Value range: [1, VDA_OD_RGN_NUM_MAX] Static attribute
astOdRgnAttr[VDA_OD_RGN_NUM_MAX]	Region attribute. For details, see the internal structure.
enVdaAlg	VDA algorithm Static attribute
enMbSize	Macroblock size Static attribute
enMbSadBits	SAD output precision Static attribute
enRefMode	Reference picture mode Static attribute
u32VdaIntvl	VDA interval Value range: [0, 256], in frame Dynamic attribute



Member	Description
u32BgUpSrcWgt	Weight of source pictures when the background is updated. The total weight is 256. Value range: [1, 255]. The value 128 is recommended. Dynamic attribute

[Note]

None

[See Also]

None

VDA_WORK_MODE_E

[Description]

Defines the VDA operating mode.

[Syntax]

```
typedef enum hivDA_WORK_MODE_E
{
    VDA_WORK_MODE_MD = 0,
    VDA_WORK_MODE_OD,
    VDA_WORK_MODE_BUTT
}VDA_WORK_MODE_E;
```

[Member]

Member	Description
VDA_WORK_MODE_MD	Motion detection
VDA_WORK_MODE_OD	Occlusion detection

[Note]

None

[See Also]

None

VDA_WORK_MODE_ATTR_U

[Description]

Defines the operating mode attributes.

[Syntax]



```
typedef union hiVDA_WORK_MODE_ATTR_U
{
    VDA_MD_ATTR_S stMdAttr;
    VDA_OD_ATTR_S stOdAttr;
} VDA_WORK_MODE_ATTR_U;
```

[Member]

Member	Description
stMdAttr	Motion detection attribute
stOdAttr	Occlusion detection attribute

[Note]

None

[See Also]

None

VDA_CHN_ATTR_S

[Description]

Defines the channel attributes.

[Syntax]

```
typedef struct hiVDA_CHN_ATTR_S
{
    VDA_WORK_MODE_E enWorkMode;
    VDA_WORK_MODE_ATTR_U unAttr;
    HI_U32 u32Width;
    HI_U32 u32Height;
} VDA_CHN_ATTR_S;
```

[Member]

Member	Description
enWorkMode	Operating mode Static attribute
unAttr	Operating mode attribute
u32Width	Width of a channel Value range: [VDA_MIN_WIDTH, VDA_MAX_WIDTH], 16-byte alignment Static attribute



Member	Description
u32Height	Channel height Value range: [VDA_MIN_HEIGHT, VDA_MAX_HEIGHT], 16-byte alignment Static attribute

[Note]

The channel size can be different from the input picture size. However, the width and height of an input picture must be greater than or equal to the channel width and height.

[See Also]

None

VDA_MB_SAD_DATA_S

[Description]

Defines the SAD information about a macroblock.

[Syntax]

```
typedef struct hiVDA_MB_SAD_DATA_S
{
    HI_VOID      *pAddr;           /*address*/
    HI_U32       u32Stride;        /*stride*/
    VDA_MB_SADBITS_E enMbSadBits; /*MB SAD size*/
} VDA_MB_SAD_DATA_S;
```

[Member]

Member	Description
pAddr	Start address of the memory for storing macroblock SADs
u32Stride	Stride of the memory for storing the macroblock data, in byte
enMbSadBits	SAD output bits

[Note]

None

[See Also]

None

VDA_OBJ_DATA_S

[Description]



Defines the OBJ region information.

[Syntax]

```
typedef struct hiVDA_OBJ_DATA_S
{
    HI_U32 u32ObjNum;
    VDA_OBJ_S *pstAddr;
    HI_U32 u32IndexOfMaxObj;
    HI_U32 u32SizeOfMaxObj;
    HI_U32 u32SizeOfTotalObj;
} VDA_OBJ_DATA_S;
```

[Member]

Member	Description
u32ObjNum	Number of OBJ regions
pstAddr	Start address of the memory for storing the OBJ region information
u32IndexOfMaxObj	Index of the maximum OBJ region
u32SizeOfMaxObj	Size of the maximum OBJ region
u32SizeOfTotalObj	Total size of OBJ regions

[Note]

- The index of the maximum OBJ region is numbered from 0.
- If the reported coordinates of an OBJ region are (0, 4), (16, 36), the size of the OBJ area is calculated as follows:

$$(16 - 0 + 4) \times (36 - 4 + 4) = 720$$

The width of the OBJ region is calculated as follows:

Actual width of the OBJ region = Width obtained from the reported region coordinates + 4.

The height of the OBJ region is calculated in the same way.

- The detected **VDA_OBJ_S** information is stored in the memory starting from pstAddr in sequence. A maximum of u32ObjNum data segments are allowed.

[See Also]

None

VDA_MD_DATA_S

[Description]

Defines the MD results.

[Syntax]

```
typedef struct hiVDA_MD_DATA_S
```



```
{  
    HI_BOOL          bMbSadValid;  
    VDA_MB_SAD_DATA_S stMbSadData;  
    HI_BOOL          bObjValid;  
    VDA_OBJ_DATA_S   stObjData;  
    HI_BOOL          bPelsNumValid;  
    HI_U32           u32AlarmPixCnt;  
}  
VDA_MD_DATA_S;
```

[Member]

Member	Description
bMbSadValid	Validity of the macroblock SAD HI_TRUE: valid HI_FALSE: invalid
stMbSadData	Macroblock SAD result
bObjValid	Validity of the OBJ region HI_TRUE: valid HI_FALSE: invalid
stObjData	OBJ region result
bPelsNumValid	Validity of the number of alarm pixels HI_TRUE: valid HI_FALSE: invalid
u32AlarmPixCnt	Total number of alarm pixels

[Note]

bMbSadValid, **bObjValid**, and **bPelsNumValid** are valid by default. No MPIS are provided for setting these members.

[See Also]

None

VDA_OD_DATA_S

[Description]

Defines the OD results.

[Syntax]

```
typedef struct hiVDA_OD_DATA_S  
{  
    HI_U32    u32RgnNum;  
    HI_BOOL   abRgnAlarm[VDA_OD_RGN_NUM_MAX];
```



```
}VDA_OD_DATA_S;
```

[Member]

Member	Description
u32RgnNum	Number of regions
abRgnAlarm[VDA_OD_RGN_NUM_MAX]	Region alarm flag HI_TRUE: alarm HI_FALSE: no alarm

[Note]

None

[See Also]

None

VDA_DATA_U

[Description]

Defines the VDA result union.

[Syntax]

```
typedef union hiVDA_DATA_U
{
    VDA_MD_DATA_S stMdData;
    VDA_OD_DATA_S stOdData;
}VDA_DATA_U;
```

[Member]

Member	Description
stMdData	Motion detection result
stOdData	Occlusion detection result

[Note]

None

[See Also]

None

VDA_DATA_S

[Description]



Defines the VDA results

[Syntax]

```
typedef struct hiVDA_DATA_S
{
    VDA_WORK_MODE_E enWorkMode;
    VDA_DATA_U     unData;
    VDA_MB_SIZE_E   enMbSize;
    HI_U32          u32MbWidth;
    HI_U32          u32MbHeight;
    HI_U64          u64Pts;
} VDA_DATA_S;
```

[Member]

Member	Description
enWorkMode	Operating mode
unData	Result union
enMbSize	Macroblock size
u32MbWidth	Channel width, in macroblock
u32MbHeight	Channel height, in macroblock
u64Pts	PTS

[Note]

None

[See Also]

None

VDA_CHN_STAT_S

[Description]

Defines the status of a VDA channel.

[Syntax]

```
typedef struct hiVDA_CHN_STAT_S
{
    HI_BOOL bStartRecvPic;
    HI_U32  u32LeftPic;
    HI_U32  u32LeftRst;
} VDA_CHN_STAT_S;
```

[Member]



Member	Description
bStartRecvPic	Whether to receive pictures
u32LeftPic	Number of remaining pictures in the channel picture queue
u32LeftRst	Number of remaining results in the buffer

[Note]

None

[See Also]

None

7.5 Error Codes

Table 7-1 describes the error codes for VDA APIs.

Table 7-1 Error codes for VDA APIs

Error Code	Error Code	Description
0xA0098001	HI_ERR_VDA_INVALID_DEVID	The device ID exceeds the valid range.
0xA0098002	HI_ERR_VDA_INVALID_CHNID	The channel ID exceeds the valid range.
0xA0098003	HI_ERR_VDA_ILLEGAL_PARAM	The parameter value exceeds its valid range.
0xA0098004	HI_ERR_VDA_EXIST	The device, channel, or resource to be created or applied for already exists.
0xA0098005	HI_ERR_VDA_UNEXIST	The device, channel, or resource to be used or destroyed does not exist.
0xA0098006	HI_ERR_VDA_NULL_PTR	The pointer is null.
0xA0098007	HI_ERR_VDA_NOT_CONFIG	The system or VDA channel is not configured.
0xA0098008	HI_ERR_VDA_NOT_SUPPORT	The parameter or function is not supported.
0xA0098009	HI_ERR_VDA_NOT_PERM	The operation, for example, attempting to modify the value of a static parameter, is forbidden.
0xA009800C	HI_ERR_VDA_NOMEM	The memory fails to be allocated due to some causes such as insufficient system memory.



Error Code	Error Code	Description
0xA009800D	HI_ERR_VDA_NOBUF	The buffer fails to be allocated due to some causes such as oversize of the data buffer applied for.
0xA009800E	HI_ERR_VDA_BUF_EMPTY	The buffer is empty.
0xA009800F	HI_ERR_VDA_BUF_FULL	The buffer is full.
0xA0098010	HI_ERR_VDA_SYS_NOTREADY	The system is not initialized or the corresponding module is not loaded.
0xA0098012	HI_ERR_VDA_BUSY	The system is busy.



Contents

8 Region Management.....	8-1
8.1 Overview	8-1
8.2 Function Description	8-1
8.2.1 Concepts.....	8-1
8.2.2 Usage Guidelines	8-6
8.3 API Reference	8-6
8.4 Data Structures	8-22
8.5 Error Codes	8-49



Figures

Figure 8-1 Region overlay.....	8-2
Figure 8-2 Cover/CoverEx region exceeding the boundaries.....	8-5
Figure 8-3 Non-16-pixel aligned overlay region	8-41



Tables

Table 8-1 Modules supporting the Hi3516A/Hi3518E V200/Hi3519 V100 region	8-3
Table 8-2 Hi3516A/Hi3518E V200/Hi3519 V100 region functions	8-4
Table 8-3 Chip differences in region functions.....	8-5
Table 8-4 Chip differences in Cover/CoverEx types	8-5
Table 8-5 Error codes for region management APIs.....	8-49



8 Region Management

8.1 Overview

To display specific information such as the channel ID and the PTS, you need to overlay OSDs on the video. You may also need to fill color blocks. The OSDs and color blocks for covering videos are called regions. The REGION module manages these regions.

Region management involves creating regions and overlaying regions on the video or covering video. After creating a region, you can call `HI_MPI_RGN_AttachToChn` to overlay this region to channels such as multiple VENC channels, multiple VI channels, or multiple VENC channels and VI channels. The display attributes, including the position, layer, and transparency of each channel, can be different. When the channel is scheduled, OSDs are overlaid on the video.

8.2 Function Description

8.2.1 Concepts

- Region type
 - Overlay: video overlay region. It supports bitmap loading and its background color can be refreshed.
 - OverlayEx: extended video overlay region. Similar to the video overlay region, the extended video overlay region also supports bitmap loading and its background color can be refreshed.
 - Cover: video cover region. It can be covered by pure color blocks.
 - CoverEx: extended video cover region. Similar to the video cover region, the extended video cover region can also be covered by pure color blocks.
 - OverlayEx/CoverEx: region corresponding to the overlay/cover region. The functions of the OverlayEx and CoverEx regions are similar to those of the overlay and cover regions respectively. However, when the OverlayEx or CoverEx region is used, extra system bandwidth is consumed. The OverlayEx and CoverEx regions are overlaid to pictures by the video graphics subsystem (VGS). A larger OverlayEx or CoverEx region indicates that higher VGS performance is required. If the VGS performance is insufficient, the frame rate will be decreased. It is recommended that the OverlayEx and CoverEx regions be used only when the overlay and cover regions are not supported or the number of overlay and cover regions is insufficient.

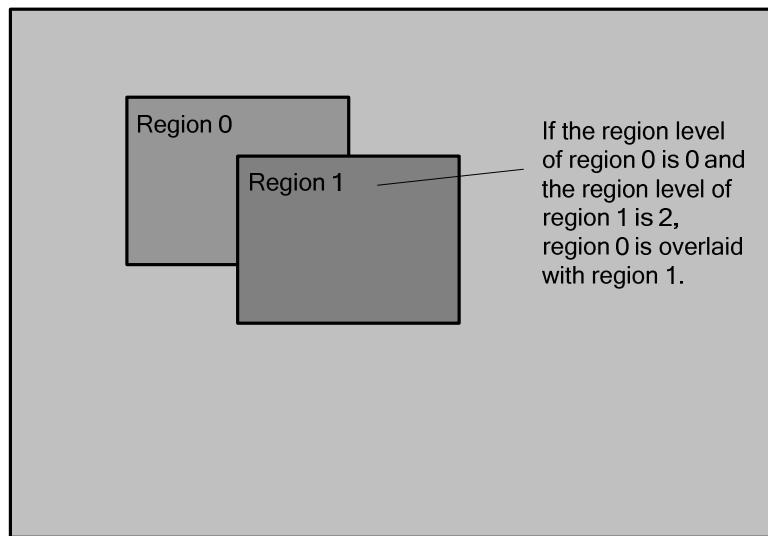


- Region level

The region level indicates the overlay level. A higher region level indicates a higher region display level. When regions are overlaid, the region with a lower level is overlaid with the region with a higher level. If the regions with the same level are overlaid, the existing region is overlaid with the new region.

When multiple regions are overlaid with the same VPSS group, the level of each region must be unique. Otherwise, only the last region at the same level can be overlaid with the VPSS group. However, multiple regions at the same level can be overlaid with the same VENC channel.

Figure 8-1 Region overlay



- Bitmap filling (valid for overlay and OverlayEx)

Bitmap filling indicates that the memory value of a bitmap is filled in the region memory from the upper left corner of the region. When the bitmap size is smaller than the region size, a part of memory is filled, and the remaining part is retained. When the sizes of the bitmap and region are the same, the region memory is filled completely. When the bitmap size is greater than the region size, the region memory is filled with the information of the bitmap size.

Bitmap filling is implemented in either of the following ways:

- Copy bitmap data to the internal display canvas by calling [HI_MPI_RGN_SetBitMap](#).
- Obtain the address for the internal backup display canvas by calling [HI_MPI_RGN_GetCanvasInfo](#), update the data at this address, and then update the backup display canvas as the canvas to be displayed by calling [HI_MPI_RGN_UpdateCanvas](#). In this way, bitmap data is updated.

- Region attribute

After creating a region, you must set its attributes, including public resource information. For example, the overlay attribute includes the pixel format, region size, and background color.

- Channel display attribute ([RGN_CHN_ATTR_S](#))

The channel display attributes indicate the region display features in a channel. For example, the overlay channel display attributes include the display position, level, foreground alpha, background alpha, and quantizer parameter (QP) information required



for encoding. When the region display attribute parameter bShow is set to TRUE, the region is displayed in the channel; otherwise, the region is hidden in the channel.

- Region color inversion

When a region is overlaid with a video, it is difficult to distinguish the video background from the overlaid region if their luminance and chrominance are close. By using the region color inversion function, the background variance is adapted, and the luminance and chrominance of the region is adjusted, which makes the region clear.

The region color inversion function can be implemented by using the region luminance statistics function provided by the VPSS. This function enables you to collect statistics on the luminance of the background of each region to be overlaid in the video sequence in real time. Then you can manually invert the region color by using the raster operation (ROP) function of the TDE, and overlay the video with the region with the inverted color by using the VPSS.

- Region QP protection

When a video is overlaid with a region and then data is compressed and encoded, the compression features of the overlaid region can be separately configured by setting the QP protection parameter. This ensures the overlaid region is clear after data compression. Only the overlay region supports QP protection, and QP protection is valid only for the H.264/H.265 encoding channel.

- Supported modules

[Table 8-1](#) describes the modules that support regions. You need to attach regions to channels by following [Table 8-1](#).

NOTE

The VIU supports regions only in offline mode. The region in concave quadrilateral is displayed as a region in triangle.

Table 8-1 Modules supporting the Hi3516A/Hi3518E V200/Hi3519 V100 region

Region Type	Module	Value Range of the Device ID	Value Range of the Channel ID
OVERLAY	VENC	0	[0, VENC_MAX_CHN_NUM-1]
COVER	VPSS	[0, VPSS_MAX_GRP_NUM-1]	0
OVERLAYEX	VPSS	[0, VPSS_MAX_GRP_NUM-1]	[0, VPSS_MAX_PHY_CHN_NUM-1]
	VO	[0, VO_MAX_LAYER_NUM-1]	[0, VO_MAX_CHN_NUM-1]
	VI	0	[0, VIU_MAX_CHN_NUM-1]
	PCIV	0	[0, PCIV_MAX_CHN_NUM-1]
COVEREX	VPSS	[0, VPSS_MAX_GRP_NUM-1]	[0, VPSS_MAX_PHY_CHN_NUM-1]



Region Type	Module	Value Range of the Device ID	Value Range of the Channel ID
	VO	[0, VO_MAX_LAYER_ NUM-1]	[0, VO_MAX_CHN_NUM-1]
	VI	0	[0, VIU_MAX_CHN_NUM-1]

- Region functions
describes the functions of each region.

NOTE

- For details about the contents when the region module supports the VOU, see chapter 4 "VO."
- The Hi3516A and Hi3518E V200 do not support the PCIV function.

Table 8-2 Hi3516A/Hi3518E V200/Hi3519 V100 region functions

Item	Overlay	Cover	OverlayEx				CoverEx		
Module	VENC	VPSS	VPSS	VO	VI	PCIV	VPSS	VO	VI
Pixel format	Argb1555 Argb4444	N/A	Argb155 5 Argb444 4 Argb888 8	Argb155 5 Argb444 4 Argb888 8	Argb155 5 Argb444 4 Argb888 8	Argb15 55 Argb44 44 Argb88 88	N/A	N/A	N/A
Overlay level	Supported	Supported	Supported	Supported	Supported	Supported	Supported	Supported	Supported
Bitmap filling	Supported	N/A	Supported	Supported	Supported	Supported	N/A	N/A	N/A
Overlay transparency	Supported	N/A	Supported	Supported	Supported	Supported	N/A	N/A	N/A
Foreground alpha range	0–128	N/A	0–255	0–255	0–255	0–255	N/A	N/A	N/A
Background alpha range	0–128	N/A	0–255	0–255	0–255	0–255	N/A	N/A	N/A
QP protection	Supported	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Color inversion	Supported (implemented by users)	N/A	Supported (implemented by users)	Supported (implemented by users)	Supported (implemented by users)	Supported (implemented by users)	N/A	N/A	N/A



Table 8-3 Chip differences in region functions

Function	Chip		
	Hi3516A	Hi3518E V200	Hi3519 V100
Cover VPSS	Neither some nor all the coordinates of the quadrilateral can exceed the valid picture boundaries. (Some coordinates of the quadrilateral can exceed the valid picture boundaries if the quadrilateral is a rectangle.)	Some or all the coordinates of the quadrilateral can exceed the valid picture boundaries. The region takes effect when the quadrilateral intersects with the valid picture, and does not take effect when the quadrilateral does not intersect with the valid picture.	Some or all the coordinates of the quadrilateral can exceed the valid picture boundaries. The region takes effect when the quadrilateral intersects with the valid picture, and does not take effect when the quadrilateral does not intersect with the valid picture.
CoverEx VI/VPSS/VO	Neither some nor all the coordinates of the quadrilateral can exceed the valid picture boundaries. (Some coordinates of the quadrilateral can exceed the valid picture boundaries if the quadrilateral is a rectangle.)	Some or all the coordinates of the quadrilateral can exceed the valid picture boundaries. The region takes effect when the quadrilateral intersects with the valid picture, and does not take effect when the quadrilateral does not intersect with the valid picture.	Some or all the coordinates of the quadrilateral can exceed the valid picture boundaries. The region takes effect when the quadrilateral intersects with the valid picture, and does not take effect when the quadrilateral does not intersect with the valid picture.
Overlay JPEGE	Supported	Supported	Supported

Figure 8-2 Cover/CoverEx region exceeding the boundaries

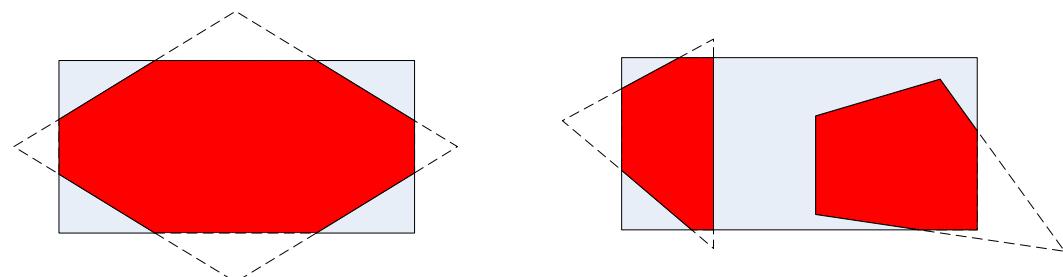


Table 8-4 Chip differences in Cover/CoverEx types

Chip	Cover/CoverEx Type					
	Cover (Rectangle)	Cover (Solid Quadrilateral)	Cover (Dashed Quadrilateral)	CoverEx (Rectangle)	CoverEx (Solid Quadrilateral)	CoverEx (Dashed Quadrilateral)
Hi3516A	Supported	Supported	Not supported	Supported	Supported	Supported



Hi3518E V200	Supported	Supported	Not supported	Supported	Supported	Supported
Hi3519 V100	Supported	Supported	Not supported	Supported	Supported	Supported

8.2.2 Usage Guidelines

To create and use a region, perform the following operations:

- Set region attributes and create a region.
- Set the display attributes of a channel such as a VENC channel, and attach the region to the channel.

To control region attributes and channel display attributes, perform the following operations:

- Obtain region attributes by calling [HI_MPI_RGN_GetAttr](#), and set region attributes by calling [HI_MPI_RGN_SetAttr](#).
- Set the bitmap information about a region by calling [HI_MPI_RGN_SetBitMap](#) (applicable only to Overlay and OverlayEx).
- Obtain the channel display attributes of a channel such as a VENC channel by calling [HI_MPI_RGN_GetDisplayAttr](#), and set channel display attributes by calling [HI_MPI_RGN_SetDisplayAttr](#).
- (Optional) Detach the region from the channel, and destroy the region.

For details, see related samples.

8.3 API Reference

The REGION module manages regions, including creating and destroying regions, obtaining and setting region attributes, and obtaining and setting channel display attributes of regions.

The module provides the following MPIs:

- [HI_MPI_RGN_Create](#): Creates a region.
- [HI_MPI_RGN_Destroy](#): Destroys a region.
- [HI_MPI_RGN_GetAttr](#): Obtains region attributes.
- [HI_MPI_RGN_SetAttr](#): Sets region attributes.
- [HI_MPI_RGN_SetBitMap](#): Sets a region bitmap.
- [HI_MPI_RGN_SetAttachField](#): Sets the frame or field picture to be overlaid when a region is overlaid with the channel picture.
- [HI_MPI_RGN_GetAttachField](#): Obtains the frame or field picture to be overlaid when a region is overlaid with the channel picture.
- [HI_MPI_RGN_AttachToChn](#): Overlays a channel with a region.
- [HI_MPI_RGN_DetachFromChn](#): Detaches a region from a channel.
- [HI_MPI_RGN_SetDisplayAttr](#): Sets the channel display attributes of a region.
- [HI_MPI_RGN_GetDisplayAttr](#): Obtains the channel display attributes of a region.
- [HI_MPI_RGN_SetDisplayAttr](#): Obtains the channel display attributes of a region.



- [HI_MPI_RGN_UpdateCanvas](#): Updates the display canvas.

HI_MPI_RGN_Create

[Description]

Creates a region.

[Syntax]

```
HI_S32 HI_MPI_RGN_Create(RGN_HANDLE Handle, const RGN_ATTR_S *pstRegion);
```

[Parameter]

Parameter	Description	Input/Output
Handle	Region handle ID The handle ID must be unique. Value range: [0, RGN_HANDLE_MAX)	Input
pstRegion	Pointer to region attributes	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 8.5 "Error Codes."

[Requirement]

- Header files: hi_comm_region.h, mpi_region.h
- Library file: libmpi.a

[Note]

- The handle is specified by users and is equivalent to the ID.
- A region handle ID can be used once only.
- Region attributes must be valid. For details about restrictions, see the description of [RGN_ATTR_S](#).
- The region attribute pointer cannot be null.
- Before creating a Cover or CoverEx region, you need only to specify the region type. Other attributes such as the region position and layer are specified when you call [HI_MPI_RGN_AttachToChn](#).
- When a region is created by calling this MPI, only the basic parameters such as the minimum width and height and maximum width and height are set. When the region is attached to a channel, specific parameters (such as the supported pixel format) are checked based on the restrictions specified by the supported channel modules.

[Example]



```
RGN_HANDLE Handle;
RGN_ATTR_S stRgnAttr;
HI_S32 s32Ret = HI_SUCCESS;
Handle = 0;
stRgnAttr.enType = OVERLAY_RGN;
stRgnAttr.unAttr.stOverlay.enPixelFormat = PIXEL_FORMAT_RGB_1555;
stRgnAttr.unAttr.stOverlay.stSize.u32Width = 16;
stRgnAttr.unAttr.stOverlay.stSize.u32Height = 16;
stRgnAttr.unAttr.stOverlay.u32BgColor = 0x0000ffff;

s32Ret = HI_MPI_RGN_Create(Handle, &stRgnAttr);
if(s32Ret != HI_SUCCESS)
{
    return s32Ret;
}

s32Ret = HI_MPI_RGN_GetAttr(Handle, &stRgnAttr);
if(s32Ret != HI_SUCCESS)
{
    return s32Ret;
}

stRgnAttr.unAttr.stOverlay.u32BgColor = 0x0000cccc;
s32Ret = HI_MPI_RGN_SetAttr(Handle, &stRgnAttr);
if(s32Ret != HI_SUCCESS)
{
    return s32Ret;
}

s32Ret = HI_MPI_RGN_Destroy(Handle);
if(s32Ret != HI_SUCCESS)
{
    return s32Ret;
}
```

[See Also]

- [HI_MPI_RGN_Destroy](#)
- [HI_MPI_RGN_GetAttr](#)
- [HI_MPI_RGN_SetAttr](#)

HI_MPI_RGN_Destroy

[Description]

Destroys a region.

[Syntax]

```
HI_S32 HI_MPI_RGN_Destroy(RGN_HANDLE Handle);
```



[Parameter]

Parameter	Description	Input/Output
Handle	Region handle ID Value range: [0, RGN_HANDLE_MAX)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 8.5 "Error Codes."

[Requirement]

- Header files: hi_comm_region.h, mpi_region.h
- Library file: libmpi.a

[Note]

Create a region before calling this MPI.

[Example]

For details, see the sample of [HI_MPI_RGN_Create](#).

[See Also]

None

HI_MPI_RGN_GetAttr

[Description]

Obtains region attributes.

[Syntax]

```
HI_S32 HI_MPI_RGN_GetAttr(RGN_HANDLE Handle, RGN_ATTR_S *pstRegion);
```

[Parameter]

Parameter	Description	Input/Output
Handle	Region handle ID Value range: [0, RGN_HANDLE_MAX)	Input
pstRegion	Pointer to region attributes	Output

[Return Value]



Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 8.5 "Error Codes."

[Requirement]

- Header files: hi_comm_region.h, mpi_region.h
- Library file: libmpi.a

[Note]

- Create a region before calling this MPI.
- The region attribute pointer cannot be null.

[Example]

For details, see the sample of [HI_MPI_RGN_Create](#).

[See Also]

None

HI_MPI_RGN_SetAttr

[Description]

Sets region attributes.

[Syntax]

```
HI_S32 HI_MPI_RGN_SetAttr(RGN_HANDLE Handle, const RGN_ATTR_S *pstRegion);
```

[Parameter]

Parameter	Description	Input/Output
Handle	Region handle ID Value range: [0, RGN_HANDLE_MAX)	Input
pstRegion	Pointer to region attributes	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 8.5 "Error Codes."



[Requirement]

- Header files: hi_comm_region.h, mpi_region.h
- Library file: libmpi.a

[Note]

- Create a region before calling this MPI.
- If a region is attached to a channel by calling [HI_MPI_RGN_AttachToChn](#), only the dynamic attributes can be changed by calling [HI_MPI_RGN_SetAttr](#). If a region is not attached, both the static and dynamic attributes can be changed by calling [HI_MPI_RGN_SetAttr](#).
- The region attribute pointer cannot be null.
- After [HI_MPI_RGN_GetCanvasInfo](#) is called, calling this MPI has no effect unless the canvas is updated by calling [HI_MPI_RGN_UpdateCanvas](#).

[Example]

For details, see the sample of [HI_MPI_RGN_Create](#).

[See Also]

None

HI_MPI_RGN_SetBitMap

[Description]

Set a region bitmap, that is, fills in the region bitmap.

[Syntax]

```
HI_S32 HI_MPI_RGN_SetBitMap(RGN_HANDLE Handle, const BITMAP_S *pstBitmap);
```

[Parameter]

Parameter	Description	Input/Output
Handle	Region handle ID Value range: [0, RGN_HANDLE_MAX)	Input
pstBitmap	Pointer to bitmap attributes	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 8.5 "Error Codes."

[Requirement]

- Header files: hi_comm_region.h, mpi_region.h



- Library file: libmpi.a

[Note]

- Create a region before calling this MPI.
- The sizes of the bitmap and region can be different.
- The bitmap is loaded from the (0, 0) of the region. If the bitmap size is greater than the region size, the bitmap is automatically cropped to the region size.
- The pixel formats of the bitmap and region must be the same.
- The bitmap attribute pointer cannot be null.
- This MPI can be called repeatedly.
- This MPI applies only to the overlay and OverlayEx regions.
- After [HI_MPI_RGN_GetCanvasInfo](#) is called, calling [HI_MPI_RGN_SetBitMap](#) has no effect unless the canvas is updated by calling [HI_MPI_RGN_UpdateCanvas](#).

[Example]

```
RGN_HANDLE Handle;
BITMAP_S stBitmap;
HI_S32 s32Ret = HI_SUCCESS;
stBitmap.enPixelFormat = PIXEL_FORMAT_RGB_1555;
stBitmap.u32Width = 4;
stBitmap.u32Height = 4;
stBitmap.pData = malloc(2 * stBitmap.u32Width * stBitmap.u32Height);
if(HI_NULL == stBitmap.pData)
{
    return HI_FAILURE;
}
memset(stBitmap.pData, 0xcc, u32Size);

s32Ret = HI_MPI_RGN_SetBitMap(Handle, &stBitmap);
if(s32Ret != HI_SUCCESS)
{
    free(stBitmap.pData);
    return s32Ret;
}
free(stBitmap.pData);
```

[See Also]

None

HI_MPI_RGN_SetAttachField

[Description]

Sets the frame or field picture to be overlaid when a region is overlaid with the channel picture.

[Syntax]



```
HI_S32 HI_MPI_RGN_SetAttachField(RGN_HANDLE Handle, VIDEO_FIELD_E  
enAttachField);
```

[Parameter]

Parameter	Description	Input/Output
Handle	Region handle ID Value range: [0, RGN_HANDLE_MAX)	Input
enAttachField	Identifier of a frame/field to be overlaid with a region	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 8.5 "Error Codes."

[Requirement]

- Header files: hi_comm_region.h, mpi_region.h
- Library file: libmpi.a

[Note]

- Create a region before calling this MPI.
- This MPI supports only the COVEREX_RGN or OVERLAYEX_RGN region.
- When a region is overlaid by field, the coordinates, width, and height of the region are configured based on the entire frame.
- The function of overlaying a region by field is valid only for interlaced 2-field pictures from the VIU. If you overlay a frame picture with a region or overlay a region of other modules, calling this MPI has no effect.
- Call HI_MPI_RGN_SetAttachField before calling [HI_MPI_RGN_AttachToChn](#).
- When a top or bottom field is overlaid with a region, the vertical coordinate and height of the region must be 4-pixel-aligned. When a frame is overlaid with a region, the vertical coordinate and height of the region must be 2-pixel-aligned.

[Example]

None

[See Also]

None

HI_MPI_RGN_GetAttachField

[Description]



Obtains the frame or field picture to be overlaid when a region is overlaid with the channel picture.

[Syntax]

```
HI_S32 HI_MPI_RGN_GetAttachField(RGN_HANDLE Handle, VIDEO_FIELD_E *penAttachField);
```

[Parameter]

Parameter	Description	Input/Output
Handle	Region handle ID Value range: [0, RGN_HANDLE_MAX)	Input
penAttachField	Pointer to the Identifier of a frame/field to be overlaid with a region	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 8.5 "Error Codes."

[Requirement]

- Header files: hi_comm_region.h, mpi_region.h
- Library file: libmpi.a

[Note]

- Create a region before calling this MPI.
- This MPI supports only the COVEREX_RGN or OVERLAYEX_RGN region.

[Example]

None

[See Also]

None

HI_MPI_RGN_AttachToChn

[Description]

Overlays a channel with a region.

[Syntax]

```
HI_S32 HI_MPI_RGN_AttachToChn(RGN_HANDLE Handle, const MPP_CHN_S *pstChn,  
const RGN_CHN_ATTR_S *pstChnAttr);
```



[Parameter]

Parameter	Description	Input/Output
Handle	Region handle ID Value range: [0, RGN_HANDLE_MAX)	Input
pstChn	Pointer to the channel structure	Input
pstChnAttr	Pointer to the channel display attributes of a region	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 8.5 "Error Codes."

[Requirement]

- Header files: hi_comm_region.h, mpi_region.h
- Library file: libmpi.a

[Note]

- Create a region before calling this MPI.
- The pointer that points to the channel structure cannot be null.
- The pointer that points to channel display attributes cannot be null.
- This MPI can be called repeatedly, but cannot be used to modify attributes.
- For the overlay area, pay attention to QP information. For details, see the description of [OVERLAY_QP_INFO_S](#).
- A region must be overlaid with channel 0 of a VPSS group. Otherwise, the region fails to be overlaid.

[Example]

None

[See Also]

None

HI_MPI_RGN_DetachFromChn

[Description]

Detaches a region from a channel.

[Syntax]

```
HI_S32 HI_MPI_RGN_DetachFromChn(RGN_HANDLE Handle, const MPP_CHN_S  
*pstChn);
```



[Parameter]

Parameter	Description	Input/Output
Handle	Region handle ID Value range: [0, RGN_HANDLE_MAX)	Input
pstChn	Pointer to the channel structure	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 8.5 "Error Codes."

[Requirement]

- Header files: hi_comm_region.h, mpi_region.h
- Library file: libmpi.a

[Note]

- Create a region before calling this MPI.
- The pointer that points to the channel structure cannot be null.
- This MPI can be called repeatedly.

[Example]

None

[See Also]

None

HI_MPI_RGN_SetDisplayAttr

[Description]

Sets the channel display attributes of a region.

[Syntax]

```
HI_S32 HI_MPI_RGN_SetDisplayAttr(RGN_HANDLE Handle, const MPP_CHN_S
*pstChn, const RGN_CHN_ATTR_S *pstChnAttr);
```

[Parameter]

Parameter	Description	Input/Output
Handle	Region handle ID Value range: [0, RGN_HANDLE_MAX)	Input



Parameter	Description	Input/Output
pstChn	Pointer to the channel structure	Input
pstChnAttr	Pointer to the channel display attributes of a region	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 8.5 "Error Codes."

[Requirement]

- Header files: hi_comm_region.h, mpi_region.h
- Library file: libmpi.a

[Note]

- Create a region before calling this MPI.
- You are advised to obtain attributes before setting attributes.
- The pointer that points to the channel structure cannot be null.
- The pointer that points to channel display attributes cannot be null.
- Overlay the channel with a region before calling this MPI.
- Only dynamic attributes can be modified. For details, see the description of [RGN_CHN_ATTR_S](#).

[Example]

```
HI_S32 s32Ret = HI_SUCCESS;
    RGN_HANDLE Handle;
    MPP_CHN_S stChn;
    RGN_ATTR_S stRgnAttr;
    RGN_CHN_ATTR_S stChnAttr;
    Handle = 0;
    stChn.enModId = HI_ID_VENC;
    stChn.s32DevId = 0;
    stChn.s32ChnId = 0;

    s32Ret = HI_MPI_RGN_GetDisplayAttr(Handle, &stChn, &stChnAttr);
    if(s32Ret != HI_SUCCESS)
    {
        NOT_PASS(s32Ret);
        goto END;
    }
```



```
stChnAttr.bShow = HI_TRUE;
stChnAttr.enType = OVERLAY_RGN;
stChnAttr.unChnAttr.stOverlayChn.stPoint.s32X = 4;
stChnAttr.unChnAttr.stOverlayChn.stPoint.s32Y = 4;
stChnAttr.unChnAttr.stOverlayChn.u32BgAlpha = 128;
stChnAttr.unChnAttr.stOverlayChn.u32FgAlpha = 128;
stChnAttr.unChnAttr.stOverlayChn.u32Layer = 0;

stChnAttr.unChnAttr.stOverlayChn.stQpInfo.bAbsQp = HI_FALSE;
stChnAttr.unChnAttr.stOverlayChn.stQpInfo.s32Qp = 0;

s32Ret = HI_MPI_RGN_SetDisplayAttr(Handle, &stChn, &stChnAttr);
if(s32Ret != HI_SUCCESS)
{
    return s32Ret;
}
```

[See Also]

None

HI_MPI_RGN_SetDisplayAttr

[Description]

Obtains the channel display attributes of a region.

[Syntax]

```
HI_S32 HI_MPI_RGN_SetDisplayAttr(RGN_HANDLE Handle, const MPP_CHN_S
*pstChn, RGN_CHN_ATTR_S *pstChnAttr);
```

[Parameter]

Parameter	Description	Input/Output
Handle	Region handle ID Value range: [0, RGN_HANDLE_MAX)	Input
pstChn	Pointer to the channel structure	Input
pstChnAttr	Pointer to the channel display attributes of a region	Output

[Return Value]

Return Value	Description
0	Success



Other values	Failure. Its value is an error code. For details, see section 8.5 "Error Codes."
--------------	--

[Requirement]

- Header files: hi_comm_region.h, mpi_region.h
- Library file: libmpi.a

[Note]

- Create a region before calling this MPI.
- The pointer that points to the channel structure cannot be null.
- The pointer that points to channel display attributes cannot be null.

[Example]

See the example of [HI_MPI_RGN_SetDisplayAttr](#).

[See Also]

None

HI_MPI_RGN_GetCanvasInfo

[Description]

Obtains the display canvas information corresponding to a region.

[Syntax]

```
HI_S32 HI_MPI_RGN_GetCanvasInfo(RGN_HANDLE Handle, RGN_CANVAS_INFO_S
*pstCanvasInfo);
```

[Parameter]

Parameter	Description	Input/Output
Handle	Region handle ID Value range: [0, RGN_HANDLE_MAX)	Input
pstCanvasInfo	Display canvas information about a region	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 8.5 "Error Codes."

[Requirement]



- Header files: hi_comm_region.h, mpi_region.h
- Library file: libmpi.a

[Note]

- Create a region before calling this MPI.
- Similar to [HI_MPI_RGN_SetBitMap](#), [HI_MPI_RGN_GetCanvasInfo](#) is used to load bitmap data to the overlay or OverlayEx region. Compared with [HI_MPI_RGN_SetBitMap](#), [HI_MPI_RGN_GetCanvasInfo](#) enables you to directly update the data of the display canvas, which reduces the operation of copying data from the memory.
- This MPI is used to obtain the canvas information corresponding to a region. After obtaining the canvas address, you can directly operate the canvas. For example, you can fill the canvas with bitmap data, and then update the display canvas data by calling [HI_MPI_RGN_UpdateCanvas](#).
- [HI_MPI_RGN_GetCanvasInfo](#) and [HI_MPI_RGN_SetBitMap](#) are mutually exclusive. If [HI_MPI_RGN_GetCanvasInfo](#) is called, calling [HI_MPI_RGN_SetBitMap](#) has no effect before [HI_MPI_RGN_UpdateCanvas](#) is called.

[Example]

```
RGN_HANDLE Handle;
RGN_ATTR_S stRgnAttr;
HI_S32 s32Ret = HI_SUCCESS;
Handle = 0;
stRgnAttr.enType = OVERLAY_RGN;
stRgnAttr.unAttr.stOverlay.enPixelFormat = PIXEL_FORMAT_RGB_1555;
stRgnAttr.unAttr.stOverlay.stSize.u32Width = 16;
stRgnAttr.unAttr.stOverlay.stSize.u32Height = 16;
stRgnAttr.unAttr.stOverlay.u32BgColor = 0x0000ffff;
/* 1. Create handle for overlay region */
s32Ret = HI_MPI_RGN_Create(Handle, &stRgnAttr);
if (s32Ret != HI_SUCCESS)
{
    return s32Ret;
}

/* 2. Get current overlay canvas info */
s32Ret = HI_MPI_RGN_GetCanvasInfo(Handle, &stCanvasInfo);
if (HI_SUCCESS != s32Ret)
{
    printf("HI_MPI_RGN_GetCanvasInfo fail! s32Ret: 0x%x.\n", s32Ret);
    return NULL;
}
/* 3. User can set canvas here. */
/* 4. Update canvas */
s32Ret = HI_MPI_RGN_UpdateCanvas(Handle);
if (HI_SUCCESS != s32Ret)
```



```
{  
    printf("HI_MPI_RGN_UpdateCanvas fail! s32Ret: 0x%x.\n", s32Ret);  
    return NULL;  
}
```

[See Also]

None

HI_MPI_RGN_UpdateCanvas

[Description]

Updates the display canvas.

[Syntax]

```
HI_S32 HI_MPI_RGN_UpdateCanvas(RGN_HANDLE Handle);
```

[Parameter]

Parameter	Description	Input/Output
Handle	Region handle ID Value range: [0, RGN_HANDLE_MAX)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 8.5 "Error Codes."

[Requirement]

- Header files: hi_comm_region.h, mpi_region.h
- Library file: libmpi.a

[Note]

- Create a region before calling this MPI.
- HI_MPI_RGN_UpdateCanvas works with [HI_MPI_RGN_GetCanvasInfo](#) to switch the display canvas after the canvas memory data is updated.
- Obtain canvas information each time before calling this MPI.
- This MPI supports only the overlay or OverlayEx region.

[Example]

See the example of [HI_MPI_RGN_GetCanvasInfo](#).

[See Also]



None

8.4 Data Structures

The data structures related to region management are as follows:

- **RGN_HANDLE_MAX**: Defines the maximum number of region handles.
- **RGN_COVER_MIN_X**: Defines the minimum value of the horizontal coordinate for the start position of a rectangular or quadrilateral Cover/CoverEx region.
- **RGN_COVER_MIN_Y**: Defines the minimum value of the vertical coordinate for the start position of a rectangular or quadrilateral Cover/CoverEx region.
- **RGN_COVER_MAX_X**: Defines the maximum value of the horizontal coordinate for the start position of a rectangular or quadrilateral Cover/CoverEx region.
- **RGN_COVER_MAX_Y**: Defines the maximum value of the vertical coordinate for the start position of a rectangular or quadrilateral Cover/CoverEx region.
- **RGN_COVER_MAX_WIDTH**: Defines the maximum width of a rectangular Cover/CoverEx region.
- **RGN_COVER_MAX_HEIGHT**: Defines the maximum height of a rectangular Cover/CoverEx region.
- **RGN_OVERLAY_MIN_X**: Defines the minimum value of the horizontal coordinate for the start position of an Overlay/OverlayEx region.
- **RGN_OVERLAY_MIN_Y**: Defines the minimum value of the vertical coordinate for the start position of an Overlay/OverlayEx region.
- **RGN_OVERLAY_MAX_X**: Defines the maximum value of the horizontal coordinate for the start position of an Overlay/OverlayEx region.
- **RGN_OVERLAY_MAX_Y**: Defines the maximum value of the vertical coordinate for the start position of an Overlay/OverlayEx region.
- **RGN_OVERLAY_MAX_WIDTH**: Defines the maximum width of an Overlay/OverlayEx region.
- **RGN_OVERLAY_MAX_HEIGHT**: Defines the maximum height of an Overlay/OverlayEx region.
- **OVERLAY_MAX_NUM_VENC**: Defines the maximum number of overlay regions for a VENC channel.
- **OVERLAY_MAX_NUM_VPSS**: Defines the maximum number of overlay regions for a VPSS channel.
- **OVERLAYEX_MAX_NUM_VPSS**: Defines the maximum number of OverlayEx regions for a VPSS channel.
- **OVERLAYEX_MAX_NUM_VO**: Defines the maximum number of OverlayEx regions for a VO channel.
- **OVERLAYEX_MAX_NUM_PCIV**: Defines the maximum number of OverlayEx regions for a PCIV channel.
- **COVEREX_MAX_NUM_VPSS**: Defines the maximum number of CoverEx regions for a VPSS channel.
- **COVER_MAX_NUM_VPSS**: Defines the maximum number of cover regions for a VPSS channel.
- **COVEREX_MAX_NUM_VO**: Defines the maximum number of CoverEx regions for a VO channel.



- [COVEREX_MAX_NUM_VI](#): Defines the maximum number of CoverEx regions for a VI channel.
- [OVERLAYEX_MAX_NUM_VI](#): Defines the maximum number of OverlayEx regions for a VI channel.
- [RGN_HANDLE](#): Defines the region handle.
- [RGN_TYPE_E](#): Defines the region type.
- [RGN_AREA_TYPE_E](#): Defines the cover and CoverEx region type.
- [RGN_QUADRANGLE_S](#): Defines the quadrangle attribute.
- [INVERT_COLOR_MODE_E](#): Defines the OSD color inversion trigger mode.
- [OVERLAY_QP_INFO_S](#): Defines the QP attributes of an overlay region.
- [OVERLAY_INVERT_COLOR_S](#): Defines the attributes related to OSD color inversion.
- [VIDEO_FIELD_E](#): Defines the type of a frame/field to be overlaid with a region.
- [OVERLAY_ATTR_S](#): Defines the overlay region attribute structure of a GROUP channel.
- [OVERLAY_CHN_ATTR_S](#): Defines the channel display attributes of an extended overlay region of a GROUP channel.
- [COVER_CHN_ATTR_S](#): Defines the channel display attributes of a cover region of a VI channel.
- [COVEREX_CHN_ATTR_S](#): Defines the channel attributes of a CoverEx region.
- [OVERLAYEX_ATTR_S](#): Defines the attribute structure of an extended overlay region of a VI channel.
- [OVERLAYEX_CHN_ATTR_S](#): Defines the channel display attribute of an overlay region of a VI channel.
- [RGN_ATTR_U](#): Defines the region attribute union.
- [RGN_CHN_ATTR_U](#): Defines the display attributes of a region channel.
- [RGN_ATTR_S](#): Defines the region attribute structure.
- [RGN_CHN_ATTR_S](#): Defines the structure of the channel display attributes of a region.
- [RGN_CANVAS_INFO_S](#): Defines canvas information.

RGN_HANDLE_MAX

[Description]

Defines the maximum number of region handles.

[Syntax]

```
#define RGN_HANDLE_MAX 1024
```

[Member]

None

[Note]

None

[See Also]

None



RGN_COVER_MIN_X

[Description]

Defines the minimum value of the horizontal coordinate for the start position of a rectangular or quadrilateral Cover/CoverEx region.

[Syntax]

Chip	Macro Definition	
Hi3516A	#define RGN_COVER_MIN_X	0
Hi3518E V200	#define RGN_COVER_MIN_X	-8190
Hi3519 V100	#define RGN_COVER_MIN_X	-8190

[Member]

None

[Note]

None

[See Also]

None

RGN_COVER_MIN_Y

[Description]

Defines the minimum value of the vertical coordinate for the start position of a rectangular or quadrilateral Cover/CoverEx region.

[Syntax]

Chip	Macro Definition	
Hi3516A	#define RGN_COVER_MIN_Y	0
Hi3518E V200	#define RGN_COVER_MIN_Y	-8190
Hi3519 V100	#define RGN_COVER_MIN_Y	-8190

[Member]

None

[Note]

None

[See Also]

None



RGN_COVER_MAX_X

[Description]

Defines the maximum value of the horizontal coordinate for the start position of a rectangular or quadrilateral Cover/CoverEx region.

[Syntax]

Chip	Macro Definition	
Hi3516A	#define RGN_COVER_MAX_X	2592
Hi3518E V200	#define RGN_COVER_MAX_X	8190
Hi3519 V100	#define RGN_COVER_MAX_X	8190

[Member]

None

[Note]

None

[See Also]

None

RGN_COVER_MAX_Y

[Description]

Defines the maximum value of the vertical coordinate for the start position of a rectangular or quadrilateral Cover/CoverEx region.

[Syntax]

Chip	Macro Definition	
Hi3516A	#define RGN_COVER_MAX_Y	2592
Hi3518E V200	#define RGN_COVER_MAX_Y	8190
Hi3519 V100	#define RGN_COVER_MAX_Y	8190

[Member]

None

[Note]

None

[See Also]

None



RGN_COVER_MAX_WIDTH

[Description]

Defines the maximum width of a rectangular Cover/CoverEx region.

[Syntax]

Chip	Macro Definition	
Hi3516A	#define RGN_COVER_MAX_WIDTH	2592
Hi3518E V200	#define RGN_COVER_MAX_WIDTH	8190
Hi3519 V100	#define RGN_COVER_MAX_WIDTH	8190

[Member]

None

[Note]

None

[See Also]

None

RGN_COVER_MAX_HEIGHT

[Description]

Defines the maximum height of a rectangular Cover/CoverEx region.

[Syntax]

Chip	Macro Definition	
Hi3516A	#define RGN_COVER_MAX_HEIGHT	2592
Hi3518E V200	#define RGN_COVER_MAX_HEIGHT	8190
Hi3519 V100	#define RGN_COVER_MAX_HEIGHT	8190

[Member]

None

[Note]

None

[See Also]

None



RGN_OVERLAY_MIN_X

[Description]

Defines the minimum value of the horizontal coordinate for the start position of an Overlay/OverlayEx region.

[Syntax]

Chip	Macro Definition	
Hi3516A	#define RGN_OVERLAY_MIN_X	0
Hi3518E V200	#define RGN_OVERLAY_MIN_X	0
Hi3519 V100	#define RGN_OVERLAY_MIN_X	0

[Member]

None

[Note]

None

[See Also]

None

RGN_OVERLAY_MIN_Y

[Description]

Defines the minimum value of the vertical coordinate for the start position of an Overlay/OverlayEx region.

[Syntax]

Chip	Macro Definition	
Hi3516A	#define RGN_OVERLAY_MIN_Y	0
Hi3518E V200	#define RGN_OVERLAY_MIN_Y	0
Hi3519 V100	#define RGN_OVERLAY_MIN_Y	0

[Member]

None

[Note]

None

[See Also]

None



RGN_OVERLAY_MAX_X

[Description]

Defines the maximum value of the horizontal coordinate for the start position of an Overlay/OverlayEx region.

[Syntax]

Chip	Macro Definition	
Hi3516A	#define RGN_OVERLAY_MAX_X	2592
Hi3518E V200	#define RGN_OVERLAY_MAX_X	8190
Hi3519 V100	#define RGN_OVERLAY_MAX_X	8190

[Member]

None

[Note]

None

[See Also]

None

RGN_OVERLAY_MAX_Y

[Description]

Defines the maximum value of the vertical coordinate for the start position of an Overlay/OverlayEx region.

[Syntax]

Chip	Macro Definition	
Hi3516A	#define RGN_OVERLAY_MAX_Y	2592
Hi3518E V200	#define RGN_OVERLAY_MAX_Y	8190
Hi3519 V100	#define RGN_OVERLAY_MAX_Y	8190

[Member]

None

[Note]

None

[See Also]

None



RGN_OVERLAY_MAX_WIDTH

[Description]

Defines the maximum width of an Overlay/OverlayEx region.

[Syntax]

Chip	Macro Definition	
Hi3516A	#define RGN_OVERLAY_MAX_WIDTH	2592
Hi3518E V200	#define RGN_OVERLAY_MAX_WIDTH	4094
Hi3519 V100	#define RGN_OVERLAY_MAX_WIDTH	4094

[Member]

None

[Note]

None

[See Also]

None

RGN_OVERLAY_MAX_HEIGHT

[Description]

Defines the maximum height of an Overlay/OverlayEx region.

[Syntax]

Chip	Macro Definition	
Hi3516A	#define RGN_OVERLAY_MAX_HEIGHT	2592
Hi3518E V200	#define RGN_OVERLAY_MAX_HEIGHT	4094
Hi3519 V100	#define RGN_OVERLAY_MAX_HEIGHT	4094

[Member]

None

[Note]

None

[See Also]

None



OVERLAY_MAX_NUM_VENC

[Description]

Defines the maximum number of overlay regions for a VENC channel.

[Syntax]

```
#define OVERLAY_MAX_NUM_VENC      8
```

[Member]

None

[Note]

None

[See Also]

None

OVERLAY_MAX_NUM_VPSS

[Description]

Defines the maximum number of overlay regions for a VPSS channel.

[Syntax]

```
#define OVERLAY_MAX_NUM_VPSS      0
```

[Member]

None

[Note]

The VPSS does not support the Overlay region.

[See Also]

None

OVERLAYEX_MAX_NUM_VPSS

[Description]

Defines the maximum number of OverlayEx regions for a VPSS channel.

[Syntax]

```
#define OVERLAYEX_MAX_NUM_VPSS     8
```

[Member]

None

[Note]

None

[See Also]



None

OVERLAYEX_MAX_NUM_VO

[Description]

Defines the maximum number of OverlayEx regions for a VO channel.

[Syntax]

```
#define OVERLAYEX_MAX_NUM_VO      1
```

[Member]

None

[Note]

None

[See Also]

None

OVERLAYEX_MAX_NUM_PCIV

[Description]

Defines the maximum number of OverlayEx regions for a PCIV channel.

[Syntax]

```
OVERLAYEX_MAX_NUM_PCIV      1
```

[Member]

None

[Note]

Only the Hi3519 supports the PCIV function. The Hi3516A and Hi3518E V200 do not support the PCIV function.

[See Also]

None

COVER_MAX_NUM_VPSS

[Description]

Defines the maximum number of cover regions for a VPSS channel.

[Syntax]

```
#define COVER_MAX_NUM_VPSS      8
```

[Member]

None

[Note]



None

[See Also]

None

COVEREX_MAX_NUM_VPSS

[Description]

Defines the maximum number of CoverEx regions for a VPSS channel.

[Syntax]

```
#define COVEREX_MAX_NUM_VPSS      8
```

[Member]

None

[Note]

None

[See Also]

None

COVEREX_MAX_NUM_VO

[Description]

Defines the maximum number of CoverEx regions for a VO channel.

[Syntax]

```
#define COVEREX_MAX_NUM_VO      1
```

[Member]

None

[Note]

None

[See Also]

None

COVEREX_MAX_NUM_VI

[Description]

Defines the maximum number of CoverEx regions for a VI channel.

[Syntax]

```
#define COVEREX_MAX_NUM_VI      16
```

[Member]



None

[Note]

None

[See Also]

None

OVERLAYEX_MAX_NUM_VI

[Description]

Defines the maximum number of OverlayEx regions for a VI channel.

[Syntax]

```
#define OVERLAYEX_MAX_NUM_VI 16
```

[Member]

None

[Note]

None

[See Also]

None

RGN_HANDLE

[Description]

Defines the region handle.

[Syntax]

```
typedef HI_U32 RGN_HANDLE;
```

[Member]

Member	Description
RGN_HANDLE	Region handle.

[Note]

None

[See Also]

None

RGN_TYPE_E

[Description]



Defines the region type.

[Syntax]

```
typedef enum hiRGN_TYPE_E
{
    OVERLAY_RGN = 0,
    COVER_RGN,
    COVEREX_RGN,
    OVERLAYEX_RGN,
    RGN_BUTT
} RGN_TYPE_E;
```

[Member]

Member	Description
OVERLAY_RGN	Video overlay region of the VENC channel
COVER_RGN	Video cover region of the VI channel
COVEREX_RGN	Extended video cover region
OVERLAYEX_RGN	Extended video overlay region

[Note]

None

[See Also]

None

RGN_AREA_TYPE_E

[Description]

Defines the cover and CoverEx region type.

[Syntax]

```
typedef enum hiRGN_AREA_TYPE_E
{
    AREA_RECT = 0,
    AREA_QUAD_RANGLE,
    AREA_BUTT
} RGN_AREA_TYPE_E;
```

[Member]

Member	Description
AREA_RECT	Rectangular region



Member	Description
AREA_QUAD_RANGLE	Any quadrangular region. Only the convex quadrilateral is supported, and the concave quadrilateral is not supported.

[Note]

None

[See Also]

None

RGN_QUADRANGLE_S

[Description]

Defines the quadrangle attribute.

[Syntax]

```
typedef struct hiRGN_QUADRANGLE_S
{
    HI_BOOL bSolid;
    HI_U32 u32Thick;
    POINT_S stPoint[4];
} RGN_QUADRANGLE_S;
```

[Member]

Member	Description
bSolid	Solid/Dashed border flag
u32Thick	Border thickness, valid only for solid borders Value range: [2, 8]
stPoint[4]	Coordinates of the four points of any quadrangle

[Note]

- The four coordinates of a quadrilateral cannot overlap, that is, they cannot be on the same horizontal or vertical line.
- Only the convex quadrilateral is supported. If the four coordinate points form a concave quadrilateral, a triangle is displayed.

[See Also]

None

INVERT_COLOR_MODE_E

[Description]



Defines the OSD color inversion trigger mode.

[Syntax]

```
typedef enum hiINVERT_COLOR_MODE_E
{
    WHITE_TO_BLACK = 0,
    BLACK_TO_WHITE,
    INVERT_COLOR_BUTT
} INVERT_COLOR_MODE_E;
```

[Member]

Member	Description
LESSTHAN_LUM_THRESH	Color inversion is triggered when the average video background luminance is less than the luminance threshold
MORETHAN_LUM_THRESH	Color inversion is triggered when the average video background luminance is greater than the luminance threshold

[Note]

This configuration is valid when color inversion is enabled.

[See Also]

None

OVERLAY_QP_INFO_S

[Description]

Defines the QP attributes of an overlay region.

[Syntax]

```
typedef struct hiOVERLAY_QP_INFO_S
{
    HI_BOOL      bAbsQp;
    HI_S32       s32Qp;
    HI_BOOL      bQpDisable;
} OVERLAY_QP_INFO_S;
```

[Member]

Member	Description
bAbsQp	Absolute QP or not



Member	Description
s32Qp	QP value When bAbsQp is HI_TRUE, the QP value range from 0 to 51. When bAbsQp is HI_FALSE, the QP value range from -51 to +51.
bQpDisable	QP protection disable for an overlay region bQpDisable must be set to HI_FALSE because the Hi3516A does not support this function.

[Note]

This data structure is used for controlling the bit rate during encoding. If you are not familiar with the data structure, you are advised to set all parameters to **0**.

[See Also]

None

OVERLAY_INVERT_COLOR_S

[Description]

Defines the attributes related to OSD color inversion.

[Syntax]

```
typedef struct hiOVERLAY_INVERT_COLOR_S
{
    SIZE_S stInvColArea;
    HI_U32 u32LumThresh;
    INVERT_COLOR_MODE_E enChgMod;
    HI_BOOL bInvColEn;
} OVERLAY_INVERT_COLOR_S;
```

[Member]

Member	Description
stInvColArea	Color inversion area (basic color inversion unit) Value range: Height: [16, 64], 16-pixel alignment Width: [16, 64], 16-pixel alignment
u32LumThresh	Luminance threshold Value range: [0, 255]
enChgMod	OSD color inversion trigger mode



Member	Description
bInvColEn	OSD color inversion enable HI_TRUE: enabled HI_FALSE: disabled

[Note]

- As the basic color inversion unit, the color of each color inversion area is inverted as a whole part. Therefore, you need to set the color inversion area based on the actual size of the OSD area.
- The color inversion area, luminance threshold, and color inversion trigger mode are public attributes. They are valid for the OSD areas whose color inversion function is enabled. If the public attributes are configured differently each time, the attributes are subject to the latest configurations.
- The **bInvColEn** parameter controls whether to enable color inversion of an OSD area. The color inversion function of each OSD area can be controlled separately.
- When the color inversion function of an OSD area is enabled, the start position and width and height of the OSD area must be 16-pixel-aligned.

[See Also]

None

VIDEO_FIELD_E

[Description]

Defines the type of a frame/field to be overlaid with a region.

[Syntax]

```
typedef enum hiVIDEO_FIELD_E
{
    VIDEO_FIELD_TOP      = 0x1,    /* even field */
    VIDEO_FIELD_BOTTOM   = 0x2,    /* odd field */
    VIDEO_FIELD_INTERLACED = 0x3,   /* two interlaced fields */
    VIDEO_FIELD_FRAME    = 0x4,    /* frame */
    VIDEO_FIELD_BUTT
} VIDEO_FIELD_E;
```

[Member]

Member	Description
VIDEO_FIELD_TOP	The top field is overlaid with a region.
VIDEO_FIELD_BOTTOM	The bottom field is overlaid with a region.
VIDEO_FIELD_INTERLACED	The interlaced picture is overlaid with a region.
VIDEO_FIELD_FRAME	The entire frame is overlaid with a region.



[Note]

RGN_ATTACH_FIELD_TOP or RGN_ATTACH_FIELD_BOTTOM is valid only for interlaced 2-field pictures.

[See Also]

None

OVERLAY_ATTR_S

[Description]

Defines the overlay region attribute structure of a VENC channel.

[Syntax]

```
typedef struct hiOVERLAY_ATTR_S
{
    PIXEL_FORMAT_E enPixelFormat;
    HI_U32 u32BgColor;
    SIZE_S stSize;
}OVERLAY_ATTR_S;
```

[Member]

Member	Description
enPixelFormat	Pixel format When an overlay region is attached to a VENC channel, the pixel format can be PIXEL_FORMAT_RGB_1555 or PIXEL_FORMAT_RGB_4444.
u32BgColor	Region background color (dynamic attribute) Value range: none
stSize	Region height and region width When an overlay region is attached to a VENC channel, the value ranges of the width and height are as follows: Width: [2, RGN_OVERLAY_MAX_WIDTH], 2-pixel-aligned Height: [2, RGN_OVERLAY_MAX_HEIGHT], 2-pixel-aligned

[Note]

- **enPixelFormat** and **stSize** are static variables only after [HI_MPI_RGN_AttachToChn\(\)](#) is called and before [HI_MPI_RGN_DetachFromChn](#) is called.
- The width and height of the Overlay region is related to the size of the memory allocated for the region. It is recommended that region width and height be configured based on actual requirements to avoid memory waste.

[See Also]



None

OVERLAY_CHN_ATTR_S

[Description]

Defines the channel display attributes of an overlay region of a VENC channel.

[Syntax]

```
typedef struct hiOVERLAY_CHN_ATTR_S
{
    POINT_S stPoint;
    HI_U32 u32FgAlpha;
    HI_U32 u32BgAlpha;
    HI_U32 u32Layer;
    OVERLAY_QP_INFO_S stQpInfo;
    OVERLAY_INVERT_COLOR_S stInvertColor;
}OVERLAY_CHN_ATTR_S;
```

[Member]

The following describes the overlay region for the VENC channel:

Member	Description
stPoint	Region position (dynamic attribute). The origin is the upper left point of the region. Value range: Horizontal coordinate: [0, 8190], 2-pixel alignment Vertical coordinate: [0, 8190], 2-pixel alignment
u32FgAlpha	Transparency of the pixel whose alpha bit is 1 (dynamic attribute). u32FgAlpha is called foreground alpha. Value range: [0, 128] A smaller value indicates more transparent pixel.
u32BgAlpha	Transparency of the pixel whose alpha bit is 0 (dynamic attribute). u32BgAlpha is called background alpha. Value range: [0, 128]. A smaller value indicates more transparent pixel.
u32Layer	Region level (dynamic attribute) Value range: [0, 7] A larger value indicates a higher region level.
stQpInfo	QP value when the region is encoded (dynamic attribute) For details about the value range, see the description of OVERLAY_QP_INFO_S .

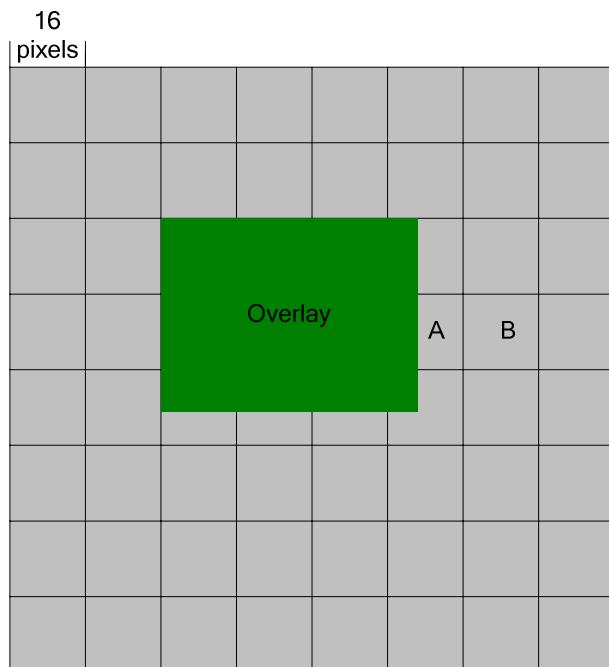


Member	Description
stInvertColor	Region color inversion configuration information (dynamic attribute) For details about the value range, see the description of OVERLAY_INVERT_COLOR_S .

[Note]

You are advised to set the start position, width, and height of an overlay region to 16-pixel aligned. As shown in [Figure 8-3](#), the width and height of the overlay region are not 16-pixel aligned. As a result, a macroblock may have two QPs. For example, macroblock A has the overlay region QP and the calculated QP. However, only one QP is allowed for each macroblock. In this case, the encoder selects the region QP for macroblock A. If the difference between the QPs of macroblock A and macroblock B is great, the picture quality of the macroblocks around the overlay region is inconsistent.

Figure 8-3 Non-16-pixel aligned overlay region



- When the region memory information is in PIXEL_FORMAT_RGB_1555 format, the Hi3516A expands the alpha value. When the alpha bit is 1, the Hi3516A performs transparency overlay by setting u32FgAlpha; when the alpha bit is 0, the Hi3516A performs transparency overlay by setting u32BgAlpha.
- The value 0 indicates 100% transparency, and the value 128 indicates opacity.
- When multiple Overlay regions are overlaid to the VENC, the pixel formats must be the same. Otherwise, the Overlay regions do not take effect.

[See Also]

None



COVER_CHN_ATTR_S

[Description]

Defines the channel display attributes of a cover region of a VI channel.

[Syntax]

```
typedef struct hiCOVER_CHN_ATTR_S
{
    RGN_AREA_TYPE_E      enCoverType;
    union
    {
        RECT_S           stRect;
        RGN_QUADRANGLE_S stQuadRangle;
    };
    HI_U32   u32Color;
    HI_U32   u32Layer;
} COVER_CHN_ATTR_S;
```

[Member]

Member	Description
enCoverType	Cover and CoverEx region type
stRect	Region position, width, and height (dynamic attribute) Value range of the position: Horizontal coordinate: [RGN_COVER_MIN_X, RGN_COVER_MAX_X], 2-pixel alignment Vertical coordinate: [RGN_COVER_MIN_Y, RGN_COVER_MAX_Y], 2-pixel alignment Value range of the width and height: Width: [2, RGN_COVER_MAX_WIDTH], 2-pixel-aligned Height: [2, RGN_COVER_MAX_HEIGHT], 2-pixel-aligned
stQuadRangle	Value range of the points of any quadrangle: Horizontal coordinate: [RGN_COVER_MIN_X, RGN_COVER_MAX_X], 2-pixel-aligned Vertical coordinate: [RGN_COVER_MIN_Y, RGN_COVER_MAX_Y], 2-pixel-aligned Value range of the thickness of the dashed borders: [2, 8], 2-pixel-aligned
u32Color	Region color (dynamic attribute)
u32Layer	Region level (dynamic attribute) The value range is related to the maximum number of cover regions supported by each module, that is, the value range is [0, Maximum number of cover regions - 1].



[Note]

- The region color data format is PIXEL_FORMAT_RGB_888.
- For Hi3518E V200 and Hi3519 V100, part or all of the Cover region can exceed the target picture. The Cover region takes effect when the region overlaps with the target picture, and does not take effect when there is no overlapping.

[See Also]

None

COVEREX_CHN_ATTR_S

[Description]

Defines the channel attributes of a CoverEx region.

[Syntax]

```
typedef struct hiCOVEREX_CHN_ATTR_S
{
    RGN_AREA_TYPE_E      enCoverType;
    union
    {
        RECT_S          stRect;
        RGN_QUADRANGLE_S stQuadRange;
    };
    HI_U32   u32Color;
    HI_U32   u32Layer;
} COVEREX_CHN_ATTR_S;
```

[Member]

Member	Description
enCoverType	Cover and CoverEx region type
stRect	Region position, width, and height (dynamic attribute) Value range of the position: Horizontal coordinate: [RGN_COVER_MIN_X, RGN_COVER_MAX_X], 2-pixel-aligned Vertical coordinate: [RGN_COVER_MIN_Y, RGN_COVER_MAX_Y], 2-pixel-aligned Value range of the width and height: Width: [2, RGN_COVER_MAX_WIDTH], 2-pixel-aligned Height: [2, RGN_COVER_MAX_HEIGHT], 2-pixel-aligned
stQuadRange	Value range of the points of any quadrangle: Horizontal coordinate: [RGN_COVER_MIN_X, RGN_COVER_MAX_X], 2-pixel-aligned



Member	Description
	Vertical coordinate: [RGN_COVER_MIN_Y, RGN_COVER_MAX_Y], 2-pixel-aligned Value range of the thickness of the dashed borders: [2, 8], 2-pixel-aligned
u32Color	Region color (dynamic attribute)
u32Layer	Region level (dynamic attribute) The value range is related to the maximum number of CoverEx regions supported by each module, that is, the value range is [0, Maximum number of CoverEx regions - 1]. A larger value indicates a higher region level.

[Note]

- The color data format for the region is PIXEL_FORMAT_RGB_888.
- For Hi3518E V200 and Hi3519 V100, part or all of the CoverEx region can exceed the target picture. The CoverEx region takes effect when the region overlaps with the target picture, and does not take effect when there is no overlapping.

[See Also]

None

OVERLAYEX_ATTR_S

[Description]

Defines the attribute structure of an extended overlay region of a VI channel.

[Syntax]

```
typedef struct hiOVERLAYEX_COMM_ATTR_S
{
    PIXEL_FORMAT_E enPixelFormat;
    HI_U32 u32BgColor;
    SIZE_S stSize;
} OVERLAYEX_ATTR_S;
```

[Member]

Member	Description
enPixelFormat	Pixel format Value range: PIXEL_FORMAT_RGB_1555. PIXEL_FORMAT_RGB_4444. PIXEL_FORMAT_RGB_8888.
u32BgColor	Region background color (dynamic attribute) Value range: none



Member	Description
stSize	Region width and height Value range: Width: [2, RGN_OVERLAY_MAX_WIDTH], 2-pixel-aligned Height: [2, RGN_OVERLAY_MAX_HEIGHT], 2-pixel-aligned

[Note]

- **enPixelFmt** and **stSize** are static variables only after [HI_MPI_RGN_AttachToChn\(\)](#) is called and before [HI_MPI_RGN_DetachFromChn](#) is called.
- The width and height and the OverlayEx region is related to the size of the memory allocated for the region. It is recommended that the region width and height be configured as required to avoid memory waste.

[See Also]

None

OVERLAYEX_CHN_ATTR_S

[Description]

Defines the channel display attribute of an overlay region of a GROUP channel.

[Syntax]

```
typedef struct hiOVERLAYEX_CHN_ATTR_S
{
    POINT_S stPoint;
    HI_U32 u32FgAlpha;
    HI_U32 u32BgAlpha;
    HI_U32 u32Layer;
}OVERLAYEX_CHN_ATTR_S;
```

[Member]

Member	Description
stPoint	Region position Value range of the OverlayEx region position: Horizontal coordinate: [RGN_OVERLAY_MIN_X, RGN_OVERLAY_MAX_X], 2-pixel-aligned Horizontal coordinate: [RGN_OVERLAY_MIN_Y, RGN_OVERLAY_MAX_Y], 2-pixel-aligned Dynamic attribute



Member	Description
u32FgAlpha	Transparency of the pixel whose alpha bit is 1 (dynamic attribute). u32FgAlpha is called foreground alpha. Value range: [0, 255] A smaller value indicates more transparent pixel.
u32BgAlpha	Transparency of the pixel whose alpha bit is 0 (dynamic attribute). u32BgAlpha is called background alpha. Value range: [0, 255] A smaller value indicates more transparent pixel.
u32Layer	Region level (dynamic attribute) The value range is related to the maximum number of OverlayEx regions supported by each module, that is, the value range is [0, Maximum number of OverlayEx regions – 1]. A larger value indicates a higher region level.

[Note]

- When the region memory information is in PIXEL_FORMAT_RGB_1555 format, the Hi3516A extends the alpha value. When the alpha bit is **1**, the Hi3516A overlays the transparency by setting **u32FgAlpha**; when the alpha bit is **0**, the Hi3516A overlays the transparency by setting **u32BgAlpha**.
- The value **0** indicates 100% transparency, and the value **255** indicates opacity.

[See Also]

None

RGN_ATTR_U

[Description]

Defines the region attribute union.

[Syntax]

```
typedef union hiRGN_ATTR_U
{
    OVERLAY_ATTR_S      stOverlay;
    OVERLAYEX_ATTR_S   stOverlayEx;
} RGN_ATTR_U;
```

[Member]

Member	Description
stOverlay	Overlay region attributes of the VENC channel.
stOverlayEx	Extended overlay region attributes.



[Note]

As the cover region and CoverEx region do not have region attributes, this union does not contain related members.

[See Also]

None

RGN_CHN_ATTR_U

[Description]

Defines the display attributes of a region channel.

[Syntax]

```
typedef union hiRGN_CHN_ATTR_U
{
    OVERLAY_CHN_ATTR_S      stOverlayChn;
    COVER_CHN_ATTR_S        stCoverChn;
    COVEREX_CHN_ATTR_S     stCoverExChn;
    OVERLAYEX_CHN_ATTR_S   stOverlayExChn;
} RGN_CHN_ATTR_U;
```

[Member]

Member	Description
stOverlayChn	Display attributes of an overlay channel of the VENC channel
stCoverChn	Display attributes of a cover region
stCoverExChn	Channel display attributes of the extended cover region
stOverlayExChn	Channel display attributes of the extended overlay region

[Note]

None

[See Also]

None

RGN_ATTR_S

[Description]

Defines the region attribute structure.

[Syntax]

```
typedef struct hiRGN_ATTR_S
{
    RGN_TYPE_E enType;
```



```
RGN_ATTR_U unAttr;  
} RGN_ATTR_S;
```

[Member]

Member	Description
enType	Region type
unAttr	Region attribute

[Note]

None

[See Also]

None

RGN_CHN_ATTR_S

[Description]

Defines the structure of the channel display attributes of a region.

[Syntax]

```
typedef struct hiRGN_CHN_ATTR_S  
{  
    HI_BOOL          bShow;  
    RGN_TYPE_E      enType;  
    RGN_CHN_ATTR_U  unChnAttr;  
} RGN_CHN_ATTR_S;
```

[Member]

Member	Description
bShow	Whether to display a region (dynamic attribute) Value: HI_TRUE or HI_FALSE
enType	Region type (static attribute)
unChnAttr	Channel display attributes of a region

[Note]

None

[See Also]

None



RGN_CANVAS_INFO_S

[Description]

Defines canvas information.

[Syntax]

```
typedef struct hIRGN_CANVAS_INFO_S
{
    HI_U32          u32PhyAddr;
    HI_U32          u32VirtAddr;
    SIZE_S          stSize;
    HI_U32          u32Stride;
    PIXEL_FORMAT_E enPixelFmt;
} RGN_CANVAS_INFO_S;
```

[Member]

Member	Description
u32PhyAddr	Canvas physical address
u32VirtAddr	Canvas virtual address
stSize	Canvas size
u32Stride	Canvas stride, which should be 64-byte-aligned for Hi3516A, Hi3518E V200, and Hi3519 V100
enPixelFmt	Canvas pixel format

[Note]

None

[See Also]

None

8.5 Error Codes

Table 8-5 describes the error codes for region management APIs.

Table 8-5 Error codes for region management APIs

Error Code	Macro Definition	Description
0xA0038001	HI_ERR_RGN_INVALID_DEVID	The device ID exceeds the valid range.
0xA0038002	HI_ERR_RGN_INVALID_CHNID	The channel ID is incorrect or the region handle is invalid.



Error Code	Macro Definition	Description
0xA0038003	HI_ERR_RGN_ILLEGAL_PARA_M	The parameter value exceeds its valid range.
0xA0038004	HI_ERR_RGN_EXIST	The device, channel, or resource to be created already exists.
0xA0038005	HI_ERR_RGN_UNEXIST	The device, channel, or resource to be used or destroyed does not exist.
0xA0038006	HI_ERR_RGN_NULL_PTR	The pointer is null.
0xA0038007	HI_ERR_RGN_NOT_CONFIG	The module is not configured.
0xA0038008	HI_ERR_RGN_NOT_SUPPORT	The parameter or function is not supported.
0xA0038009	HI_ERR_RGN_NOT_PERM	The operation, for example, attempting to modify the value of a static parameter, is forbidden.
0xA003800C	HI_ERR_RGN_NOMEM	The memory fails to be allocated due to some causes such as insufficient system memory.
0xA003800D	HI_ERR_RGN_NOBUF	The buffer fails to be allocated due to some causes such as oversize of the data buffer applied for.
0xA003800E	HI_ERR_RGN_BUF_EMPTY	The buffer is empty.
0xA003800F	HI_ERR_RGN_BUF_FULL	The buffer is full.
0xA0038010	HI_ERR_RGN_NOTREADY	The system is not initialized or the corresponding module is not loaded.
0xA0038011	HI_ERR_RGN_BADADDR	The address is invalid.
0xA0038012	HI_ERR_RGN_BUSY	The system is busy.



Contents

9 Audio	9-1
9.1 Overview	9-1
9.2 Functions	9-1
9.2.1 AI and AO	9-1
9.2.2 Audio Encoding and Decoding	9-13
9.2.3 Mapping Between Audio Interfaces and Devices	9-16
9.2.4 Embedded Audio Codec.....	9-17
9.3 API Reference	9-21
9.3.1 AI	9-21
9.3.2 AO	9-53
9.3.3 AENC.....	9-79
9.3.4 ADEC.....	9-91
9.3.5 Embedded audio Codec	9-101
9.4 Data Structures	9-156
9.4.1 AI and AO	9-156
9.4.2 AENC.....	9-201
9.4.3 ADEC.....	9-205
9.4.4 Embedded audio Codec	9-211
9.5 Error Codes	9-214



Figures

Figure 9-1 Schematic diagram of audio recording and playing.....	9-2
Figure 9-2 1/2/4/8/16-channel multiplexing of I ² S.....	9-3
Figure 9-3 PCM TX timing	9-4
Figure 9-4 UpVQE processing flowchart.....	9-9
Figure 9-5 DnVQE processing flowchart.....	9-9
Figure 9-6 Echo cancellation diagram.....	9-10
Figure 9-7 Mapping between AIPs/AOPs and AI/AO devices	9-16



Tables

Table 9-1 Maximum number of AI and AO channels supported by AIO interfaces	9-4
Table 9-2 Libraries of the VQE modules	9-6
Table 9-3 VQE, Hi-Fi VQE, and Talk VQE supported by various chips.....	9-8
Table 9-4 Audio encoding and decoding protocols.....	9-14
Table 9-5 Structure of the HiSilicon voice frame	9-16
Table 9-6 Ranges of AI and AO device IDs.....	9-17
Table 9-7 Parameters of the ioctl function.....	9-18
Table 9-8 Default parameter configurations in various scenario modes of VQE.....	9-187
Table 9-9 Default parameter configurations in various scenario modes of Hi-Fi VQE	9-189
Table 9-10 Default parameter configurations in various scenario modes of Talk VQE.....	9-191
Table 9-11 Default parameter configurations in various scenario modes of DnVQE.....	9-193
Table 9-12 Error codes for the AI MPIs	9-214
Table 9-13 Error codes for the AO MPIs	9-215
Table 9-14 Error codes for the AENC MPIs	9-216
Table 9-15 Error codes for the ADEC MPIs	9-217



9 Audio

9.1 Overview

The audio module consists of the audio input unit (AIU), audio output unit (AOU), audio encoding (AENC) module, and audio decoding (ADEC) module. The AIU and AOU control the audio interfaces to implement the audio input (AI) and audio output (AO) functions. The AENC module and ADEC module encode and decode G711, G726, or ADPCM streams, and record and play LPCM audio files.

9.2 Functions

9.2.1 AI and AO

9.2.1.1 Audio Interfaces, AI Devices, and AO Devices

Audio input/output (AIO) interfaces are used to connect the chip to the audio coder/decoder (codec) to implement audio recording and playback. The AIO interfaces are classified into audio input ports (AIPs) and audio output ports (AOPs).

From the aspect of software, the units for implementing abstract input functions of audio interfaces are called AI devices, and the units for implementing abstract output functions of audio interfaces are called AO devices.

Software maps each input or output interface to the AI or AO device based on interface functions. For example, AIP0 supports only audio inputs and maps to AiDev0. AOP0 supports only audio outputs and maps to AoDev0.

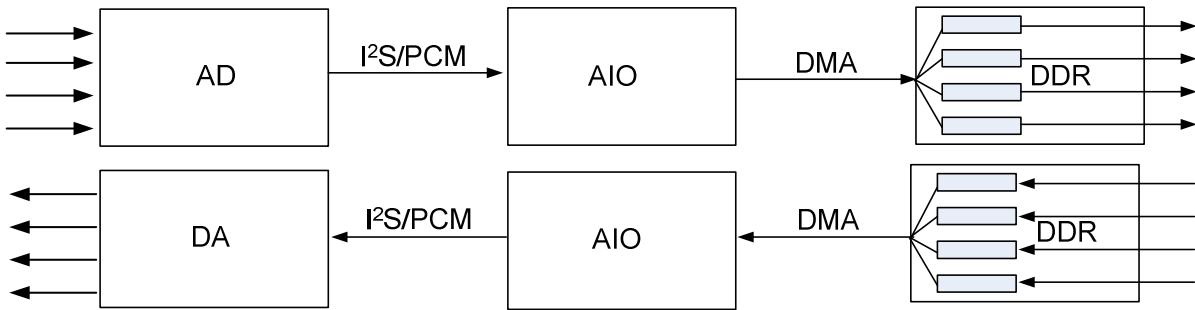
9.2.1.2 Recording and Playing Principles

The raw audio signals are analog signals. After processed by the audio codec based on the sampling rate and sampling precision, the analog signals are converted into digital signals. The audio codec transmits the digital signals to the AI device using the I²S or PCM timing. The chip transmits the audio data from the AI device to the memory in DMA mode to implement recording.

The principle of playing audio data is the same as that of recording audio data. The chip transmits the audio data from the memory to the AO device in DMA mode. The AO device transmits data to the audio codec using the I²S or PCM timing. The audio codec converts digital signals into analog signals and outputs the analog signals.



Figure 9-1 Schematic diagram of audio recording and playing



9.2.1.3 Audio Interface Timing and AI and AO Channel Arrangement

Audio Interface Timing

Audio interfaces support the standard I²S interface timing and PCM interface timing, and provide various configurations to interconnect with audio codecs complying with multiple specifications. For details about timings, see chapter 13 "Audio Interfaces" in the *related data sheet*.

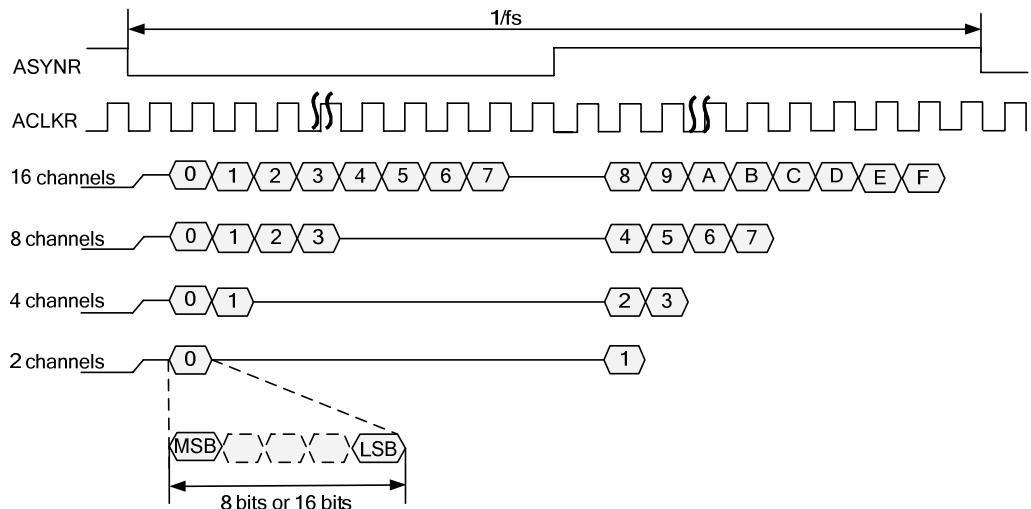
To ensure component connection, understand the I²S or PCM protocol and timing capability of the interconnected audio codecs. The following describes features of the Hi35xx I²S and PCM:

- Based on the standard I²S or PCM protocol, the most significant bit (MSB) is always prior to the least significant bit (LSB) in transmission. That is, serial data is transmitted from the MSB to the LSB.
- The AI device supports multi-channel I²S and PCM RX timings. When the AI device connects to the audio codec, the timing mode, sync clock, and sampling bit width of the audio codec must be consistent with those configured in the AI device. Otherwise, the collected data may be incorrect.
- The AI and AO devices support master and slave modes. In mast mode, the AI or AO device provides clocks. If both the AI and AO device connect to the same codec, the input and output clocks are shared. There is no such a limitation in other cases. In slave mode, the external audio codec provides independent input and output clocks.
- When the AI or AO device works in slave mode, you are advised to configure the interconnected codec and then the AI or AO device. When the AI or AO device works in master mode, you are advised to configure the AI or AO device and then the interconnected codec. The recommendations are provided based on timings.
- In master mode, some AI and AO devices provide only the frame sync clock and bit stream clock for timing synchronization but not the main clock (MCLK). In this case, if the audio codec uses the external crystal oscillator clock as the working clock, the sound may be distorted. Therefore, you are advised to select the slave mode or use the clock generated by the bit stream clock as the working clock of the audio codec.
- If the AI or AO device works in master mode and provides the MCLK, the MCLK is configured as follows:
 - When the sampling rate is 96 kHz, 48 kHz, 24 kHz, or 12 kHz, a 12.288 MHz MCLK is provided.



- When the sampling rate is 32 kHz, 16 kHz, or 8 kHz, the bit width for the 32 kHz sampling rate is not 256 bits, and the bit width for the 8 kHz sampling rate is not 16 bits, a 12.288 MHz MCLK is provided.
- When the sampling rate is 64 kHz, 32 kHz, 16 kHz, or 8 kHz, the bit width for the 32 kHz sampling rate is 256 bits, or the bit width for the 8 kHz sampling rate is 16 bits, an 8.192 MHz MCLK is provided.
- When the sampling rate is 44.1 kHz, 22.05 kHz, or 11.025 kHz, an 11.2896 MHz MCLK is provided.
- The AI or AO device supports standard and customized PCM modes. Only the mono mode is supported in PCM mode. Therefore, the output of the audio-right channel of the connected codec must be disabled in PCM mode; otherwise, noises occur. According to the PCM standard mode protocol, the AI and AO data is delayed by one BCLK cycle relative to the frame sync signal when the standard PCM master mode or standard PCM slave mode is selected. If an external codec is connected in standard PCM master mode or standard slave PCM mode, ensure that the number of delay BCLK cycles of the external codec data relative to the frame sync signal must be greater than number of delay clock cycles of AI and AO data relative to the frame sync signal (one BCLK cycle). Otherwise, noises occur.

Figure 9-2 1/2/4/8/16-channel multiplexing of I²S



AI and AO Channels

AI and AO channels are concepts from the aspect of software. The following describes the meanings of AI and AO channels. For example, when an AI device uses the I²S timing supporting multi-channel multiplexing to receive data, the standard I²S protocol supports only an audio-left channel and an audio-right channel. In this case, the AI device receives at most 128-bit audio data on the audio-left channel and the audio-right channel respectively. Assume that the audio codec has the multiplexing function. The 16 channels of 16-bit audio data are multiplexed into 2 channels of 128-bit data, one transmitted to the audio-left channel and the other transmitted to the audio-right channel. Then the AI device parses the 16-channel audio data. The 16 channels are called 16 audio channels.

The multiplexing modes supported by AIO interfaces vary according to the protocol. See and [Table 9-1](#).



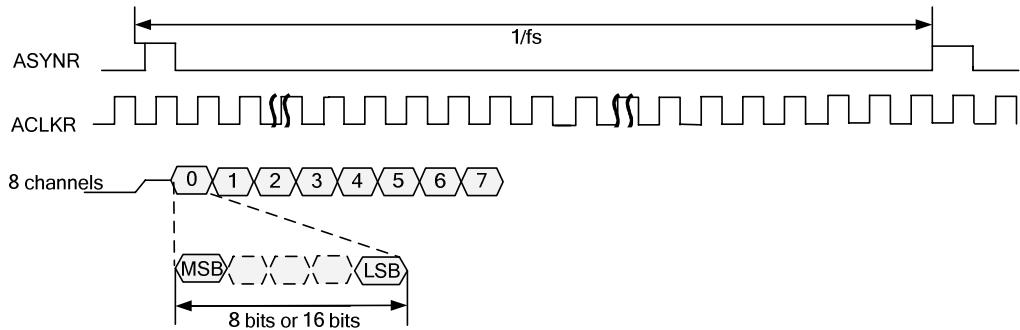
Table 9-1 Maximum number of AI and AO channels supported by AIO interfaces

AIO Multiplexing	Maximum Audio Data	Channel Instance
I ² S RX timing	128 bits on the audio-left channel and audio-right channel respectively	Sixteen 16-bit channels
PCM RX timing	128 bits on a mono channel	Eight 16-bit channels
I ² S TX timing	16 bits on the audio-left channel and audio-right channel respectively	Two 16-bit channels
PCM TX timing	16 bits on a mono channel	One 16-bit channel

Based on the sampling precision and timing, the AI channel and the AO channel can be split into AiChn0, AiChn1, ..., and AiChnN or AoChn0, AoChn1, ..., AoChnN respectively when the data amount does not exceed the threshold for the AI or AO device. Audio data can be correctly received or sent when the I²S or PCM timing configured on the AI or AO device is consistent with that configured in the audio codec. The process of unifying the timings of the AI/AO device and the audio codec is called interconnection.

Assume that the standard PCM slave mode, 8 kHz sampling rate, 16-bit sampling precision, and four channels are configured for the AI, and 8 kHz frame sync clock, 8 kHz x 64 bit stream clock, and 64-bit (the audio codec can multiplex four 16-bit sampling points as one sampling point) sampling point are configured for the external codec. In this case, the channel arrangement for the PCM timing is shown in [Figure 9-3](#).

Figure 9-3 PCM TX timing



The Hi35xx supports a maximum of 16 AI channels, two AO channels in I²S mode, and one AO channel in PCM mode. When the embedded codec is connected, the Hi35xx supports only two AI channels and two AO channels due to the limitations exerted by the embedded audio codec. When you configure the AI and AO devices, ensure that the number of channels is an even number or 1. When there are multiple channels, the channels whose channel IDs are smaller than half of the number of channels are audio-left channels, and the other channels are audio-right channels. Two channels whose channel ID difference is half of the number of channels are a stereo channel pair. As shown in [Figure 9-3](#), channel 0 and channel 4, channel 1 and channel 5, channel 2 and channel 6, and channel 3 and channel 7 are four stereo channel pairs, and 4-channel 16-bit stereo inputs are supported. In this case, you need to operate only audio-left channels 0–3.



The AI and AO devices for AIO interfaces are independent. If the AI and AO device connects to the same codec, their working modes must be the same. For example, the working modes of both the AI and AO devices are set to I²S timing slave mode. When the AIO interface works in master mode, the product of the number of channels multiplied by the sampling precision for the AI device must be the same as that for the AO device. This ensures that the sync clock transmitted from the AIO interface to the audio codec is the same for the AI and AO devices. If the AI and AO devices do not connect to the same codec, there are no such limitations. When the AIO interface works in slave mode, the product of the number of channels multiplied by the sampling precision for the AI device can be different from that for the AO device. The audio codec can transmit different sync clocks to the AI and AO devices. The Hi35xx SDK allows you to bind AI and AO channels by calling the binding interfaces of the SYS module, implementing real-time playback of audio data.

9.2.1.4 Resampling

The Hi35xx AIU and AOU can resample audio data. If [HI_MPI_AI_GetFrame](#) is called when AI resampling sampling is enabled, resampling is internally performed and then the processed data is obtained. If AO resampling is enabled, audio data is resampled before it is transmitted to the AO channel for playing.

Audio resampling at any two sampling rates (excluding the 64 kHz and 96 kHz sampling rates) is supported. The supported input or output sampling rates for resampling include 8 kHz, 11.025 kHz, 12 kHz, 16 kHz, 22.05 kHz, 24 kHz, 32 kHz, 44.1 kHz, and 48 kHz. The 64 kHz and 96 kHz input/output sampling rates are not supported. Resampling is available only for the mono channel.

- During AI resampling, the input sampling rate is the same as the sampling rate configured in the AI device attributes, and the output sampling rate must be different from the sampling rate configured in the AI device attributes. You need only to configure the output sampling rate. The number of sampling points in each frame before resampling is the same as that configured in the AI device attributes.
- During AO resampling, the output sampling rate is the same as the sampling rate configured in the AO device attributes, and the input sampling rate must be different from the sampling rate configured in the AO device attributes. You need only to configure the input sampling rate. The number of sampling points in each frame after resampling is the same as that configured in the AO device attributes.
- If the AI data needs to be sent to the AENC for encoding and AI resampling is enabled, the audio frame length after resampling (**Out PtNumPerFrm**) must be less than or equal to **u32PtNumPerFrm** in the encoding channel attribute during CPU software encoding, and be 80, 160, 240, 320, or 480 during VOIE encoding. The conversion relationship between **Out PtNumPerFrm**, **InPtNumPerFrm** (audio frame length before resampling), **InSampleRate** (sampling rate before resampling), and **OutSampleRate** (sampling rate after resampling) is as follows: OutPtNumPerFrm = OutSampleRate x InPtNumPerFrm/InSampleRate

The Hi3516A/Hi3518E V200/Hi3519 V100 does not support VOIE encoding.



CAUTION

- When an AI channel is bound to an AO channel by calling the system binding interface and data is transferred between the AI channel and AO channel, AI or AO resampling is invalid.
- When an AI channel is bound to an AENC channel by calling the system binding interface and data is transferred between the AI channel and AENC channel, AI resampling is valid.
- In non-system binding mode, you can call [HI_MPI_AI_GetFrame](#) to obtain AI frames after resampling and transmit the frames to the AENC or AO channel. In this way, data is transferred between the AI and AENC channels or AI and AO channels, and AI or AO resampling is valid.
- The preceding limits are not applicable to the data transfer between an ADEC channel and an AO channel. That is, when an ADEC channel is bound to an AO channel by calling the system binding interface, AO resampling is still valid.
- Calling [HI_MPI_AI_EnableReSmp](#) and [HI_MPI_AI_DisableReSmp](#) enables and disables the resampling (RES) module of the voice quality enhancement (VQE) respectively.

9.2.1.5 VQE

The Hi35xx AIU and AOU support the VQE function. During VQE processing, the scheduling logics UpVQE and DnVQE are used to process data in the AI channel and AO channel respectively based on the similarities and differences of the AI channel and AO channel.

- The VQE function of the AI uplink channel is provided by the acoustic echo cancellation (AEC), audio noise reduction (ANR), automatic gain control (AGC), high-pass filtering (HPS), recording noise reduction (RNR), equalizer (EQ), dynamic range control (DRC), parameter equalizer (PEQ), and high dynamic range (HDR), as shown in [Figure 9-4](#). The VQE function of the AI uplink channel is implemented in [HI_MPI_AI_GetFrame](#), and the processed data is returned.
- The VQE function of the AO downlink channel is provided by the ANR, AGC, HPF, and EQ modules, as shown in [Figure 9-5](#). The VQE function of the AO downlink channel is implemented in [HI_MPI_AO_SendFrame](#).

The VQE consists of two scheduling interface modules (UpVQE and DnVQE), eleven functional modules (AEC, ANR, AGC, RNR, EQ, HPF, gain, RES, HDR, DRC, and PEQ), and one common module. The two scheduling logics dynamically load the functional modules by calling the dlopen function of the [libdl](#) library, and schedule the functional modules in a unified manner by unifying the API formats of the functional modules. The VQE functional modules can be cropped. You can choose the dynamic library files of the used functional modules based on the actual application scenario. See [Table 9-2](#).

Table 9-2 Libraries of the VQE modules

Module	Description	Library File	In the AI Uplink Channel or Not	In the AO Downlink Channel or Not	Clipping Supported or Not	Dependency Relationship	Exclusive Relationship
UpVQE	AI audio effect processing	libupvqe.a (.so)	Yes	No	No	N/A	N/A



Module	Description	Library File	In the AI Uplink Channel or Not	In the AO Downlink Channel or Not	Clipping Supported or Not	Dependency Relationship	Exclusive Relationship
	scheduling interface						
DnVQE	AO audio effect processing scheduling interface	libdnvqe.a (.so)	No	Yes	No	N/A	N/A
AEC	Acoustic echo cancellation	libhive_AEC.so	Yes	No	Yes	COMMON	RNR/DRC/P EQ
ANR	Active noise reduction	libhive_ANR.so	Yes	Yes	Yes	COMMON	RNR/DRC/P EQ
AGC	Automatic gain control	libhive_AGC.so	Yes	Yes	Yes	COMMON	RNR/DRC/P EQ
RNR	Recording noise reduction	libhive_RNR.so	Yes	No	Yes	N/A	AEC/ANR/ AGC/EQ
DRC	Dynamic range control	libhive_DRC.so	Yes	No	Yes	N/A	AEC/ANR/ AGC/EQ
PEQ	Parameter equalizer	libhive_PEQ.so	Yes	No	Yes	N/A	AEC/ANR/ AGC/EQ
HDR	High dynamic range	libhive_HDR.so	Yes	No	Yes	N/A	N/A
EQ	Equalizer	libhive_EQ.so	Yes	Yes	Yes	AGC/COMMON	RNR/DRC/P EQ
HPF	High-pass filtering	libhive_HPF.so	Yes	Yes	Yes	N/A	N/A
Gain	Volume adjustment	libhive_GAIN.so	Yes	No	Yes	N/A	N/A
RES	Resampling	libhive_RES.so	Yes	Yes	Yes	N/A	N/A
Common	Common module	libhive_common.so	Yes	Yes	Yes	N/A	N/A



CAUTION

- The MPIs relevant to AI uplink audio effect processing include `HI_MPI_AI_SetVqeAttr`, `HI_MPI_AI_GetVqeAttr`, `HI_MPI_AI_SetHiFiVqeAttr`, `HI_MPI_AI_GetHiFiVqeAttr`, `HI_MPI_AI_SetTalkVqeAttr`, `HI_MPI_AI_GetTalkVqeAttr`, `HI_MPI_AI_EnableVqe`, `HI_MPI_AI_DisableVqe`, `HI_MPI_AI_SetVqeVolume` (relevant only to `libhive_GAIN.so`), and `HI_MPI_AI_SetVqeVolume` (relevant only to `libhive_GAIN.so`). Three pairs of MPIs, `HI_MPI_AI_SetVqeAttr` and `HI_MPI_AI_GetVqeAttr`, `HI_MPI_AI_SetHiFiVqeAttr` and `HI_MPI_AI_GetHiFiVqeAttr`, `HI_MPI_AI_SetTalkVqeAttr` and `HI_MPI_AI_GetTalkVqeAttr`, must be called together. For example, the parameter that is set by calling `HI_MPI_AI_SetVqeAttr` can only be obtained by calling `HI_MPI_AI_GetVqeAttr`. If `HI_MPI_AI_SetVqeAttr` or `HI_MPI_AI_SetHiFiVqeAttr` or `HI_MPI_AI_SetTalkVqeAttr` is called to obtain the parameter, an error code is returned. In addition, any one pair of MPIs cannot be called with other pairs of MPIs at the same time. When one of the set MPIs is called to configure parameters and before `HI_MPI_AI_EnableVqe` is called to enable VQE, the other two set MPIs can be called to configure parameters. If you want to configure parameters after `HI_MPI_AI_EnableVqe` is called, you need to call `HI_MPI_AI_DisableVqe` to disable VQE, and then call one of the set MPIs to configure parameters. For details about the three pairs of MPIs supported by various chips, see [Table 9-3](#). In the table, VQE, Hi-Fi VQE, and Talk VQE represent the three pairs of MPIs respectively.
- The MPIs relevant to AO downlink audio effect processing include `HI_MPI_AO_SetVqeAttr`, `HI_MPI_AO_GetVqeAttr`, `HI_MPI_AO_EnableVqe`, and `HI_MPI_AO_DisableVqe`.
- The MPIs relevant to `libhive_RES.so` include `HI_MPI_AI_EnableReSmp`, `HI_MPI_AI_DisableReSmp`, `HI_MPI_AO_EnableReSmp`, and `HI_MPI_AO_DisableReSmp`.
- `libhive_RES.so` can be cropped only when resampling is disabled and the AI and AO sampling rates are the same as the VQE working sampling rate.
- `libhive_common.so` can be cropped when the AEC, AGC, ANR, and EQ functional modules are not used.
- When the dynamic audio libraries are used, you need to specify the path of the dynamic audio libraries or copy them to `/user/lib` before running the application. The dynamic libraries are loaded when the application is running, and do not need to be linked during compilation.

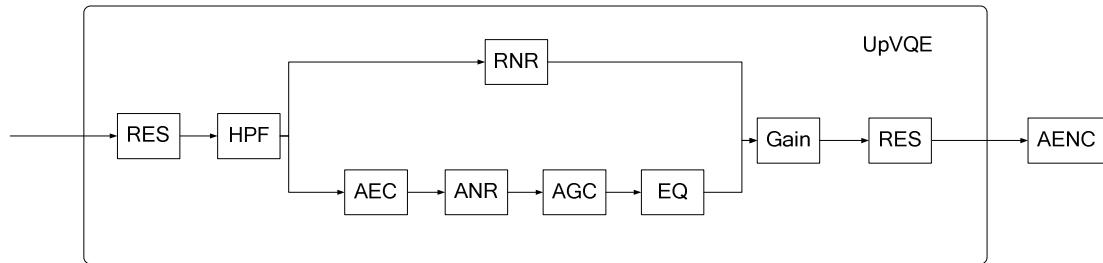
Table 9-3 VQE, Hi-Fi VQE, and Talk VQE supported by various chips

Chip Type	VQE Supported or Not	Hi-Fi VQE Supported or Not	Talk VQE Supported or Not
Hi3516A	Yes	Yes	No
Hi3518E V200	Yes	No	No
Hi3519 V100	No	Yes	Yes

UpVQE is the scheduling logic for AI data processing. [Figure 9-4](#) shows the UpVQE processing flowchart in the AI uplink channel.

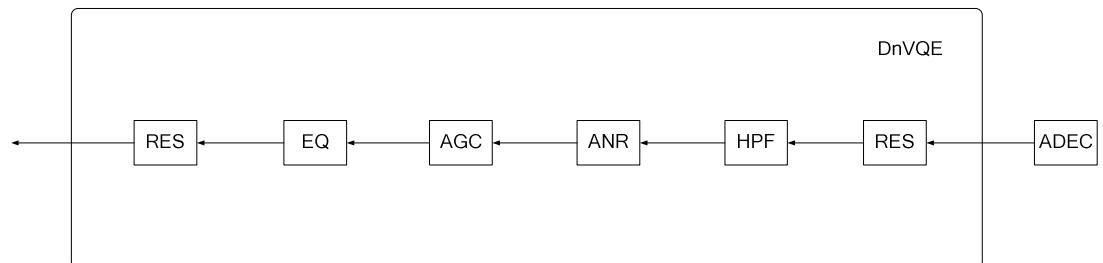


Figure 9-4 UpVQE processing flowchart



DnVQE is the scheduling logic for AO data processing. [Figure 9-5](#) shows the DnVQE processing flowchart in the AO downlink channel.

Figure 9-5 DnVQE processing flowchart



CAUTION

- If an AI channel is bound to an AO channel by calling the system binding interface and data is transferred between the AI channel and AO channel, enabling any of the VQE modules in the AI or AO channel has no effect.
- The VQE function does not support the stereo.
- If an AI channel is bound to an AENC channel by calling the system binding interface, data is transferred between the AI channel and AENC channel, and the VQE function of the AI uplink channel is enabled, VQE and then encoding are performed in the AENC channel.
- If data is transferred between the AI channel and the AENC/AO channel in non-system binding mode and the VQE function of the AI uplink channel is enabled, the VQE function is implemented in [HI_MPI_AI_GetFrame](#). In this case, you can obtain AI frames after VQE processing by calling [HI_MPI_AI_GetFrame](#), and then transmit the frames to the AENC or AO channel. In this way, data is transferred between the AI and AENC channels or AI and AO channels.
- VQE processes only data with the sampling rate of 8 kHz, 11.025 kHz, 12 kHz, 16 kHz, 22.05 kHz, 24 kHz, 32 kHz, 44.1 kHz, or 48 kHz, and does not process data with the sampling rate of 64 kHz or 96 kHz.

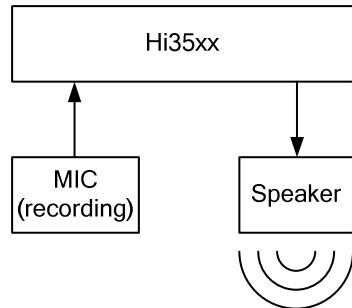
AEC

The AEC module applies to the scenarios that require echo cancellation. For example, when the remote voice data is played on an AO device in the IPC intercom scenario, the voice data



is captured by a local MIC, and the AEC module can remove the playback sound (echo) of the AO device from the recorded voice data.

Figure 9-6 Echo cancellation diagram



Different from other functional modules that require only the signal in (SIN) data, the AEC module requires the SIN data as well as the reference in (RIN) data for algorithm processing to obtain the processed signal out (SOU) data. SIN indicates the near-end input with the echo, and RIN indicates the reference frame (echo) data. The AEC module can be enabled only when the mono-channel mode is used, the AI/AO working sampling rate is 8 kHz or 16 kHz, and the length of the AI frame (voice data captured by the MIC) must be the same as the configured length of the AO frame (remote voice played on an AO device).

ANR

The ANR module applies to the scenarios where the external noise needs to be removed and the voice input needs to be retained.

Compared with the RNR algorithm, the ANR algorithm focuses on the noise reduction effect. It filters out environment noises and retains the voice data. However, some details are lost. Therefore, the ANR algorithm applies to the network video recorder (NVR) and IPC scenarios where customers expect to retain voices and filters out noises.



CAUTION

The ANR module supports the sampling rates of 8 kHz and 16 kHz, and does not support the stereo.

RNR

The RNR module applies to the scenarios where the environment noise needs to be removed and the small signal input needs to be retained.

Compared with the ANR algorithm, the RNR algorithm focuses on the retention of detail inputs (small signals) during noise reduction. The noise reduction strength of the RNR algorithm is lower than that of the ANR algorithm, whereas more field sounds are retained and the scenario reproduction effect is good. Therefore, the RNR algorithm applies to the motion DV scenario.



CAUTION

When the RNR function is used in the VQE interfaces, the sampling rates 8 kHz, 16 kHz, and 48 kHz are supported. When this function is used in the HiFiVqe interfaces, only the 48 kHz sampling rate is supported. When this function is used in the TalkVqe interfaces, the 8 kHz and 16 kHz sampling rates are supported and the stereo is not supported.

DRC

The DRC module is used to control the output level and maintains the output gains within a certain range. It is required in the scenario where the sound needs to be neither too large nor too small.

The function of DRC is similar to that of AGC. However, the algorithm implementation and adjustment strength of DRC and AGC are different. The DRC module is used for the motion digital video (DV) by working with RNR and is exclusive with AEC/ANR/AGC/EQ.



CAUTION

The DRC function supports only the 48 kHz sampling rate and does not support the stereo.

PEQ

The PEQ module applies to the equalizer processing for the audio data to adjust the gains of audio at different frequency bands.

Compared with EQ, the PEQ can be adjusted more flexibly, which is suitable to the motion DV scenario.



CAUTION

The PEQ function supports only 48 kHz sampling rate and does not support the stereo.

HDR

The HDR module applies to the input volume control of the codec. The gains of codec can be dynamically adjusted to maintain the codec volume within a reasonable range, which ensures that the sounds are not too loud or too small.



CAUTION

When the HDR function is used in the VQE interfaces, the sampling rates 8 kHz, 16 kHz, and 48 kHz are supported. When this function is used in the HiFiVqe interfaces, only the 48 kHz sampling rate is supported. When this function is used in the TalkVqe interfaces, the 8 kHz and 16 kHz sampling rates are supported and the stereo is not supported.

HPF

The HPF module removes the low-frequency noises.

The low-frequency noises are mainly hardware noises or power frequency noises, that is, unpleasant booming sound. You can implement spectrum analysis on the board audio streams that are recorded in a quiet environment, and then determine whether to add the HPF module to the VQE. If the low-frequency noises are not obvious and the low-frequency sound sources need to be retained, you are advised not to add this module to the VQE. The recommended parameter configuration is **AUDIO_HPF_FREQ_120** or **AUDIO_HPF_FREQ_150**.



CAUTION

When the HPF function is used in the VQE interfaces, the sampling rates 8 kHz, 16 kHz, and 48 kHz are supported. When this function is used in the HiFiVqe interfaces, only the 48 kHz sampling rate is supported. When this function is used in the TalkVqe interfaces, the 8 kHz and 16 kHz sampling rates are supported and the stereo is not supported. You are advised to always enable the HPF function when VQE is used.

AGC

The AGC module is used to control the gain of the output level. It limits the output volume to a certain range when the input volume changes, and applies to the scenarios where the volume cannot be too high to too low.

The AGC module increases the volume of the input source sound to ensure that the volume after algorithm processing is high even if the volume of the sound source is too low. If the AGC function is enabled in the AI channel, the output volume is controlled by calling [HI_MPI_AI_SetVqeVolume](#) but not by adjusting the AI gain.



CAUTION

The AGC module supports the sampling rates of 8 kHz and 16 kHz, and does not support the stereo.

EQ

The EQ module implements equalization processing on the audio data to adjust the gains of the sound at various frequency bands.



CAUTION

- The equalizer supports the sampling rates of 8 kHz and 16 kHz, and does not support the stereo.
- The equalizer can be used only after the AGC module is enabled.

Gain

The gain module is a volume adjustment module for adjusting the volume after the AGC module is enabled.

The AGC module implements dynamic gain control on the volume. The range of the SIN voice level allowed by the algorithm is 0 dB to -40 dB. After algorithm processing, the maximum voice level is -2 dB, and the maximum voice gain is 30 dB. The SOU voice level can be -2 dB to -10 dB, and it is difficult to adjust the SOU volume by adjusting the AI gain. Therefore, the gain module is located after the AGC module in the VQE processing flowchart (as shown in [Figure 9-4](#)) to implement volume adjustment of the AI uplink channel.

RES

The RES module is a resampling module. When the VQE functional modules are enabled in the AI uplink channel or AO downlink channel, resampling is implemented before and after other VQE processing (as shown in [Figure 9-4](#) and [Figure 9-5](#)). During the first resampling process, the audio data at the input sampling rate is converted into that at the working sampling rate supported by the functional modules (8 kHz, 16 kHz, or 48 kHz). During the second resampling process, data at the working sampling rate is converted into that at the output sampling rate.

9.2.2 Audio Encoding and Decoding

9.2.2.1 Audio Encoding and Decoding Processes

For the Hi35xx SDK, G711, G726, ADPCM_DVI4, or ADPCM_ORG_DVI4 audio decoding is CPU software decoding. All decoding functions are implemented based on the HiSilicon audio encoding and decoding library that is independently encapsulated. The core decoder works in user mode and performs decoding by using the CPU. The SDK can bind an AI channel to an AENC channel by using an SYS module to implement the recording and encoding function; the SDK can also bind an ADEC channel to an AO channel to implement the decoding and playback function.

9.2.2.2 Audio Encoding and Decoding Protocols

[Table 9-4](#) describes the audio encoding and decoding protocols supported by the Hi35xx.

**Table 9-4** Audio encoding and decoding protocols

Protocol	Sampling Rate	Frame Duration (Sampling Point)	Bit Rate (kbit/s)	Compression Rate	CPU Usage	Description
G711	8 kHz	80/160/ 240/320/480	64	2	1 MHz	<p>Advantages: best voice quality; low CPU consumption; wide application; and free of charge.</p> <p>Disadvantage: low compression efficiency.</p> <p>The G.711 provides A-law and μ-law compression codes, which are applicable to the ISTN and most digital telephone lines. North America and Japan usually adopt μ-law code. Europe and other regions usually adopt A-law code.</p>
G726	8 kHz	80/160/ 240/320/480	16, 24, 32, 40 (Note: G726 encoding is a lossy compression. When the bit rate is low, the compression rate and quantization error are large, which may affect the voice quality.)	8–3.2	5 MHz	<p>Advantages: simple algorithm; good voice quality; ensured voice quality after conversions for several times; network-level voice quality at a low bit rate.</p> <p>Disadvantage: low compression efficiency.</p> <p>The G726_16KBPS and the MEDIA_G726_16KBPS encoders differ in the format of packages output. The G726_16KBPS is applicable to network transmission and the MEDIA_G726_16KBPS is applicable to ASF storage. For details, see the <i>RFC3551</i>.</p>
ADPCM	8 kHz	80/160/ 240/320/480 or 81/161/241/3 21/481	32	4	2 MHz	Advantages: simple algorithm; good voice quality; ensured voice quality after conversions for several times; network-level voice



Protocol	Sampling Rate	Frame Duration (Sampling Point)	Bit Rate (kbit/s)	Compression Rate	CPU Usage	Description
						quality at a low bit rate. Disadvantage: low compression efficiency. The ADPCM_IMA, ADPCM_ORG_DVI4, and ADPCM_DVI formats are packet encapsulation formats of the ADPCM. The IMA packet encapsulation uses the first sampling point as the predicted value, that is, one more sampling point needs to be input per frame. The DVI packet encapsulation uses the preceding frame as the predicted value. Therefore, the input sampling points for the IMA encoding are 81/161/241/321/481 and the input sampling points for the DVI encoding are 80/160/240/320/480.



CAUTION

- The CPU usage is calculated based on the 288 MHz ARM9. For example, the value 2 MHz indicates that 2 MHz CPU is used for decoding, and the CPU usage is 0.69 (2/288).
- G726 encoding is a lossy compression. When the bit rate is low, the compression rate and quantization error are large, which may affect the voice quality.
- Exceptions might occur when packets are discarded during transmission because the predicted transmission value is not provided for ADPCM_ORG_DVI4. Therefore, you are advised not to use this protocol in audio stream transmission applications.
- When the LPCM protocol is used, the AIU and AOU can record and play audio files in LPCM format.

9.2.2.3 Structure of the HiSilicon Voice Frame

When the HiSilicon voice encoding/decoding library is used to encode G711, G726, and ADPCM audio data, the encoded stream complies with the structure described in [Table 9-5](#). That is, a 4-byte frame header is added before the payload of each frame stream. When the voice encoding/decoding library is used to decode the audio data, the header information needs to be read.



Table 9-5 Structure of the HiSilicon voice frame

Parameter Position (Unit: HI_S16)	Parameter Bit	Meanings
0	[15:8]	Flag of the frame type 01: voice frame Other values: reserved
	[7:0]	Reserved
1	[15:8]	Frame circulation counter: $0 \geq 255$
	[7:0]	Length of the payload (unit: HI_S16)
2	[15:0]	Payload data
3	[15:0]	Payload data
...	[15:0]	Payload data
$2 + n - 1$	[15:0]	Payload data
$2 + n$	[15:0]	Payload data

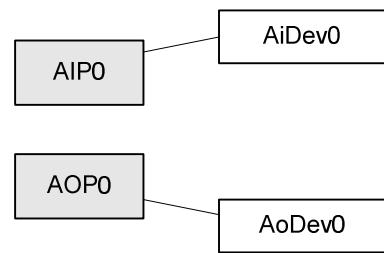
9.2.3 Mapping Between Audio Interfaces and Devices

The integrated AIO interfaces include audio input ports (AIPs) and audio output ports (AOPs). For details about AIO interface specifications, see the data sheet of the corresponding chip.

[Figure 9-7](#) shows the mapping between AIPs/AOPs and AI/AO devices for the Hi3516A/Hi3518E V200.

You can configure whether to share the frame sync clock and bit stream clock of AOP0 with AIP0. If AOP0 and AIP0 share the frame sync clock and bit stream clock, the frame sync clock and bit stream clock of AiDev0 and AoDev0 must be also the same. That is, the product of the sampling precision multiplied by the number of channels for AiDev0 must be the same as that for AoDev0. In addition, the sampling frequencies of AiDev0 and AoDev0 must be the same.

[Figure 9-7](#) Mapping between AIPs/AOPs and AI/AO devices





NOTE

- AIP0 and AOP0 simultaneously connect to a codec for implementing intercom.
- When AOP1 connects to the HDMI, only the I²S master mode is supported.

Ranges of AI and AO Device IDs

[Table 9-6](#) lists the ranges of AI and AO device IDs.

Table 9-6 Ranges of AI and AO device IDs

Chip	Range of the AI Device ID	Range of the AO Device ID
Hi3516A/Hi3518E V200/Hi3519 V100	[0]	[0]

9.2.4 Embedded Audio Codec

9.2.4.1 Overview

The Hi3516A/Hi3518E V200/Hi3519 V100 provides an embedded audio codec (V500 for the Hi3516A/Hi3518E V200 and V600 for Hi3519 V100) for connecting to AIP0 and AOP0 to implement audio playback and recording. AIP0 and AOP0 can connect to the embedded audio codec or an external codec. Because the embedded audio codec cannot transmit the sync clock, AIP0 and AOP0 must work only in master I²S timing mode when AIP0 and AOP0 connect to the embedded audio codec. Audio data can be received or transmitted only when the interconnection timing for AIP0, AOP0, and the audio codec are properly configured.

9.2.4.2 Important Concepts

The embedded audio codec consists of an analog part and a digital part. The analog part selects the MICIN input or LINEIN input by using the analog mixing (MICPGA). The analog mixing supports gain adjustment. The digital part includes the ADC and DAC. The ADC and DAC are used to convert analog signals and digital signals, and adjust the volume. You can adjust both the analog volume and digital volume. It is recommended that you adjust the analog volume first.

The DAC supports soft mute and soft unmute. Soft mute is used to control the digital audio value until it decreases to 0, and soft unmute is used to control the digital audio value until it increases to the configured gain value. The rate for changing the value is controlled by `mute_rate`. When soft mute is enabled, the digital audio value gradually decreases to 0; when soft mute is disabled, the digital audio value restores to the configured gain value.

The working clock of the embedded audio codec is provided by AIP0 and AOP0. The clocks of the analog part and digital part are originated from the working clock of the audio codec, and the clocks of the ADC and DAC are originated from the clock of the digital part. If the sound is abnormal when default configurations are used, you can reverse the clocks of the digital part and analog part or the clocks of the DAC and ADC. The default sampling precision of the audio codec is 16 bits.

The embedded audio codec supports de-emphasis filtering, pop sound suppression, and high-pass filtering. These functions are enabled by default.

When an embedded codec is used, AI device 0 cannot share the clocks of AO device 0.



9.2.4.3 ioctl Function

The user-mode interface of the embedded audio codec is an ioctl function. Its syntax is as follows:

```
int ioctl (int fd,  
          unsigned long cmd,  
          ...  
          );
```

This function is a standard Linux interface and has variable parameters. Only three parameters are required for the audio codec. The syntax is as follows:

```
int ioctl (int fd,  
          unsigned long cmd,  
          CMD_DATA_TYPE *cmddata);
```

The value of **CMD_DATA_TYPE** varies according to the values of **cmd**. [Table 9-7](#) describes the three parameters of the ioctl function.

Table 9-7 Parameters of the ioctl function

Parameter	Description	Input/Output
fd	Descriptor of the embedded audio codec file. It is returned after the embedded audio codec file is opened by calling the open function.	Input
cmd	The main command control words are as follows: <ul style="list-style-type: none">• ACODEC_SOFT_RESET_CTRL: Restores the embedded audio codec to default settings.• ACODEC_SET_I2S1_FS: Sets the sampling rate of the I²S1 interface.• ACODEC_SET_INPUT_VOL: Sets the total input volume.• ACODEC_GET_INPUT_VOL: Obtains the total input volume.• ACODEC_SET_OUTPUT_VOL: Sets the total output volume.• ACODEC_GET_OUTPUT_VOL: Obtains the total output volume.• ACODEC_SET_MIXER_MIC: Selects the input mode.• ACODEC_SET_GAIN_MICL: Sets the analog gain of the audio-left input channel.• ACODEC_SET_GAIN_MICR: Sets the analog gain of the audio-right input channel.• ACODEC_SET_DACL_VOL: Sets the volume of the audio-left output channel.• ACODEC_SET_DACR_VOL: Sets the volume of the audio-right output channel.• ACODEC_SET_ADCL_VOL: Sets the volume of the	Input



Parameter	Description	Input/Output
	<p>audio-left input channel.</p> <ul style="list-style-type: none">• ACODEC_SET_ADCR_VOL: Sets the volume of the audio-right input channel.• ACODEC_SET_MICL_MUTE: Sets the mute of the audio-left input channel.• ACODEC_SET_MICR_MUTE: Sets the mute of the audio-right input channel.• ACODEC_SET_DACL_MUTE: Sets the mute of the audio-left output channel.• ACODEC_SET_DACR_MUTE: Sets the mute of the audio-right output channel.• ACODEC_DAC_SOFT_MUTE: Controls the soft mute function of the DAC.• ACODEC_DAC_SOFT_UNMUTE: Controls the soft unmute function of the DAC.• ACODEC_ENABLE_BOOSTL: Controls the boost analog gain of the audio-left channel.• ACODEC_ENABLE_BOOSTR: Controls the boost analog gain of the audio-right channel.• ACODEC_GET_GAIN_MICL: Obtains the analog gain of the audio-left input channel.• ACODEC_GET_GAIN_MICR: Obtains the analog gain of the audio-right input channel.• ACODEC_GET_DACL_VOL: Obtains the volume of the audio-left output channel.• ACODEC_GET_DACR_VOL: Obtains the volume of the audio-right output channel.• ACODEC_GET_ADCL_VOL: Obtains the volume of the audio-left input channel.• ACODEC_GET_ADCR_VOL: Obtains the volume of the audio-right input channel.• ACODEC_SET_PD_DACL: Powers off the audio-left output channel.• ACODEC_SET_PD_DACR: Powers off the audio-right output channel.• ACODEC_SET_PD_ADCL: Powers off the audio-left input channel.• ACODEC_SET_PD_ADCR: Powers off the audio-right input channel.• ACODEC_SET_PD_LINEINL: Controls the power-off of the line-in input of the audio-left channel.• ACODEC_SET_PD_LINEINR: Controls the power-off of the line-in input of the audio-right channel.• ACODEC_SEL_DAC_CLK: Sets the clock edges of the DAC to same or opposite.• ACODEC_SEL_ADC_CLK: Sets the clock edges of	



Parameter	Description	Input/Output
	<p>the ADC to the same edges or reversed edges.</p> <ul style="list-style-type: none">• ACODEC_SEL_ANA_MCLK: Sets the clock edges of the analog part and digital part to same or opposite.• ACODEC_DACL_SEL_TRACK: Sets the DACL for selecting the audio channel.• ACODEC_DACR_SEL_TRACK: Sets the DACR for selecting the audio channel.• ACODEC_ADCL_SEL_TRACK: Sets the ADCL for selecting the audio channel.• ACODEC_ADCR_SEL_TRACK: Sets the ADCR for selecting the audio channel.• ACODEC_SET_DAC_DE_EMPHASIS: Controls the de-emphasis filtering function of the DAC.• ACODEC_SET_ADC_HP_FILTER: Controls the high-pass filtering function of the ADC.• ACODEC_DAC_POP_FREE: Deletes the POP noise of the DAC.• ACODEC_DAC_SOFT_MUTE_RATE: Controls the rate for soft mute of the DAC.• ACODEC_DAC_SEL_I2S: Selects the I²S interface for the DAC.• ACODEC_ADC_SEL_I2S: Selects the I²S interface for the ADC.• ACODEC_SET_I2S1_DATAWIDTH: Sets the data width of the I²S1 interface.• ACODEC_SET_I2S2_DATAWIDTH: Sets the data width of the I²S2 interface.• ACODEC_SET_I2S2_FS: Sets the sampling rate of the I²S2 interface.• ACODEC_SET_DACR2DACL_VOL: Mixes the volume of the DACR with that of the DACL.• ACODEC_SET_DACL2DACR_VOL: Mixes the volume of the DACL with that of the DACR.• ACODEC_SET_ADCL2DACL_VOL: Mixes the volume of the ADCL with that of the DACL.• ACODEC_SET_ADCR2DACL_VOL: Mixes the volume of the ADCR with that of the DACL.• ACODEC_SET_DACL2DACR_VOL: Mixes the volume of the DACR with that of the DACL.• ACODEC_SET_ADCR2DACR_VOL: Mixes the volume of the ADCR with that of the DACR.	
cmddata	Data pointers corresponding to command control words	Input/output



9.3 API Reference

9.3.1 AI

The AIU is used to configure and enable AI devices and obtain audio frames.

The AIU provides the following MPIS:

- [HI_MPI_AI_SetPubAttr](#): Sets attributes of an AI device.
- [HI_MPI_AI_GetPubAttr](#): Obtains attributes of an AI device.
- [HI_MPI_AI_Enable](#): Enables an AI device.
- [HI_MPI_AI_Disable](#): Disables an AI device.
- [HI_MPI_AI_EnableChn](#): Enables an AI channel.
- [HI_MPI_AI_DisableChn](#): Disables an AI channel.
- [HI_MPI_AI_GetFrame](#): Obtains audio frames.
- [HI_MPI_AI_ReleaseFrame](#): Releases the buffer for storing audio frames.
- [HI_MPI_AI_SetChnParam](#): Sets the parameters of an AI channel.
- [HI_MPI_AI_GetChnParam](#): Obtains the parameters of an AI channel.
- [HI_MPI_AI_EnableReSmp](#): Enables AI resampling.
- [HI_MPI_AI_DisableReSmp](#): Disables AI resampling.
- [HI_MPI_AI_SetVqeAttr](#): Sets the VQE attributes of an AI channel.
- [HI_MPI_AI_GetVqeAttr](#): Obtains the VQE attributes of an AI channel.
- [HI_MPI_AI_SetHiFiVqeAttr](#): Sets the AI VQE (Hi-Fi) attributes.
- [HI_MPI_AI_GetHiFiVqeAttr](#): Obtains the AI VQE (Hi-Fi) attributes.
- [HI_MPI_AI_SetTalkVqeAttr](#): Sets the AI VQE (Talk) attributes.
- [HI_MPI_AI_GetTalkVqeAttr](#): Obtains the AI VQE (Talk) attributes.
- [HI_MPI_AI_EnableVqe](#): Enables the VQE functions of an AI channel.
- [HI_MPI_AI_DisableVqe](#): Disables the VQE functions of an AI channel.
- [HI_MPI_AI_SetTrackMode](#): Sets the AI track mode.
- [HI_MPI_AI_GetTrackMode](#): Obtains the AI track mode.
- [HI_MPI_AI_GetFd](#): Obtains the device file descriptor (FD) of an AI channel.
- [HI_MPI_AI_ClrPubAttr](#): Clears the public attributes of an AI device.
- [HI_MPI_AI_SaveFile](#): Enables the function of saving AI files.
- [HI_MPI_AI_SetVqeVolume](#): Sets the volume of an AI device.
- [HI_MPI_AI_QueryFileStatus](#): Queries whether the AI channel is in the state of saving files.
- [HI_MPI_AI_GetVqeVolume](#): Obtains the volume of an AI device.
- [HI_MPI_AI_EnableAecRefFrame](#): Enables the function of obtaining the AEC reference frame when the AEC function is disabled.
- [HI_MPI_AI_DisableAecRefFrame](#): Disables the function of obtaining the AEC reference frame when the AEC function is disabled.

[HI_MPI_AI_SetPubAttr](#)

[Description]



Sets attributes of an AI device.

[Syntax]

```
HI_S32 HI_MPI_AI_SetPubAttr(AUDIO_DEV AiDevId, const AIO_ATTR_S *pstAttr);
```

[Parameter]

Parameter	Description	Input/Output
AiDevId	AI device ID For details about the value range, see Table 9-6 .	Input
pstAttr	Pointer to the AI attributes	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_ai.h
- Library file: libmpi.a

[Note]

The attributes of the AI device determine the format of the input data. The attributes include working mode, sampling rate, sampling precision, buffer size, number of sampling points per frame, extension flag, clock selection, and number of channels. An AI device can properly connect to an external codec only when the preceding attributes are consistent with these of the timings of the connected codec.

• Working mode

Currently, the AI and AO devices support master and slave I²S modes, master and slave standard PCM modes, and master and slave customized PCM modes.

When an AI device works in slave mode, you are advised to configure the interconnected codec and then the AI device. When an AI device works in master mode, you are advised to configure the AI device and then the interconnected codec. The recommendations are provided based on timings.

• Sampling rate

Sampling rate indicates the number of sampling points per second. A high sampling rate means that the distortion is low. The data that needs to be processed increases. Typically, 8 kHz sampling rate is used for voice and 32 kHz or higher sampling rate is used for audio. Ensure that the sampling rate to be set is supported by the interconnected audio codec.

• Sampling precision



Sampling precision indicates the width of sampling point data on a channel. This attribute determines the allocation of channels throughout the device. The sampling precision can be set to 8 bits or 16 bits (only 16 bits for the Hi3516A/Hi3518E V200/Hi3519 V100). The actual sampling precision is affected by the audio codec.

- Buffer size

u32FrmNum in the AIO_ATTR_S is used to configure the number of audio frames in the buffer storing the audio data received by the AIU. The recommended value for this parameter is **5** or greater. Otherwise, exceptions such as frame loss may occur during sampling.

- Number of sampling points per frame

When the sampling rate is high, you are advised to increase the number of sampling points per frame. To transmit the captured audio data for encoding, ensure that the duration for each frame is longer than or equal to 10 ms. Otherwise, the sound is abnormal after decoding. For example, if the sampling rate is 16 kHz, the number of sampling points for each frame must be at least 160. If the sound is intermittent, the number of sampling points in each frame can be increased accordingly. The parameter configuration is related to the chip performance.

- Number of channels

This attribute indicates the number of channels supporting the AI function of the current input device. The number of channels must be the same as that of the audio codec. The number of channels can be 1, 2, 4, 8, and 16.

- Extension flag

This flag is invalid for the AI device.

- Clock selection

- An AI device can properly work only when it works with the ADC. To obtain data from the correct channel, you must be familiar with the relationship between the channel IDs and the distribution of data collected by the ADC. When the AI device works in master mode, its output clock depends on the sampling rate, sampling precision, and number of channels. The sampling rate multiplied by the number of channels is the bit width for a sampling using the AI device timing.
- The AI and AO devices are independent. You can configure **u32ClkSel** to determine whether to share the configurations of the frame sync clocks and bit stream clocks of AiDev0 and AoDev0. If **u32ClkSel** is set to **1**, the preceding configurations of AiDev0 and AoDev0 are shared. In this case, ensure that the clock configurations of AiDev0 are the same as those of AoDev0, that is, the product of the sampling precision multiplied by the number of channels for AiDev0 must be the same as that for AoDev0, and the sampling rates for AiDev0 and AoDev0 must also be the same. If **u32ClkSel** is set to **0**, the frame sync clocks and bit stream clocks of AiDev0 and AoDev0 are provided by difference sources.
- When the embedded codec is connected, the frame sync clocks and bit stream clocks of AiDev0 and AoDev0 cannot be shared, and **u32ClkSel** must be set to **0**.

[Example]

To set attributes of an AI device and enable the AI device, see the following codes:

```
HI_S32 s32ret;  
AIO_ATTR_S stAttr;  
AUDIO_DEV AiDevId = 0;  
stAttr.enBitwidth = AUDIO_BIT_WIDTH_16;  
stAttr.enSamplerate = AUDIO_SAMPLE_RATE_8000;
```



```
stAttr.enSoundmode = AUDIO_SOUND_MODE_MONO;
stAttr.enWorkmode = AIO_MODE_I2S_SLAVE;
stAttr.u32EXFlag = 0;
stAttr.u32FrmNum = 5;
stAttr.u32PtNumPerFrm = 160;
stAttr.u32ChnCnt = 2;
stAttr.u32ClkSel = 1;
/* set public attribute of AI device*/
s32ret = HI_MPI_AI_SetPubAttr(AiDevId, &stAttr);
if(HI_SUCCESS != s32ret)
{
    printf("set ai %d attr err:0x%x\n", AiDevId, s32ret);
    return s32ret;
}
/* enable AI device */
s32ret = HI_MPI_AI_Enable(AiDevId);
if(HI_SUCCESS != s32ret)
{
    printf("enable ai dev %d err:0x%x\n", AiDevId, s32ret);
    return s32ret;
}
```

[See Also]

None

HI_MPI_AI_GetPubAttr

[Description]

Obtains attributes of an AI device.

[Syntax]

```
HI_S32 HI_MPI_AI_GetPubAttr(AUDIO_DEV AiDevId, AIO_ATTR_S*pstAttr);
```

[Parameter]

Parameter	Description	Input/Output
AiDevId	AI device ID For details about the value range, see Table 9-6 .	Input
pstAttr	Pointer to the general AI attributes	Output

[Return Value]



Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_ai.h
- Library file: libmpi.a

[Note]

- The obtained attributes are the attributes configured last time.
- If attributes are never configured, a code indicating failure is returned.

[Example]

```
HI_S32 s32ret;
AUDIO_DEV AiDevId = 0;
AIO_ATTR_S stAttr;

s32ret = HI_MPI_AI_GetPubAttr(AiDevId, &stAttr);
if(HI_SUCCESS != s32ret)
{
    printf("get ai %d attr err:0x%x\n", AiDevId,s32ret);
    return s32ret;
}
```

[See Also]

None

HI_MPI_AI_Enable

[Description]

Enables an AI device.

[Syntax]

```
HI_S32 HI_MPI_AI_Enable(AUDIO_DEV AiDevId);
```

[Parameter]

Parameter	Description	Input/Output
AiDevId	AI device ID For details about the value range, see Table 9-6.	Input

[Return Value]



Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_ai.h
- Library file: libmpi.a

[Note]

- The attributes of the AI device must be configured before the AI device is enabled. Otherwise, an error indicating that attributes are not configured is returned.
- A code indicating success is returned if the AI device is enabled.

[Example]

See [HI_MPI_AI_SetPubAttr](#).

[See Also]

None

HI_MPI_AI_Disable

[Description]

Disables an AI device.

[Syntax]

```
HI_S32 HI_MPI_AI_Disable(AUDIO_DEV AiDevId);
```

[Parameter]

Parameter	Description	Input/Output
AiDevId	AI device ID For details about the value range, see Table 9-6 .	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]



- Header files: hi_comm_aio.h, mpi_ai.h
- Library file: libmpi.a

[Note]

- A code indicating success is returned if the AI device is disabled.
- All AI channels of the AI device must be disabled before the AI device is disabled.

[Example]

```
HI_S32 s32ret;
AUDIO_DEV AiDevId = 0;

s32ret = HI_MPI_AI_Disable(AiDevId);
if(HI_SUCCESS != s32ret)
{
    printf("disable ai %d err:0x%x\n", AiDevId);
    return s32ret;
}
```

[See Also]

None

HI_MPI_AI_EnableChn

[Description]

Enables an AI channel.

[Syntax]

```
HI_S32 HI_MPI_AI_EnableChn(AUDIO_DEV AiDevId, AI_CHN AiChn);
```

[Parameter]

Parameter	Description	Input/Output
AiDevId	AI device ID For details about the value range, see Table 9-6 .	Input
AiChn	AI channel ID The ID range depends on the AI device attribute parameters u32ChnCnt (maximum number of channels) and enSoundmode (audio channel mode). For details, see AUDIO_SOUND_MODE_E .	Input

[Return Value]

Return Value	Description
0	Success



Return Value	Description
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, i_ai.h
- Library file: libmpi.a

[Note]

Before the AI channel is enabled, the AI device on which the AI channel exists must be enabled. Otherwise, an error indicating that the device is not enabled is returned.

[Example]

None

[See Also]

None

HI_MPI_AI_DisableChn

[Description]

Disables an AI channel.

[Syntax]

```
HI_S32 HI_MPI_AI_DisableChn(AUDIO_DEV AiDevId, AI_CHN AiChn);
```

[Parameter]

Parameter	Description	Input/Output
AiDevId	AI device ID For details about the value range, see Table 9-6 .	Input
AiChn	AI channel ID Value range: [0, AIO_MAX_CHN_NUM].	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]



- Header files: hi_comm_aio.h, mpi_ai.h
- Library file: libmpi.a

[Note]

None

[Example]

None

HI_MPI_AI_GetFrame

[Description]

Obtains audio frames.

[Syntax]

```
HI_S32 HI_MPI_AI_GetFrame(AUDIO_DEV AiDevId, AI_CHN AiChn, AUDIO_FRAME_S
    *pstFrm, AEC_FRAME_S *pstAecFrm, HI_S32 s32MilliSec);
```

[Parameter]

Parameter	Description	Input/Output
AiDevId	AI device ID For details about the value range, see Table 9-6 .	Input
AiChn	AI channel ID The ID range depends on the AI device attribute parameters u32ChnCnt (maximum number of channels) and enSoundmode (audio channel mode).	Input
pstFrm	Pointer to the structure of the audio frame	Output
pstAecFrm	Pointer to the structure of the reference frame for echo cancellation	Output
s32MilliSec	Timeout period for obtaining data <ul style="list-style-type: none">• -1: block mode. The wait is infinite if there is no data.• 0: non-block mode. An error is reported and the MPI is returned if there is no data.• > 0: The MPI is blocked for s32MilliSec ms. If a timeout occurs, an error is reported and the MPI is returned.	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."



[Requirement]

- Header files: hi_comm_aio.h, mpi_ai.h
- Library file: libmpi.a

[Note]

- If echo cancellation is enabled in the AI channel, **pstAecFrm** cannot be **null**; if echo cancellation is disabled in the AI channel, **pstAecFrm** can be **null**.
- The AIU buffers the audio frame data that can be fetched by users. You can set the buffer depth by calling [HI_MPI_AI_SetChnParam](#). The default buffer depth is **0**.
- The value of **s32MilliSec** must be greater than or equal to **-1**. When **s32MilliSec** is **-1**, data is obtained in block mode. When **s32MilliSec** is **0**, data is obtained in non-block mode. When **s32MilliSec** is greater than **0**, the MPI is returned and an error is reported if there is no data after the MPI is blocked for **s32MilliSec** ms.
- Enable the corresponding AI channel before obtaining audio frame data.
- It is recommended that data be obtained in block mode when VQE is enabled.

[Example]

None

[See Also]

None

HI_MPI_AI_ReleaseFrame

[Description]

Releases the buffer for storing audio frames.

[Syntax]

```
HI_S32 HI_MPI_AI_ReleaseFrame(AUDIO_DEV AiDevId, AI_CHN AiChn,  
AUDIO_FRAME_S *pstFrm, AEC_FRAME_S *pstAecFrm);
```

[Parameter]

Parameter	Description	Input/Output
AiDevId	AI device ID For details about the value range, see Table 9-6 .	Input
AiChn	AI channel ID The ID range depends on the AI device attribute parameters u32ChnCnt (maximum number of channels) and enSoundmode (audio channel mode).	Input
pstFrm	Pointer to the structure of the audio frame	Input
pstAecFrm	Pointer to the structure of the reference frame for echo cancellation	Input



[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_ai.h
- Library file: libmpi.a

[Note]

If you do not want to release the buffer that stores the reference frame for echo cancellation, set pstAecFrm to null.

[Example]

None

[See Also]

None

HI_MPI_AI_SetChnParam

[Description]

Sets the parameters of an AI channel.

[Syntax]

```
HI_S32 HI_MPI_AI_SetChnParam(AUDIO_DEV AiDevId, AI_CHN AiChn,  
AI_CHN_PARAM_S *pstChnParam);
```

[Parameter]

Parameter	Description	Input/Output
AiDevId	AI device ID For details about the value range, see Table 9-6 .	Input
AiChn	AI channel ID The ID range depends on the AI device attribute parameters u32ChnCnt (maximum number of channels) and enSoundmode (audio channel mode).	Input
pstChnParam	Audio channel parameter	Input

[Return Value]



Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_ai.h
- Library file: libmpi.a

[Note]

- The channel parameter has only one variable. This variable is used to set the depth of the buffer for storing the audio frames obtained by users. The default buffer depth is 0. The maximum value of the variable is 30.
- You are advised to call [HI_MPI_AI_GetChnParam](#) to obtain default configurations before calling [HI_MPI_AI_SetChnParam](#) to modify configurations. This facilitates the future expansion.

[Example]

None

[See Also]

None

HI_MPI_AI_GetChnParam

[Description]

Obtains the parameters of an AI channel.

[Syntax]

```
HI_S32 HI_MPI_AI_GetChnParam(AUDIO_DEV AiDevId, AI_CHN AiChn,  
AI_CHN_PARAM_S *pstChnParam);
```

[Parameter]

Parameter	Description	Input/Output
AiDevId	AI device ID For details about the value range, see Table 9-6 .	Input
AiChn	AI channel ID The ID range depends on the AI device attribute parameters u32ChnCnt (maximum number of channels) and enSoundmode (audio channel mode).	Input
pstChnParam	Audio channel parameter	Output



[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_ai.h
- Library file: libmpi.a

[Note]

None

[Example]

None

[See Also]

None

HI_MPI_AI_EnableReSmp

[Description]

Enables AI resampling.

[Syntax]

```
HI_S32 HI_MPI_AI_EnableReSmp(AUDIO_DEV AiDevId, AI_CHN AiChn,  
AUDIO_SAMPLE_RATE_E enOutSampleRate);
```

[Parameter]

Parameter	Description	Input/Output
AiDevId	AI device ID For details about the value range, see Table 9-6 .	Input
AiChn	AI channel ID The ID range depends on the maximum number of channels specified by the AI device attribute parameter u32ChnCnt	Input
enOutSampleRate	Output sampling rate for audio resampling	Input

[Return Value]



Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_ai.h
- Library files: libmpi.a, libupvqe.a, libhive_RES.so, libhive_common.so

[Note]

- Call this MPI to enable resampling after enabling an AI channel.
- Resampling can be repeatedly enabled. Ensure that the attributes configured every time are the same.
- If you want to enable resampling after disabling an AI channel, you must enable this channel and call this MPI again.

[Example]

To set the AI resampling rate from 32 kHz to 8 kHz, see the following codes:

```
/* dev attr of ai */
AUDIO_SAMPLE_RATE_E enOutSampleRate;
stAioAttr.u32ChnCnt = 2;
stAioAttr.enBitwidth = AUDIO_BIT_WIDTH_16;
stAioAttr.enSamplerate = AUDIO_SAMPLE_RATE_32000;
stAioAttr.enSoundmode = AUDIO_SOUND_MODE_MONO;
stAioAttr.u32EXFlag = 1;
stAioAttr.u32FrmNum = 30;
stAioAttr.u32PtNumPerFrm = 320*4;
enOutSampleRate = AUDIO_SAMPLE_RATE_8000;
s32Ret = HI_MPI_AI_EnableReSmp(AiDev, AiChn, enOutSampleRate);
if (HI_SUCCESS != s32Ret)
{
printf("func(%s) line(%d): failed, s32Ret:0x%x\n", __FUNCTION__, __LINE__,
s32Ret);
return s32Ret;
}
```

HI_MPI_AI_DisableReSmp

[Description]

Disables AI resampling.

[Syntax]

```
HI_S32 HI_MPI_AI_DisableReSmp(AUDIO_DEV AiDevId, AI_CHN AiChn);
```



[Parameter]

Parameter	Description	Input/Output
AiDevId	AI device ID For details about the value range, see Table 9-6 .	Input
AiChn	AI channel ID Value range: [0, AIO_MAX_CHN_NUM)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_ai.h
- Library files: libmpi.a, libupvqe.a, libhive_RES.so, libhive_common.so

[Note]

- This MPI is called to disable AI resampling when resampling is not used any more.
- Before calling this MPI, you must disable the AENC channel and AO channel corresponding to the AI device. Otherwise, calling this MPI fails.

HI_MPI_AI_SetVqeAttr

[Description]

Sets the VQE attributes of an AI channel.

[Syntax]

```
HI_S32 HI_MPI_AI_SetVqeAttr(AUDIO_DEV AiDevId, AI_CHN AiChn, AUDIO_DEV  
AoDevId, AO_CHN AoChn, AI\_VQE\_CONFIG\_S *pstVqeConfig);
```

[Parameter]

Parameter	Description	Input/Output
AiDevId	AI device ID For details about the value range, see Table 9-6 .	Input
AiChn	AI channel ID Value range: [0, AIO_MAX_CHN_NUM)	Input



Parameter	Description	Input/Output
AoDevId	ID of the AO device for which echo cancellation is performed For details about the value range, see Table 9-6 .	Input
AoChn	ID of the AO channel for which echo cancellation is performed Value range: [0, AIO_MAX_CHN_NUM)	Input
pstVqeConfig	Pointer to the data structure of the AI VQE configuration	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_ai.h
- Library files: libmpi.a, libupvqe.a, libhive_common.so, libhive_RNR.so, libhive_HPF.so, libhive_GAIN.so, libhive_EQ.so, libhive_ANR.so, libhive_AGC.so, libhive_AEC.so, libhive_HDR

[Note]

- Configure the VQE attributes of the corresponding AI channel before enabling the VQE functions.
- Enable the corresponding AI channel before configuring AI VQE attributes.
- The VQE attributes of an AI channel cannot be dynamically configured. Therefore, before setting the VQE attributes of an AI channel again, you need to disable its VQE functions.
- You can enable some or all VQE functions by setting the corresponding VQE attributes. RNR is used only in the motion DV scenario, whereas ANR is used in the IPC scenario. RNR and ANR cannot be used at the same time.

[Example]

None

HI_MPI_AI_GetVqeAttr

[Description]

Obtains the VQE attributes of an AI channel.

[Syntax]



```
HI_S32 HI_MPI_AI_GetVqeAttr(AUDIO_DEV AiDevId, AI_CHN AiChn,  
AI_VQE_CONFIG_S *pstVqeConfig);
```

[Parameter]

Parameter	Description	Input/Output
AiDevId	AI device ID For details about the value range, see Table 9-6 .	Input
AiChn	AI channel ID Value range: [0, AIO_MAX_CHN_NUM)	Input
pstVqeConfig	Pointer to the data structure of the AI VQE configuration	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_ai.h
- Library files: libmpi.a, libupvqe.a, libhive_common.so, libhive_RNR.so, libhive_HPF.so, libhive_GAIN.so, libhive_EQ.so, libhive_ANR.so, libhive_AGC.so, libhive_AEC.so, libhive_HDR.so

[Note]

Set the VQE attributes of the corresponding AI channel before obtaining VQE attributes.

[Example]

None

HI_MPI_AI_SetHiFiVqeAttr

[Description]

Sets the AI VQE (Hi-Fi) attributes.

[Syntax]

```
HI_S32 HI_MPI_AI_SetHiFiVqeAttr(AUDIO_DEV AiDevId, AI_CHN AiChn,  
AI_HIFIVQE_CONFIG_S *pstVqeConfig);
```

[Parameter]



Parameter	Description	Input/Output
AiDevId	AI device ID For details about the value range, see Table 9-5 .	Input
AiChn	AI channel ID Value range: [0, AIO_MAX_CHN_NUM)	Input
pstVqeConfig	Pointer to the data structure of the AI VQE configuration	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_ai.h
- Library files: libmpi.a, libupvqe.a, libhive_common.so, libhive_RNR.so, libhive_HPF.so, libhive_GAIN.so, libhive_HDR.so, libhive_PEQ.so, libhive_DRC.so

[Note]

- Set the VQE attributes of the corresponding AI channel before enabling the VQE functions.
- Enable the corresponding AI channel before configuring AI VQE attributes.
- The VQE attributes of an AI channel cannot be dynamically configured. Therefore, you need to disable its VQE functions before reconfiguring the VQE attributes of an AI channel.
- You can enable some or all VQE functions by setting the corresponding VQE attributes. Hi-Fi VQE is only used in the motion DV scenario.

[Example]

None

HI_MPI_AI_GetHiFiVqeAttr

[Description]

Obtains the AI VQE (Hi-Fi) attributes.

[Syntax]

```
HI_S32 HI_MPI_AI_GetHiFiVqeAttr(AUDIO_DEV AiDevId, AI_CHN AiChn,  
AI_HIFIVQE_CONFIG_S *pstVqeConfig);
```

[Parameter]



Parameter	Description	Input/Output
AiDevId	AI device ID For details about the value range, see Table 9-5 .	Input
AiChn	AI channel ID Value range: [0, AIO_MAX_CHN_NUM)	Input
pstVqeConfig	Pointer to the data structure of the AI VQE configuration	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_ai.h
- Library files: libmpi.a, libupvqe.a, libhive_common.so, libhive_RNR.so, libhive_HPF.so, libhive_GAIN.so, libhive_HDR.so, libhive_PEQ.so, libhive_DRC.so

[Note]

Set the VQE attributes of the corresponding AI channel before obtaining VQE attributes.

[Example]

None

HI_MPI_AI_SetTalkVqeAttr

[Description]

Sets the AI VQE (Talk) attributes.

[Syntax]

```
HI_S32 HI_MPI_AI_SetTalkVqeAttr(AUDIO_DEV AiDevId, AI_CHN AiChn,  
AUDIO_DEV AoDevId, AO_CHN AoChn, AI_TALKVQE_CONFIG_S *pstVqeConfig);
```

[Parameter]

Parameter	Description	Input/Output
AiDevId	AI device ID For details about the value range, see Table 9-5 .	Input



Parameter	Description	Input/Output
AiChn	AI channel ID Value range: [0, AIO_MAX_CHN_NUM)	Input
AoDevId	ID of the AO device used for echo cancellation For details about the value range, see Table 9-5 .	Input
AoChn	ID of the AO channel used for echo cancellation Value range: [0, AIO_MAX_CHN_NUM)	Input
pstVqeConfig	Pointer to the data structure of the AI VQE configuration	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_ai.h
- Library files: libmpi.a, libupvqe.a, libhive_common.so, libhive_RNR.so, libhive_HPF.so, libhive_GAIN.so, libhive_EQ.so, libhive_ANR.so, libhive_AGC.so, libhive_AEC.so, libhive_HDR.so

[Note]

- Set the VQE attributes of the corresponding AI channel before enabling the VQE functions.
- Enable the corresponding AI channel before configuring AI VQE attributes.
- The VQE attributes of an AI channel cannot be dynamically configured. Therefore, you need to disable its VQE functions before reconfiguring the VQE attributes of an AI channel.
- You can enable some or all VQE functions by setting the corresponding VQE attributes. Talk VQE is used in the IPC scenario.

[Example]

None

HI_MPI_AI_GetTalkVqeAttr

[Description]

Obtains the AI VQE (Talk) attributes.

[Syntax]



```
HI_S32 HI_MPI_AI_GetTalkVqeAttr(AUDIO_DEV AiDevId, AI_CHN AiChn,  
AI_TALKVQE_CONFIG_S *pstVqeConfig);
```

[Parameter]

Parameter	Description	Input/Output
AiDevId	AI device ID For details about the value range, see Table 9-6 .	Input
AiChn	AI channel ID Value range: [0, AIO_MAX_CHN_NUM)	Input
pstVqeConfig	Pointer to the data structure of the AI VQE configuration	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_ai.h
- Library files: libmpi.a, libupvqe.a, libhive_common.so, libhive_RNR.so, libhive_HPF.so, libhive_GAIN.so, libhive_EQ.so, libhive_ANR.so, libhive_AGC.so, libhive_AEC.so, libhive_HDR.so

[Note]

Set the VQE attributes of the corresponding AI channel before obtaining VQE attributes.

[Example]

None

HI_MPI_AI_EnableVqe

[Description]

Enables the VQE functions of an AI channel.

[Syntax]

```
HI_S32 HI_MPI_AI_EnableVqe(AUDIO_DEV AiDevId, AI_CHN AiChn);
```

[Parameter]



Parameter	Description	Input/Output
AiDevId	AI device ID For details about the value range, see Table 9-6 .	Input
AiChn	AI channel ID Value range: [0, AIO_MAX_CHN_NUM)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_ai.h
- Library files: libmpi.a, libupvqe.a, libhive_common.so, libhive_RNR.so, libhive_HPF.so, libhive_GAIN.so, libhive_EQ.so, libhive_ANR.so, libhive_AGC.so, libhive_AEC.so, libhive_HDR.so, libhive_PEQ.so, libhive_DRC.so

[Note]

- Enable the corresponding AI channel before enabling the VQE functions.
- If the VQE functions of an AI channel are enabled repeatedly, a code indicating success is returned.
- After an AI channel is disabled, if you want to enable it again and use the VQE functions, you need to call this MPI to enable the VQE functions.

[Example]

None

HI_MPI_AI_DisableVqe

[Description]

Disables the VQE functions of an AI channel.

[Syntax]

```
HI_S32 HI_MPI_AI_DisableVqe(AUDIO_DEV AiDevId, AI_CHN AiChn);
```

[Parameter]

Parameter	Description	Input/Output
AiDevId	AI device ID For details about the value range, see Table 9-6 .	Input



Parameter	Description	Input/Output
AiChn	AI channel ID Value range: [0, AIO_MAX_CHN_NUM)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_ai.h
- Library files: libmpi.a, libupvqe.a, libhive_common.so, libhive_RNR.so, libhive_HPF.so, libhive_GAIN.so, libhive_EQ.so, libhive_ANR.so, libhive_AGC.so, libhive_AEC.so, libhive_HDR.so, libhive_PEQ.so, libhive_DRC.so

[Note]

- If the VQE functions of an AI channel are not used any more, call this MPI to disable them.
- If the VQE functions of an AI channel are disabled repeatedly, a code indicating success is returned.

[Example]

None

HI_MPI_AI_SetTrackMode

[Description]

Sets the AI track mode.

[Syntax]

```
HI_S32 HI_MPI_AI_SetTrackMode(AUDIO_DEV AiDevId, AUDIO_TRACK_MODE_E enTrackMode);
```

[Parameter]

Parameter	Description	Input/Output
AiDevId	AI device ID For details about the value range, see Table 9-6 .	Input
enTrackMode	AI track mode	Input



[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_ai.h
- Library file: libmpi.a

[Note]

- Call this MPI after enabling an AI device successfully.
- The track mode can be set only when the AI device works in I²S mode.

[Example]

```
HI_S32 s32Ret;
AUDIO_DEV AiDev = 0;
AUDIO_TRACK_MODE_E enTrackMode = AUDIO_TRACK_NORMAL;
AUDIO_TRACK_MODE_E temp;
s32Ret = HI_MPI_AI_SetTrackMode(AiDev, enTrackMode);
if (HI_SUCCESS != s32Ret)
{
    printf("Ai set track mode failure! AiDev: %d, enTrackMode: %d, s32Ret: 0x%x.\n", AiDev, enTrackMode, s32Ret);
    return s32Ret;
}
s32Ret = HI_MPI_AI_GetTrackMode(AiDev, &temp);
if (HI_SUCCESS != s32Ret)
{
    printf("Ai get track mode failure! AiDev: %d, s32Ret: 0x%x.\n", AiDev, s32Ret);
    return s32Ret;
}
```

HI_MPI_AI_GetTrackMode

[Description]

Obtains the AI track mode.

[Syntax]

```
HI_S32 HI_MPI_AI_GetTrackMode(AUDIO_DEV AiDevId, AUDIO_TRACK_MODE_E *penTrackMode);
```



[Parameter]

Parameter	Description	Input/Output
AiDevId	AI device ID For details about the value range, see Table 9-6 .	Input
penTrackMode	Pointer to the AI track mode	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_ai.h
- Library file: libmpi.a

[Note]

- Call this MPI after enabling an AI device successfully.
- The track mode can be obtained only when the AI device works in I²S mode.

[Example]

None

HI_MPI_AI_GetFd

[Description]

Obtains the device FD of an AI channel.

[Syntax]

```
HI_S32 HI_MPI_AI_GetFd(AUDIO_DEV AiDevId, AI_CHN AiChn)
```

[Parameter]

Parameter	Description	Input/Output
AiDevId	AI device ID For details about the value range, see Table 9-6 .	Input
AiChn	AI channel ID Value range: [0, AIO_MAX_CHN_NUM)	Input



[Return Value]

Return Value	Description
Positive value	Valid
Non-positive value	Invalid

[Requirement]

- Header files: hi_comm_aio.h, mpi_ai.h
- Library file: libmpi.a

[Note]

None

[Example]

```
HI_S32 s32AiFd;
HI_S32 s32ret;
AUDIO_DEV AiDevId = 0;
AI_CHN AiChn = 0;
s32AiFd = HI_MPI_AI_GetFd(AiDevId, AiChn);
if(s32AiFd <= 0)
{
    return HI_FAILURE;
}
```

[See Also]

None

HI_MPI_AI_ClrPubAttr

[Description]

Clears the public attributes of an AI device.

[Syntax]

```
HI_S32 HI_MPI_AI_ClrPubAttr(AUDIO_DEV AiDevId);
```

[Parameter]

Parameter	Description	Input/Output
AiDevId	AI device ID For details about the value range, see Table 9-6 .	Input

[Return Value]



Return Value	Description
Positive value	Valid
Non-positive value	Invalid

[Requirement]

- Header files: hi_comm_aio.h, mpi_ai.h
- Library file: libmpi.a

[Note]

- Before the device attributes are cleared, ensure that the device is stopped.
- When the AO device shares the clock of the AI device and both the AI device and AO device are disabled, the attribute of the AO device needs to be forcibly switched (the modified attribute does not meet the requirement for sharing the clock: **enSamplerate** and **u32ClkChnCnt x** (Sampling bit width corresponding to **enBitwidth**) are the same for the AI and AO devices). It is recommended that this MPI be called to clear the internal coupling relationship with the AI before switching.

[See Also]

None

HI_MPI_AI_SaveFile

[Description]

Enables the function of saving AI files.

[Syntax]

```
HI_S32 HI_MPI_AI_SaveFile(AUDIO_DEV AiDevId, AI_CHN AiChn,  
AUDIO_SAVE_FILE_INFO_S *pstSaveFileInfo);
```

[Parameter]

Parameter	Description	Input/Output
AiDevId	AI device ID For details about the value range, see Table 9-6 .	Input
AiChn	AI channel ID Value range: [0, AIO_MAX_CHN_NUM)	Input
pstSaveFileInfo	Pointer to the attribute structure of the function of saving AI files	Input

[Return Value]



Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_ai.h
- Library file: libmpi.a

[Note]

This MPI is used only to dump the files before and after VQE processing when the AI channel is bound to the AENC or AO channel in non-system binding mode. If VQE is disabled or the AI channel is bound to the AENC or AO channel in system binding mode, AI data fails to be dumped by calling this MPI. After this MPI is called, three files with specified sizes are generated in the specified folder. **Sin.pcm** is the input frame before VQE processing, **Rin.pcm** is the echo cancellation reference frame before VQE processing, and **Sou.pcm** is the output frame after VQE processing.

[See Also]

None

HI_MPI_AI_SetVqeVolume

[Description]

Sets the volume of an AI device.

[Syntax]

```
HI_S32 HI_MPI_AI_SetVqeVolume(AUDIO_DEV AiDevId, AI_CHN AiChn, HI_S32 s32VolumeDb);
```

[Parameter]

Parameter	Description	Input/Output
AiDevId	AI device ID For details about the value range, see Table 9-6 .	Input
AiChn	AI channel ID Value range: [0, AIO_MAX_CHN_NUM]	Input
s32VolumeDb	Volume of an AI device (in dB) Value range: [-20, +10]	Input

[Return Value]



Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_ai.h
- Library files: libmpi.a, libupvqe.a, libhive_common.so, libhive_GAIN.so

[Note]

Call this MPI after AI VQE is enabled.

[See Also]

None

HI_MPI_AI_QueryFileStatus

[Description]

Queries whether the AI channel is in the state of saving files.

[Syntax]

```
HI_S32 HI_MPI_AI_QueryFileStatus(AUDIO_DEV AiDevId, AI_CHN AiChn,  
AUDIO_FILE_STATUS_S* pstFileStatus);
```

[Parameter]

Parameter	Description	Input/Output
AiDevId	AI device ID For details about the value range, see Table 9-6 .	Input
AiChn	AI channel ID Value range: [0, AIO_MAX_CHN_NUM]	Input
pstFileStatus	Pointer to the state attribute structure	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]



- Header files: hi_comm_aio.h, mpi_ai.h
- Library file: libmpi.a

[Note]

This MPI is used to query whether the AI channel is in the state of saving files. After [HI_MPI_AI_SaveFile](#) is called to save a file, you can call this MPI to query whether the file to be saved reaches the specified size. If **bSaving** in **pstFileStatus** is **HI_TRUE**, the specified size is not reached. If **bSaving** in **pstFileStatus** is **HI_FALSE**, the specified size is reached.

[See Also]

None

HI_MPI_AI_GetVqeVolume

[Description]

Obtains the volume of an AI device.

[Syntax]

```
HI_S32 HI_MPI_AI_GetVqeVolume(AUDIO_DEV AiDevId, AI_CHN AiChn, HI_S32  
*ps32VolumeDb);
```

[Parameter]

Parameter	Description	Input/Output
AiDevId	AI device ID For details about the value range, see Table 9-6 .	Input
AiChn	AI channel ID Value range: [0, AIO_MAX_CHN_NUM)	Input
ps32VolumeDb	Pointer to the volume of an AI device	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_ai.h
- Library files: libmpi.a, libupvqe.a, libhive_common.so, libhive_GAIN.so

[Note]

Call this MPI after AI VQE is enabled.



[See Also]

None

HI_MPI_AI_EnableAecRefFrame

[Description]

Enables the function of obtaining the AEC reference frame when the AEC function is disabled.

[Syntax]

```
HI_S32 HI_MPI_AI_EnableAecRefFrame(AUDIO_DEV AiDevId, AI_CHN AiChn,  
AUDIO_DEV AoDevId, AO_CHN AoChn);
```

[Parameter]

Parameter	Description	Input/Output
AiDevId	AI device ID For details about the value range, see Table 9-6 .	Input
AiChn	AI channel ID The ID range depends on the maximum number of channels specified by the AI device attribute parameter u32ChnCnt .	Input
AoDevId	ID of the AO device used to obtain the AEC reference frame For details about the value range, see Table 9-6 .	Input
AoChn	ID of the AO channel used to obtain the AEC reference frame Value range: [0, AIO_MAX_CHN_NUM)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_ai.h
- Library file: libmpi.a

[Note]



- This MPI is used only to enable the function of obtaining the AEC reference frame when the AEC function is disabled. The AEC reference frame is returned after [HI_MPI_AI_GetFrame](#) is called and is used for upper-level AEC processing by the user.
- If this MPI is called when the AEC function is enabled, the error code [HI_ERR_AI_NOT_SUPPORT](#) is returned.
- This MPI can be called only when the AI channel is enabled.
- The stereo is not supported.

[See Also]

None

HI_MPI_AI_DisableAecRefFrame

[Description]

Disables the function of obtaining the AEC reference frame when the AEC function is disabled.

[Syntax]

```
HI_S32 HI_MPI_AI_DisableAecRefFrame(AUDIO_DEV AiDevId, AI_CHN AiChn);
```

[Parameter]

Parameter	Description	Input/Output
AiDevId	AI device ID For details about the value range, see Table 9-6 .	Input
AiChn	AI channel ID Value range: [0, AIO_MAX_CHN_NUM)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_ai.h
- Library file: libmpi.a

[Note]

If this MPI is called repeatedly, a code indicating success is returned.

[Example]

None



[See Also]

None

9.3.2 AO

An AOU mainly implements the following functions: enabling an AOU and transmitting audio frames to AO channels.

The AOU provides the following MPIs:

- [HI_MPI_AO_SetPubAttr](#): Sets attributes of an AO device.
- [HI_MPI_AO_GetPubAttr](#): Obtains attributes of an AO device.
- [HI_MPI_AO_Enable](#): Enables an AO device.
- [HI_MPI_AO_Disable](#): Disables an AO device.
- [HI_MPI_AO_EnableChn](#): Enables an AO channel.
- [HI_MPI_AO_DisableChn](#): Disables an AO channel.
- [HI_MPI_AO_SendFrame](#): Transmits AO frames.
- [HI_MPI_AO_EnableReSmp](#): Enables AO resampling.
- [HI_MPI_AO_DisableReSmp](#): Disables AO resampling.
- [HI_MPI_AO_PauseChn](#): Pauses an AO channel.
- [HI_MPI_AO_ResumeChn](#): Resumes an AO channel.
- [HI_MPI_AO_ClearChnBuf](#): Clears the current audio data buffer on the AO channel.
- [HI_MPI_AO_QueryChnStat](#): Queries the status of the current audio data buffer on the AO channel.
- [HI_MPI_AO_SetTrackMode](#): Sets the track mode of an AO device.
- [HI_MPI_AO_GetTrackMode](#): Obtains the track mode of an AO device.
- [HI_MPI_AO_SetVolume](#): Sets the volume of an AO device.
- [HI_MPI_AO_GetVolume](#): Obtains the volume of an AO device.
- [HI_MPI_AO_SetMute](#): Sets the mute status of an AO device.
- [HI_MPI_AO_GetMute](#): Obtains the mute status of an AO device.
- [HI_MPI_AO_GetFd](#): Obtains the device FD of an AO channel.
- [HI_MPI_AO_SaveFile](#): Enables the function of saving AO files.
- [HI_MPI_AO_QueryFileStatus](#): Queries whether the AO channel is in the state of saving files.
- [HI_MPI_AO_ClrPubAttr](#): Clears the public attributes of an AO device.
- [HI_MPI_AO_SetVqeAttr](#): Sets the AO VQE attributes.
- [HI_MPI_AO_GetVqeAttr](#): Obtains the AO VQE attributes.
- [HI_MPI_AO_EnableVqe](#): Enables AO VQE.
- [HI_MPI_AO_DisableVqe](#): Disables AO VQE.

HI_MPI_AO_SetPubAttr

[Description]

Sets attributes of an AO device.

[Syntax]



```
HI_S32 HI_MPI_AO_SetPubAttr(AUDIO_DEV AoDevId, const AIO_ATTR_S *pstAttr);
```

[Parameter]

Parameter	Description	Input/Output
AoDevId	AO device ID For details about the value range, see Table 9-6 .	Input
pstAttr	AO device attributes	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_ao.h
- Library file: libmpi.a

[Note]

- The AO device must be disabled before attributes are set. If the AO device is enabled, disable it before setting attributes.
- The AO device must work with the DAC. To transmit data to the right channel, users need to know data (sent by the DAC) allocation and channel mapping.
- If an external codec is connected and the AO device works in slave mode, you are advised to configure the codec and then the AO device. If an external codec is connected and the AO device works in master mode, you are advised to configure the AO device and then the codec. This is the timing requirement. If the embedded codec is connected, you must configure the embedded codec and then the AO device.
- When the embedded codec is connected, the frame sync clocks and bit stream clocks of AiDev0 and AoDev0 cannot be shared, and **u32ClkSel** must be set to **0**.
- When the AO device works in master mode, the configuration of the AO device output clock depends on the sampling rate, sampling precision, and number of channels. The sampling precision multiplied by the number of channels is the bit width of each sampling in accordance with the AO device timing.
- The extension flag is invalid to the AO device.
- For details about other attributes of the AO device, see the related description about the AIU in section [9.3.1 "AI."](#)

[Example]

```
HI_S32 s32ret;
AIO_ATTR_S stAttr;
AUDIO_DEV AoDevId = 0;
```



```
stAttr.enBitwidth = AUDIO_BIT_WIDTH_16;
stAttr.enSamplerate = AUDIO_SAMPLE_RATE_8000;
stAttr.enSoundmode = AUDIO_SOUND_MODE_MONO;
stAttr.enWorkmode = AIO_MODE_I2S_SLAVE;
stAttr.u32EXFlag = 0;
stAttr.u32FrmNum = 5;
stAttr.u32PtNumPerFrm = 160;
stAttr.u32ChnCnt = 2;
stAttr.u32ClkSel = 0;

/* set ao public attr*/
s32ret = HI_MPI_AO_SetPubAttr(AoDevId, &stAttr);
if(HI_SUCCESS != s32ret)
{
    printf("set ao %d attr err:0x%x\n", AoDevId, s32ret);
    return s32ret;
}
/* enable ao device*/
s32ret = HI_MPI_AO_Enable(AoDevId);
if(HI_SUCCESS != s32ret)
{
    printf("enable ao dev %d err:0x%x\n", AoDevId, s32ret);
    return s32ret;
}
```

[See Also]

None

HI_MPI_AO_GetPubAttr

[Description]

Obtains attributes of an AO device.

[Syntax]

```
HI_S32 HI_MPI_AO_GetPubAttr(AUDIO_DEV AoDevId, AIO_ATTR_S *pstAttr);
```

[Parameter]

Parameter	Description	Input/Output
AoDevId	AO device ID For details about the value range, see Table 9-6 .	Input
pstAttr	Pointer to the AO attributes	Output



[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_ao.h
- Library file: libmpi.a

[Note]

- The obtained attributes are the attributes configured last time.
- If attributes are never configured, a code indicating failure is returned.

[Example]

```
HI_S32 s32ret;
AUDIO_DEV AoDevId = 0;
AIO_ATTR_S stAttr;

/* first enable ao device*/

s32ret = HI_MPI_AO_GetPubAttr(AoDevId, &stAttr);
if(HI_SUCCESS != s32ret)
{
    printf("get ao %d attr err:0x%x\n", AoDevId,s32ret);
    return s32ret;
}
```

[See Also]

None

HI_MPI_AO_Enable

[Description]

Enables an AO device.

[Syntax]

```
HI_S32 HI_MPI_AO_Enable(AUDIO_DEV AoDevId);
```

[Parameter]



Parameter	Description	Input/Output
AoDevId	AO device ID For details about the value range, see Table 9-6 .	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_ao.h
- Library file: libmpi.a

[Note]

- The attributes of the AO device must be configured before the AO device is enabled. Otherwise, an error indicating that attributes are not configured is returned.
- A code indicating success is returned if the AO device is enabled.

[Example]

See [HI_MPI_AO_SetPubAttr](#).

[See Also]

None

HI_MPI_AO_Disable

[Description]

Disables an AO device.

[Syntax]

```
HI_S32 HI_MPI_AO_Disable(AUDIO_DEV AoDevId);
```

[Parameter]

Parameter	Description	Input/Output
AoDevId	AO device ID For details about the value range, see Table 9-6 .	Input

[Return Value]



Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_ao.h
- Library file: libmpi.a

[Note]

- A code indicating success is returned if the AO device is disabled.
- All AO channels of the AO device must be disabled before the AO device is disabled.

[Example]

None

[See Also]

None

HI_MPI_AO_EnableChn

[Description]

Enables an AO channel.

[Syntax]

```
HI_S32 HI_MPI_AO_EnableChn(AUDIO_DEV AoDevId, AO_CHN AoChn);
```

[Parameter]

Parameter	Description	Input/Output
AoDevId	AO device ID For details about the value range, see Table 9-6 .	Input
AoChn	AO channel ID The ID range depends on the AO device attribute parameters u32ChnCnt (maximum number of channels) and enSoundmode (audio channel mode).	Input

[Return Value]

Return Value	Description
0	Success



Return Value	Description
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_ao.h
- Library file: libmpi.a

[Note]

Before the AO channel is enabled, the AO device on which the AI channel exists must be enabled. Otherwise, an error indicating that the device is not enabled is returned.

[Example]

See [HI_MPI_AO_SetPubAttr](#).

[See Also]

None

HI_MPI_AO_DisableChn

[Description]

Disables an AO channel.

[Syntax]

```
HI_S32 HI_MPI_AO_DisableChn(AUDIO_DEV AoDevId, AO_CHN AoChn);
```

[Parameter]

Parameter	Description	Input/Output
AoDevId	AO device ID For details about the value range, see Table 9-6 .	Input
AoChn	AO channel ID Value range: [0, AIO_MAX_CHN_NUM)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]



- Header files: hi_comm_aio.h, mpi_ao.h
- Library file: libmpi.a

[Note]

None

[Example]

None

HI_MPI_AO_SendFrame

[Description]

Transmits AO frames.

[Syntax]

```
HI_S32 HI_MPI_AO_SendFrame(AUDIO_DEV AoDevId, AO_CHN AoChn, const
AUDIO_FRAME_S *pstData, HI_S32 s32MilliSec);
```

[Parameter]

Parameter	Description	Input/Output
AoDevId	AO device ID For details about the value range, see Table 9-6 .	Input
AoChn	AO channel ID The ID range depends on the AO device attribute parameters u32ChnCnt (maximum number of channels) and enSoundmode (audio channel mode).	Input
pstData	Pointer to the structure of the audio frame	Input
s32MilliSec	Timeout period for sending data -1: block mode 0: non-block mode > 0: The interface is blocked for s32MilliSec ms. If a timeout occurs, an error is reported and the interface is returned.	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]



- Header files: hi_comm_aio.h, mpi_ai.h
- Library file: libmpi.a

[Note]

- HI_MPI_AO_SendFrame is used to transmit audio frames to the AO channel. If the AO channel is bound to the AI or ADEC channel by calling HI_MPI_SYS_Bind, do not call HI_MPI_AO_SendFrame.
- The value of **s32MilliSec** must be greater than or equal to **-1**. When **s32MilliSec** is **-1**, data is sent in block mode. When **s32MilliSec** is **0**, data is sent in non-block mode. When **s32MilliSec** is greater than **0**, the MPI is returned and an error is reported if there is no data after the MPI is blocked for **s32MilliSec** ms.
- Enable the corresponding AO channel before transmitting audio frames to the AO channel by calling this MPI.

[See Also]

None

HI_MPI_AO_EnableReSmp

[Description]

Enables AO resampling.

[Syntax]

```
HI_S32 HI_MPI_AO_EnableReSmp(AUDIO_DEV AoDevId, AO_CHN AoChn,  
AUDIO_SAMPLE_RATE_E enInSampleRate);
```

[Parameter]

Parameter	Description	Input/Output
AoDevId	AO device ID For details about the value range, see Table 9-6 .	Input
AoChn	AO channel ID The ID range depends on the maximum number of channels specified by the AO device attribute parameter u32ChnCnt .	Input
enInSampleRate	Input sampling rate for audio resampling	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."



[Requirement]

- Header files: hi_comm_aio.h, mpi_ao.h
- Library files: libmpi.a, libupvqe.a, libhive_RES.so, libhive_common.so

[Note]

- This MPI is called before an AO channel is bound and after an AO channel is enabled.
- Resampling can be repeatedly enabled. Ensure that the configured input sampling rate for resampling is the same as the previous input sampling rate.
- If you want to enable resampling after disabling an AO channel, you must enable this channel and call this MPI again.
- The input sampling rate for AO resampling must be different from that specified in AO device attributes.

[Example]

To set the resampling rate for the ADEC module to decode data to the AOU from 8 kHz to 32 kHz, see the following codes:

```
/* dev attr of ao */
AUDIO_SAMPLE_RATE_E enInSampleRate;
stAioAttr.u32ChnCnt = 2;
stAioAttr.enBitwidth = AUDIO_BIT_WIDTH_16;
stAioAttr.enSamplerate = AUDIO_SAMPLE_RATE_32000;
stAioAttr.enSoundmode = AUDIO_SOUND_MODE_MONO;
stAioAttr.u32EXFlag = 1;
stAioAttr.u32FrmNum = 30;
stAioAttr.u32PtNumPerFrm = 320*4;
enInSampleRate = AUDIO_SAMPLE_RATE_8000;
s32Ret = HI_MPI_AO_EnableReSmp(AoDev, AoChn, enInSampleRate);
if (HI_SUCCESS != s32Ret)
{
    printf("func(%s) line(%d): failed, s32Ret:0x%x\n", __FUNCTION__,
__LINE__, s32Ret);
    return s32Ret;
}
```

HI_MPI_AO_DisableReSmp

[Description]

Disables AO resampling.

[Syntax]

```
HI_S32 HI_MPI_AO_DisableReSmp(AUDIO_DEV AoDevId, AO_CHN AoChn);
```

[Parameter]



Parameter	Description	Input/Output
AoDevId	AO device ID For details about the value range, see Table 9-6 .	Input
AoChn	AO channel ID Value range: [0, AIO_MAX_CHN_NUM)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_ao.h
- Library files: libmpi.a, libupvqe.a, libhive_RES.so, libhive_common.so

[Note]

This MPI is called to disable AO resampling when resampling is not used any more.

[Example]

None

HI_MPI_AO_PauseChn

[Description]

Pauses an AO channel.

[Syntax]

```
HI_S32 HI_MPI_AO_PauseChn(AUDIO_DEV AoDevId, AO_CHN AoChn);
```

[Parameter]

Parameter	Description	Input/Output
AoDevId	AO device ID For details about the value range, see Table 9-6 .	Input
AoChn	AO channel ID The ID range depends on the AO device attribute parameters u32ChnCnt (maximum number of channels) and enSoundmode (audio channel mode).	Input



[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_ao.h
- Library file: libmpi.a

[Note]

- After an AO channel is paused, if the ADEC channel bound to the AO channel transmits audio frames to this channel, the frames are blocked; if the AI channel bound to the AO channel transmits audio frames to this channel, the frames are stored in the channel buffer when the channel buffer is not full or the frames are discarded when the channel buffer is full.
- When the AO channel is disabled, you are not allowed to call this MPI to pause the AO channel.

[Example]

None

[See Also]

None

HI_MPI_AO_ResumeChn

[Description]

Resumes an AO channel.

[Syntax]

```
HI_S32 HI_MPI_AO_ResumeChn(AUDIO_DEV AoDevId, AO_CHN AoChn);
```

[Parameter]

Parameter	Description	Input/Output
AoDevId	AO device ID For details about the value range, see Table 9-6 .	Input
AoChn	AO channel ID The ID range depends on the AO device attribute parameters u32ChnCnt (maximum number of channels) and enSoundmode (audio channel mode).	Input

[Return Value]



Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_ao.h
- Library file: libmpi.a

[Note]

- This MPI is called to resume a paused AO channel.
- If an AO channel is paused or enabled, calling this MPI returns a code indicating success. If an AO channel is in other states, calling this MPI returns an error.

[Example]

None

HI_MPI_AO_ClearChnBuf

[Description]

Clears the current audio data buffer on an AO channel.

[Syntax]

```
HI_S32 HI_MPI_AO_ClearChnBuf(AUDIO_DEV AoDevId, AO_CHN AoChn);
```

[Parameter]

Parameter	Description	Input/Output
AoDevId	AO device ID For details about the value range, see Table 9-6 .	Input
AoChn	AO channel ID The ID range depends on the AO device attribute parameters u32ChnCnt (maximum number of channels) and enSoundmode (audio channel mode).	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."



[Requirement]

- Header files: hi_comm_aio.h, mpi_ao.h
- Library file: libmpi.a

[Note]

- This MPI is called after the AO channel is enabled.
- This MPI must work with the [HI_MPI_ADEC_ClearChnBuf](#) MPI to completely clear the buffer data on the decoding channel.

[Example]

None

HI_MPI_AO_QueryChnStat

[Description]

Queries the status of the current audio data buffer on the AO channel.

[Syntax]

```
HI_S32 HI_MPI_AO_QueryChnStat(AUDIO_DEV AoDevId, AO_CHN AoChn,  
AO_CHN_STATE_S *pstStatus);
```

[Parameter]

Parameter	Description	Input/Output
AoDevId	AO device ID For details about the value range, see Table 9-6 .	Input
AoChn	AO channel ID The ID range depends on the AO device attribute parameters u32ChnCnt (maximum number of channels) and enSoundmode (audio channel mode).	Input
pstStatus	Pointer to the buffer status data structure	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_ao.h
- Library file: libmpi.a



[Note]

Call this MPI only after enabling the AO channel successfully.

[Example]

None

HI_MPI_AO_SetTrackMode

[Description]

Sets the track mode of an AO device.

[Syntax]

```
HI_S32 HI_MPI_AO_SetTrackMode(AUDIO_DEV AoDevId, AUDIO_TRACK_MODE_E enTrackMode);
```

[Parameter]

Parameter	Description	Input/Output
AoDevId	AO device ID For details about the value range, see Table 9-6 .	Input
enTrackMode	Track mode of an AO device	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_ao.h
- Library file: libmpi.a

[Note]

- Call this MPI after enabling an AO device successfully.
- The track mode can be set only when the AO device works in I²S mode.

[Example]

```
HI_S32 s32Ret;  
AUDIO_DEV AoDev = 0;  
AUDIO_TRACK_MODE_E enTrackMode = AUDIO_TRACK_NORMAL;  
AUDIO_TRACK_MODE_E temp;  
s32Ret = HI_MPI_AO_SetTrackMode(AoDev, enTrackMode);
```



```
if (HI_SUCCESS != s32Ret)
{
    printf("Ao set track mode failure! AoDev: %d, enTrackMode: %d, s32Ret:
0x%x.\n", AoDev, enTrackMode, s32Ret);
    return s32Ret;
}
s32Ret = HI_MPI_AO_GetTrackMode(AoDev, &temp);
if (HI_SUCCESS != s32Ret)
{
    printf("Ao get track mode failure! AoDev: %d, s32Ret: 0x%x.\n", AoDev,
s32Ret);
    return s32Ret;
}
```

HI_MPI_AO_GetTrackMode

[Description]

Obtains the track mode of an AO device.

[Syntax]

```
HI_S32 HI_MPI_AO_GetTrackMode(AUDIO_DEV AoDevId, AUDIO_TRACK_MODE_E
*penTrackMode);
```

[Parameter]

Parameter	Description	Input/Output
AoDevId	AO device ID For details about the value range, see Table 9-6 .	Input
penTrackMode	Pointer to the track mode of an AO device	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_ao.h
- Library file: libmpi.a

[Note]



- Call this MPI after enabling an AO device successfully.
- The track mode can be obtained only when the AO device works in I²S mode.

[Example]

None

HI_MPI_AO_SetVolume

[Description]

Sets the volume of an AO device.

[Syntax]

```
HI_S32 HI_MPI_AO_SetVolume(AUDIO_DEV AoDevId, HI_S32 s32VolumeDb);
```

[Parameter]

Parameter	Description	Input/Output
AoDevId	Audio device ID For details about the value range, see Table 9-6 .	Input
s32VolumeDb	Volume (in dB) Value range: [-121, +6]	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_ao.h
- Library file: libmpi.a

[Note]

Call this MPI only after enabling an AO device successfully.

[Example]

None

HI_MPI_AO_GetVolume

[Description]

Obtains the volume of an AO device.



[Syntax]

```
HI_S32 HI_MPI_AO_GetVolume(AUDIO_DEV AoDevId, HI_S32 *ps32VolumeDb);
```

[Parameter]

Parameter	Description	Input/Output
AoDevId	Audio device ID For details about the value range, see Table 9-6 .	Input
ps32VolumeDb	Pointer to the volume	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_ao.h
- Library file: libmpi.a

[Note]

Call this MPI only after enabling an AO device successfully.

[Example]

None

HI_MPI_AO_SetMute

[Description]

Sets the mute status of an AO device.

[Syntax]

```
HI_S32 HI_MPI_AO_SetMute(AUDIO_DEV AoDevId, HI_BOOL bEnable, AUDIO_FADE_S
*pstFade);
```

[Parameter]

Parameter	Description	Input/Output
AoDevId	AO device ID For details about the value range, see Table 9-6 .	Input



Parameter	Description	Input/Output
bEnable	AO device mute enable HI_TRUE: enabled HI_FALSE: disabled	Input
pstFade	Pointer to the fade-in/fade-out data structure	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_ao.h
- Library file: libmpi.a

[Note]

- Call this MPI after enabling an AO device successfully.
- You can use the fade-in/fade-out function when calling this MPI. If you do not want to use the fade-in/fade-out function, set **pstFade** to **null**.

[Example]

None

HI_MPI_AO_GetMute

[Description]

Obtains the mute status of an AO device.

[Syntax]

```
HI_S32 HI_MPI_AO_GetMute(AUDIO_DEV AoDevId, HI_BOOL *pbEnable,  
AUDIO_FADE_S *pstFade);
```

[Parameter]

Parameter	Description	Input/Output
AoDevId	AO device ID For details about the value range, see Table 9-6 .	Input
pbEnable	Pointer to the mute status of an AO device	Output
pstFade	Pointer to the fade-in/fade-out data structure	Output



[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_ao.h
- Library file: libmpi.a

[Note]

Call this MPI after enabling an AO device successfully.

[Example]

None

HI_MPI_AO_GetFd

[Description]

Obtains the device FD of an AO channel.

[Syntax]

```
HI_S32 HI_MPI_AO_GetFd(AUDIO_DEV AoDevId, AO_CHN AoChn)
```

[Parameter]

Parameter	Description	Input/Output
AoDevId	AO device ID For details about the value range, see Table 9-6 .	Input
AoChn	AO channel ID Value range: [0, AIO_MAX_CHN_NUM)	Input

[Return Value]

Return Value	Description
Positive value	Valid
Non-positive value	Invalid

[Requirement]



- Header files: hi_comm_aio.h, mpi_ao.h
- Library file: libmpi.a

[Note]

None

[Example]

```
HI_S32 s32AoFd;  
HI_S32 s32ret;  
AUDIO_DEV AoDevId = 0;  
AO_CHN AoChn = 0;  
  
/* first enable ao device */
```

```
s32AoFd = HI_MPI_AO_GetFd(AoDevId, AoChn);  
if(s32AoFd <= 0)  
{  
    return HI_FAILURE;  
}
```

[See Also]

None

HI_MPI_AO_SaveFile

[Description]

Enables the function of saving AO files.

[Syntax]

```
HI_S32 HI_MPI_AO_SaveFile(AUDIO_DEV AoDevId, AO_CHN AoChn,  
AUDIO_SAVE_FILE_INFO_S* pstSaveFileInfo);
```

[Parameter]

Parameter	Description	Input/Output
AoDevId	AO device ID For details about the value range, see Table 9-6 .	Input
AoChn	AO channel ID Value range: [0, AIO_MAX_CHN_NUM)	Input
pstSaveFileInfo	Pointer to the attribute structure of the function of saving AO files	Input

[Return Value]



Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_ao.h
- Library file: libmpi.a

[Note]

When the VQE function is disabled or the AI channel is bound to the AO channel in system binding mode, AO data fails to be dumped by calling this MPI. After this MPI is called, two files with specified size are generated in the specified folder. **Sin.pcm** is the input frame before VQE processing, and **Sou.pcm** is the output frame after VQE processing.

[See Also]

None

HI_MPI_AO_QueryFileStatus

[Description]

Queries whether the AO channel is in the state of saving files.

[Syntax]

```
HI_S32 HI_MPI_AO_QueryFileStatus(AUDIO_DEV AoDevId, AO_CHN AoChn,  
AUDIO_FILE_STATUS_S* pstFileStatus);
```

[Parameter]

Parameter	Description	Input/Output
AoDevId	AO device ID For details about the value range, see Table 9-6 .	Input
AoChn	AO channel ID Value range: [0, AIO_MAX_CHN_NUM)	Input
pstFileStatus	Pointer to the state attribute structure	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."



[Requirement]

- Header files: hi_comm_aio.h, mpi_ao.h
- Library file: libmpi.a

[Note]

This MPI is used to query whether the AO channel is in the state of saving files. After [HI_MPI_AO_SaveFile](#) is called to save a file, you can call this MPI to query whether the file to be saved reaches the specified size.

- If **bSaving** in **pstFileStatus** is **HI_TRUE**, the specified size is not reached.
- If **bSaving** in **pstFileStatus** is **HI_FALSE**, the specified size is reached.

[See Also]

None

HI_MPI_AO_ClrPubAttr

[Description]

Clears the public attributes of an AO device.

[Syntax]

```
HI_S32 HI_MPI_AO_ClrPubAttr(AUDIO_DEV AoDevId);
```

[Parameter]

Parameter	Description	Input/Output
AoDevId	AO device ID For details about the value range, see Table 9-6 .	Input

[Return Value]

Return Value	Description
Positive value	Valid
Non-positive value	Invalid

[Requirement]

- Header files: hi_comm_aio.h, mpi_ao.h
- Library file: libmpi.a

[Note]

- Before the device attributes are cleared, ensure that the device is stopped.
- When the AI device shares the clock of the AO device and both the AI device and AO device are disabled, the attribute of the AI device needs to be forcibly switched (the



modified attribute does not meet the requirement for sharing the clock: **enSamplerate** and **u32ClkChnCnt x** (Sampling bit width corresponding to **enBitwidth**) are the same for the AI and AO devices). It is recommended that this MPI be called to clear the internal coupling relationship with the AO before switching.

[See Also]

None

HI_MPI_AO_SetVqeAttr

[Description]

Sets the AO VQE attributes.

[Syntax]

```
HI_S32 HI_MPI_AO_SetVqeAttr(AUDIO_DEV AoDevId, AO_CHN AoChn,  
AO_VQE_CONFIG_S *pstVqeConfig);
```

[Parameter]

Parameter	Description	Input/Output
AoDevId	AO device ID For details about the value range, see Table 9-6 .	Input
AoChn	AO channel ID Value range: [0, AIO_MAX_CHN_NUM)	Input
pstVqeConfig	Pointer to the AO VQE attributes	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_ao.h
- Library files: libmpi.a, libdnvqe.a, libhive_common.so, libhive_HPF.so, libhive_EQ.so, libhive_ANR.so, libhive_AGC.so

[Note]

- Set the VQE attributes of the corresponding AO channel before enabling the VQE functions.
- Enable the corresponding AO channel before configuring AO VQE attributes.



- The VQE attributes of an AO channel cannot be dynamically configured. Therefore, you need to disable the VQE functions of an AO channel before reconfiguring the VQE attributes.
- You can enable some or all VQE functions by setting the corresponding VQE attributes.

[See Also]

None

HI_MPI_AO_GetVqeAttr

[Description]

Obtains the AO VQE attributes.

[Syntax]

```
HI_S32 HI_MPI_AO_GetVqeAttr(AUDIO_DEV AoDevId, AO_CHN AoChn,  
AO_VQE_CONFIG_S *pstVqeConfig);
```

[Parameter]

Parameter	Description	Input/Output
AoDevId	AO device ID For details about the value range, see Table 9-6 .	Input
AoChn	AO channel ID Value range: [0, AIO_MAX_CHN_NUM)	Input
pstVqeConfig	Pointer to the AO VQE attributes	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_ao.h
- Library files: libmpi.a, libdnvqe.a, libhive_common.so, libhive_HPF.so, libhive_EQ.so, libhive_ANR.so, libhive_AGC.so

[Note]

Set the VQE attributes of the corresponding AO channel before obtaining them.

[See Also]

None



HI_MPI_AO_EnableVqe

[Description]

Enables AO VQE.

[Syntax]

```
HI_S32 HI_MPI_AO_EnableVqe(AUDIO_DEV AoDevId, AO_CHN AoChn);
```

[Parameter]

Parameter	Description	Input/Output
AoDevId	AO device ID For details about the value range, see Table 9-6 .	Input
AoChn	AO channel ID Value range: [0, AIO_MAX_CHN_NUM)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_ao.h
- Library files: libmpi.a, libdnvqe.a, libhive_common.so, libhive_HPF.so, libhive_EQ.so, libhive_ANR.so, libhive_AG.so

[Note]

- Enable the corresponding AO channel before enabling VQE.
- If the VQE functions of an AO channel are enabled for multiple times, a code indicating success is returned.
- After an AO channel is disabled, if you want to enable it again and use the VQE functions, you need to call this MPI to enable AO VQE again.

[See Also]

None

HI_MPI_AO_DisableVqe

[Description]

Disables AO VQE.

[Syntax]



```
HI_S32 HI_MPI_AO_DisableVqe(AUDIO_DEV AoDevId, AO_CHN AoChn);
```

[Parameter]

Parameter	Description	Input/Output
AoDevId	AO device ID For details about the value range, see Table 9-6 .	Input
AoChn	AO channel ID Value range: [0, AIO_MAX_CHN_NUM)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_ao.h
- Library files: libmpi.a, libdnvqe.a, libhive_common.so, libhive_HPF.so, libhive_EQ.so, libhive_ANR.so, libhive_AG.so

[Note]

- If AO VQE is not used any more, call this MPI to disable it.
- If the VQE functions of an AO channel are disabled for multiple times, a code indicating success is returned.

[See Also]

None

9.3.3 AENC

An AENC module mainly provides the following functions: creating AENC channels, transmitting audio frames, and obtaining encoded streams.

The AENC module provides the following MPIS:

- [HI_MPI_AENC_CreateChn](#): Creates an AENC channel.
- [HI_MPI_ADEC_DestroyChn](#): Destroys an AENC channel.
- [HI_MPI_AENC_SendFrame](#): Transmits audio frames for encoding.
- [HI_MPI_AENC_GetStream](#): Obtains AENC streams.
- [HI_MPI_AENC_ReleaseStream](#): Releases AENC streams.
- [HI_MPI_AENC_GetFd](#): Obtains the device FD corresponding to an AENC channel ID.
- [HI_MPI_AENC_RegeisterEncoder](#): Registers an encoder.



- [HI_MPI_AENC_UnRegisterEncoder](#): Deregisters an encoder.
- [HI_MPI_AENC_SaveFile](#): Enables the function of saving AENC files before the encoding channel starts encoding.
- [HI_MPI_AENC_QueryFileStatus](#): Queries whether the AENC channel is in the state of saving files.

HI_MPI_AENC_CreateChn

[Description]

Creates an AENC channel.

[Syntax]

```
HI_S32 HI_MPI_AENC_CreateChn(AENC_CHN AeChn, const AENC_CHN_ATTR_S *pstAttr);
```

[Parameter]

Parameter	Description	Input/Output
AeChn	AENC channel ID Value range: [0, AENC_MAX_CHN_NUM)	Input
pstAttr	Pointer to the AENC channel attributes	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, hi_comm_aenc.h, mpi_aenc.h
- Library files: libmpi.a, lib_VoiceEngine.a, libupvqe.a

[Note]

- **enType** specifies an encoding protocol for the AENC channel. Currently, G711, G726, and ADPCM are supported. For details, see [Table 9-4](#).
- The protocols listed in [Table 9-4](#) support only 16-bit linear PCM audio data. If the input data is 8-bit sampling precision data, the AENC module extends the data to 16 bits. You are advised to set the extension flag to 1 when configuring the general attributes of the AI device. In this way, AI data is automatically extended from 8 bits to 16 bits.
- [Table 9-5](#) describes the structure of HiSilicon voice frames.
- Some attributes of the AENC channel must match attributes of input audio data, for example, sampling rate and duration of a frame (number of sampling points per frame).



- The buffer size is in unit of frame and its value range is [2, **MAX_AUDIO_FRAME_NUM**]. You are advised to set the size to **10** or larger. A small buffer may cause exceptions such as frame loss.
- This MPI can be called when the channel ID is not allocated. Otherwise, an error indicating that the channel is created is returned.

[Example]

```
HI_S32 s32ret;
AENC_CHN_ATTR_S stAencAttr;
ADEC_ATTR_ADPCM_S stAdpcmAenc;
AI_DEV AiDev = 0;
AI_CHN AiChn = 0;
AENC_CHN AencChn = 0;
AUDIO_FRAME_S stAudioFrm;
AUDIO_STREAM_S stAudioStream;
MPP_CHN_S stSrcChn, stDestChn;

stAencAttr.enType = PT_ADPCM; /* ADPCM */
stAencAttr.u32BufSize = 8;
stAencAttr.pValue = &stAdpcmAenc;
stAdpcmAenc.enADPCMTYPE = ADPCM_TYPE_DVI4;

/* create aenc chn*/
s32ret = HI_MPI_AENC_CreateChn(AencChn, &stAencAttr);
if (HI_SUCCESS != s32ret)
{
    printf("create aenc chn %d err:0x%08x\n", AencChn, s32ret);
    return s32ret;
}

/* bind AENC to AI channel */
stSrcChn.enModId = HI_ID_AI;
stSrcChn.s32DevId = AiDev;
stSrcChn.s32ChnId = AiChn;

stDestChn.enModId = HI_ID_AENC;
stDestChn.s32DevId = 0;
stDestChn.s32ChnId = AencChn;
s32Ret = HI_MPI_SYS_Bind(&stSrcChn, &stDestChn);
if (s32Ret != HI_SUCCESS)
{
    return s32Ret;
}

/* get stream from aenc chn */
```



```
s32ret = HI_MPI_AENC_GetStream(AencChn, &stAudioStream);
if (HI_SUCCESS != s32ret)
{
    printf("get stream from aenc chn %d fail \n", AencChn);
    return s32ret;
}

/* deal with audio stream */

/* release audio stream */
s32ret = HI_MPI_AENC_ReleaseStream(AencChn, &stAudioStream);
if (HI_SUCCESS != s32ret )
{
    return s32ret;
}

/* destroy aenc chn */
s32ret = HI_MPI_AENC_DestroyChn(AencChn) ;
if (HI_SUCCESS != s32ret )
{
    return s32ret;
}
```

[See Also]

None

HI_MPI_AENC_DestroyChn

[Description]

Destroys an AENC channel.

[Syntax]

```
HI_S32 HI_MPI_AENC_DestroyChn(AENC_CHN AeChn);
```

[Parameter]

Parameter	Description	Input/Output
AeChn	AENC channel ID Value range: [0, AENC_MAX_CHN_NUM)	Input

[Return Value]

Return Value	Description
0	Success



Return Value	Description
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, hi_comm_aenc.h, mpi_aenc.h
- Library files: libmpi.a, lib_VoiceEngine.a, libupvqe.a

[Note]

- If no channel is created, calling this MPI returns a code indicating success.
- A code indicating failure is returned if users attempt to destroy an AENC channel on which streams are being obtained/released or frames are being sent. Users cannot destroy such an AENC channel.

[Example]

See [HI_MPI_AENC_CreateChn](#).

[See Also]

None

HI_MPI_AENC_SendFrame

[Description]

Transmits audio frames for encoding.

[Syntax]

```
HI_S32 HI_MPI_AENC_SendFrame(AENC_CHN AeChn, const AUDIO_FRAME_S *pstFrm,  
const AEC_FRAME_S *pstAecFrm);
```

[Parameter]

Parameter	Description	Input/Output
AeChn	AENC channel ID Value range: [0, AENC_MAX_CHN_NUM)	Input
pstFrm	Pointer to the structure of the audio frame	Input
pstAecFrm	Pointer to the structure of the reference frame for echo cancellation	Input

[Return Value]

Return Value	Description
0	Success



Return Value	Description
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, hi_comm_aenc.h, mpi_aenc.h
- Library files: libmpi.a, libupvqe.a

[Note]

- To disable echo cancellation, set pstAecFrm to null.
- This MPI is a non-block interface. If the audio stream buffer is full, an error code indicating failure is returned.
- HI_MPI_AENC_SendFrame is used to transmit audio frames for encoding. If the AENC channel is bound to the AI channel by calling HI_MPI_SYS_Bind, do not call HI_MPI_AENC_SendFrame.
- Create the corresponding AENC channel before transmitting audio frames for encoding by calling this MPI.

[Example]

None

[See Also]

None

HI_MPI_AENC_GetStream

[Description]

Obtains encoded streams.

[Syntax]

```
HI_S32 HI_MPI_AENC_GetStream(AENC_CHN AeChn, AUDIO_STREAM_S*pstStream,  
HI_S32 s32MilliSec);
```

[Parameter]

Parameter	Description	Input/Output
AeChn	AENC channel ID Value range: [0, AENC_MAX_CHN_NUM)	Input
pstStream	Obtained audio streams	Output



Parameter	Description	Input/Output
s32MilliSec	Timeout period for obtaining data -1: block mode. The wait is infinite if there is no data. 0: non-block mode. An error is reported and the MPI is returned if there is no data. > 0: The MPI is blocked for s32MilliSec ms. If a timeout occurs, an error is reported and the MPI is returned.	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, hi_comm_aenc.h, mpi_aenc.h
- Library file: libmpi.a

[Note]

- Streams can be obtained after an AENC channel is created. Otherwise, a code indicating failure is returned. If users attempt to destroy an AENC channel on which streams are being obtained, a code indicating failure is returned.
- The value of **s32MilliSec** must be greater than or equal to -1. When **s32MilliSec** is -1, data is obtained in block mode. When **s32MilliSec** is 0, data is obtained in non-block mode. When **s32MilliSec** is greater than 0, the MPI is returned and an error is reported if there is no data after the MPI is blocked for **s32MilliSec** ms.
- To directly obtain raw AI data, create an AENC channel, set **enType** to **PT_LPCM**, and bind the AENC channel. Audio data along this AENC channel is the raw AII data.

[Example]

See [HI_MPI_AENC_CreateChn](#).

[See Also]

None

HI_MPI_AENC_ReleaseStream

[Description]

Releases streams obtained from an AENC channel.

[Syntax]

```
HI_S32 HI_MPI_AENC_ReleaseStream(AENC_CHN AeChn, const AUDIO_STREAM_S
```



```
*pstStream);
```

[Parameter]

Parameter	Description	Input/Output
AeChn	AENC channel ID Value range: [0, AENC_MAX_CHN_NUM)	Input
pstStream	Pointer to the obtained streams	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, hi_comm_aenc.h, mpi_aenc.h
- Library files: libmpi.a, lib_VoiceEngine.a

[Note]

- You are advised to release streams after use. If the streams are not released in time, encoding may be blocked.
- Streams must be released from the channel on which the streams are obtained. Modification of stream structure is not allowed. Otherwise, streams cannot be released, and the stream buffer is lost and even a program exception occurs.
- Streams can be released after an AENC channel is created. Otherwise, a code indicating failure is returned. If users attempt to destroy an AENC channel on which streams are being released, a code indicating failure is returned.

[Example]

See [HI_MPI_AENC_CreateChn](#).

[See Also]

None

HI_MPI_AENC_GetFd

[Description]

Obtains the device FD corresponding to an AENC channel ID.

[Syntax]

```
HI_S32 HI_MPI_AENC_GetFd(AENC_CHN AeChn)
```

[Parameter]



Parameter	Description	Input/Output
AeChn	AENC channel ID Value range: [0, AENC_MAX_CHN_NUM)	Input

[Return Value]

Return Value	Description
Positive value	Valid
Non-positive value	Invalid

[Requirement]

- Header files: hi_comm_aio.h, hi_comm_aenc.h, mpi_aenc.h
- Library file: libmpi.a

[Note]

None

HI_MPI_AENC_RegeisterEncoder

[Description]

Registers an encoder.

[Syntax]

```
HI_S32 HI_MPI_AENC_RegeisterEncoder(HI_S32 *ps32Handle, AENC_ENCODER_S *pstEncoder)
```

[Parameter]

Parameter	Description	Input/Output
ps32Handle	Registration handle	Output
pstEncoder	Structure defining the encoder attributes	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."



[Requirement]

- Header files: hi_comm_aenc.h, mpi_aenc.h
- Library file: libmpi.a

[Note]

- You can register an encoder with the AENC module by sending the encoder attribute structure to this module, and the registration handle is returned. You can deregister this encoder by using the registration handle.
- You can register a maximum of 20 encoders (including the five encoders LPCM, G711a, G711u, G726, and ADPCM registered with the AENC module) with this module.
- For example, you are not allowed to register another G726 encoder if you have registered one.

[Example]

None

[See Also]

None

HI_MPI_AENC_UnRegisterEncoder

[Description]

Deregisters an encoder.

[Syntax]

```
HI_S32 HI_MPI_AENC_UnRegisterEncoder(HI_S32 s32Handle)
```

[Parameter]

Parameter	Description	Input/Output
s32Handle	Registration handle returned when an encoder is registered	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aenc.h, mpi_aenc.h
- Library file: libmpi.a

[Note]



Typically, encoders do not need to be deregistered.

[Example]

None

[See Also]

None

HI_MPI_AENC_SaveFile

[Description]

Enables the function of saving AENC files before the encoding channel starts encoding.

[Syntax]

```
HI_S32 HI_MPI_AENC_SaveFile(AENC_CHN AeChn, AUDIO_SAVE_FILE_INFO_S  
*pstSaveFileInfo);
```

[Parameter]

Parameter	Description	Input/Output
AeChn	AENC channel ID Value range: [0, AENC_MAX_CHN_NUM)	Input
pstSaveFileInfo	Pointer to the attribute structure of the function of saving AENC files	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_aenc.h
- Library file: libmpi.a

[Note]

This MPI is used only to dump audio data files before and after VQE processing and before encoding when the AI channel is bound to the AENC channel in system binding mode. If the AI channel is bound to the AENC channel in non-system-binding mode or VQE is disabled, encoding channel data fails to be dumped by calling this MPI. After this MPI is called, three files with specified sizes are generated in the specified folder. **Sin.pcm** is the input frame before VQE processing, **Rin.pcm** is the echo cancellation reference frame before VQE processing, and **Sou.pcm** is the output frame after VQE processing.



[See Also]

None

HI_MPI_AENC_QueryFileStatus

[Description]

Queries whether the AENC channel is in the state of saving files.

[Syntax]

```
HI_S32 HI_MPI_AENC_QueryFileStatus(AENC_CHN AeChn, AUDIO_FILE_STATUS_S*  
pstFileStatus);
```

[Parameter]

Parameter	Description	Input/Output
AeChn	AENC channel ID Value range: [0, AENC_MAX_CHN_NUM)	Input
pstFileStatus	Pointer to the state attribute structure	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, mpi_aenc.h
- Library file: libmpi.a

[Note]

This MPI is used to query whether the AENC channel is in the state of saving files. After [HI_MPI_AENC_SaveFile](#) is called to save a file, you can call this MPI to query whether the file to be saved reaches the specified size.

- If **bSaving** in **pstFileStatus** is **HI_TRUE**, the specified size is not reached.
- If **bSaving** in **pstFileStatus** is **HI_FALSE**, the specified size is reached.

[See Also]

None



9.3.4 ADEC

An ADEC module mainly provides the following functions: creating ADEC channels, transmitting audio frames, and obtaining decoded audio frame streams.

The ADEC module provides the following MPIs:

- [HI_MPI_ADEC_CreateChn](#): Creates an ADEC channel.
- [HI_MPI_ADEC_DestroyChn](#): Destroys an ADEC channel.
- [HI_MPI_ADEC_SendStream](#): Sends audio streams to an ADEC channel.
- [HI_MPI_ADEC_ClearChnBuf](#): Clears the current audio data buffer on an ADEC channel.
- [HI_MPI_ADEC_RegeisterDecoder](#): Registers a decoder.
- [HI_MPI_ADEC_UnRegisterDecoder](#): Deregisters a decoder.
- [HI_MPI_ADEC_GetFrame](#): Obtains the decoded audio frame data.
- [HI_MPI_ADEC_ReleaseFrame](#): Releases the buffer for storing the decoded audio frame data.
- [HI_MPI_ADEC_SendEndOfStream](#): Sends the stream end identifier to the decoder and clears the stream buffer.

HI_MPI_ADEC_CreateChn

[Description]

Creates an ADEC channel.

[Syntax]

```
HI_S32 HI_MPI_ADEC_CreateChn(ADEC_CHN AdChn, ADEC_CHN_ATTR_S *pstAttr);
```

[Parameter]

Parameter	Description	Input/Output
AdChn	ADEC channel ID Value range: [0, ADEC_MAX_CHN_NUM)	Input
pstAttr	Pointer to the ADEC channel attributes	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, hi_comm_apec.h, mpi_apec.h



- Library files: libmpi.a, lib_VoiceEngine.a

[Note]

- **enType** specifies a decoding protocol for the ADEC channel. Currently, G711, G726, and ADPCM are supported.
- Some attributes of the ADEC channel must be consistent with attributes of the AO device, for example, sampling rate and length of a frame (number of sampling points per frame).
- The buffer size is in unit of frame and its value range is [2, **MAX_AUDIO_FRAME_NUM**]. You are advised to set the size to **10** or greater. A small buffer may cause exceptions such as frame loss.
- This MPI can be called when the channel is not created or it is destroyed. If an ADEC channel is created, an error indicating that the channel is created is returned.

[Example]

```
HI_S32 s32ret;
ADEC_CHN_ATTR_S stAdecAttr;
ADEC_ATTR_ADPCM_S stAdpcm;
ADEC_CHN AdChn = 0;
AUDIO_STREAM_S stAudioStream;
AUDIO_DEV AoDev = 0;
AO_CHN AoChn = 0;
MPP_CHN_S stSrcChn;
MPP_CHN_S stDestChn;

stAdecAttr.enType = PT_ADPCM;
stAdecAttr.u32BufSize = 8;
stAdecAttr.enMode = ADEC_MODE_STREAM;
stAdecAttr.pValue = &stAdpcm;
stAdpcm.enADPCMType = ADPCM_TYPE_DVI4;

/* create adec chn*/
s32ret = HI_MPI_ADEC_CreateChn(AdChn, &stAdecAttr);
if (s32ret)
{
    printf("create adec chn %d err:0x%x\n", AdChn,s32ret);
    return s32ret;
}

/* bind ADEC to AO channel*/
stSrcChn.enModId = HI_ID_ADEC;
stSrcChn.s32DevId = 0;
stSrcChn.s32ChnId = AdChn;
stDestChn.enModId = HI_ID_AO;
stDestChn.s32DevId = AoDev;
stDestChn.s32ChnId = AoChn;
```



```
s32ret = HI_MPI_SYS_Bind(&stSrcChn, &stDestChn);  
if (s32ret)  
{  
    printf("bind adec chn %d to ao(%d, %d) error:0x%x\n", AdChn, AoDev,  
    AoChn, s32ret);  
    return s32ret;  
}  
  
/* get audio stream from network or file*/  
  
/* send audio stream to adec chn */  
s32ret = HI_MPI_ADEC_SendStream(AdChn, &stAudioStream);  
if (s32ret)  
{  
    printf("send stream to adec fail\n");  
    return s32ret;  
}  
  
/* destroy adec chn */  
s32ret = HI_MPI_ADEC_DestroyChn(AdChn);  
if (HI_SUCCESS != s32ret)  
{  
    return s32ret;  
}
```

[See Also]

None

HI_MPI_ADEC_DestroyChn

[Description]

Destroys an ADEC channel.

[Syntax]

```
HI_S32 HI_MPI_ADEC_DestroyChn(ADEC_CHN AdChn);
```

[Parameter]

Parameter	Description	Input/Output
AdChn	ADEC channel ID Value range: [0, ADEC_MAX_CHN_NUM)	Input

[Return Value]



Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, hi_comm_adec, mpi_adec.h
- Library files: libmpi.a, lib_VoiceEngine.a

[Note]

- If no channel is created, calling this MPI returns a code indicating success.
- A code indicating failure is returned if users attempt to destroy an ADEC channel on which streams are being obtained/released or frames are being sent.

[Example]

See [HI_MPI_ADEC_CreateChn](#).

[See Also]

None

HI_MPI_ADEC_SendStream

[Description]

Sends streams to an ADEC channel.

[Syntax]

```
HI_S32 HI_MPI_ADEC_SendStream(ADEC_CHN AdChn, const AUDIO_STREAM_S
*pstStream, HI_BOOL bBlock);
```

[Parameter]

Parameter	Description	Input/Output
AdChn	ADEC channel ID Value range: [0, ADEC_MAX_CHN_NUM)	Input
pstStream	Audio streams	Input
bBlockFlag	Block flag Value range: HI_TRUE: block HI_FALSE: non-block	Input

[Return Value]



Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, hi_comm_adec.h, mpi_adec.h
- Library files: libmpi.a, lib_VoiceEngine.a

[Note]

- Specify the decoding mode to Pack or Stream when creating an ADEC channel.
 - The Pack mode is used when users determine that a stream is exactly one frame, for example, a stream directly obtained from the AENC channel or the stream of a frame (the frame boundary is easily determined because the length of the voice encoding stream is fixed) in a file. This mode provides high decoding efficiency.
 - The Stream mode is used when users cannot determine whether a stream is one frame. In this mode, efficiency is low and delay may occur.
- The LPCM supports only the Pack mode. Other audio formats support the two decoding modes.
- Streams can be sent after an ADEC channel is created. Otherwise, a code indicating failure is returned. If users attempt to destroy an ADEC channel on which streams are being sent, a code indicating failure is returned.
- Streams can be sent in block or non-block mode.
- When streams are sent in block mode, if the buffer caching decoded audio frames is full, this MPI fails to be called. This MPI can be successfully called only after the decoded audio frames are released or the ADEC channel is destroyed.
- Make sure that the stream data on the ADEC channel is correct. Otherwise, the decoder may exit exceptionally.

[Example]

See [HI_MPI_ADEC_CreateChn](#).

[See Also]

None

HI_MPI_ADEC_ClearChnBuf

[Description]

Clears the current audio data buffer on an ADEC channel.

[Syntax]

```
HI_S32 HI_MPI_ADEC_ClearChnBuf (ADEC_CHN AdChn);
```

[Parameter]



Parameter	Description	Input/Output
AdChn	ADEC channel ID Value range: [0, ADEC_MAX_CHN_NUM)	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_aio.h, hi_comm_adec.h, mpi_adec.h
- Library file: libmpapi.a

[Note]

- This MPI can be called after an ADEC channel is created. Otherwise, a code indicating that the channel does not exist is returned.
- When this MPI is called, the Stream mode is not recommended for clearing buffer data. In Stream mode, ensure that data transmitted to the decoder comprises a stream containing a complete frame after clearing the buffer. Otherwise, the decoder may operate exceptionally.
- No matter whether the data is decoded in Stream mode, users must ensure that transmitting data to be decoded must be synchronous with clearing the buffer.

[Example]

None

[See Also]

None

HI_MPI_ADEC_RegeisterDecoder

[Description]

Registers a decoder.

[Syntax]

```
HI_S32 HI_MPI_ADEC_RegeisterDecoder(HI_S32 *ps32Handle, ADEC_DECODER_S  
*pstDecoder);
```

[Parameter]



Parameter	Description	Input/Output
ps32Handle	Registration handle	Output
pstDecoder	Structure defining the encoder attributes	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_adec.h, mpi_adec.h
- Library file: libmpi.a

[Note]

- You can register a decoder with the ADEC module by sending the decoder attribute structure to this module, and the registration handle is returned. You can deregister this decoder by using the registration handle.
- You can register a maximum of 20 decoders (including the five decoders LPCM, G711a, G711μ, G726, and ADPCM registered with the ADEC module) with this module.
- For example, you are not allowed to register another G726 decoder if you have registered one.

[Example]

None

[See Also]

None

HI_MPI_ADEC_UnRegisterDecoder

[Description]

Deregisters a decoder.

[Syntax]

```
HI_S32 HI_MPI_ADEC_UnRegisterDecoder(HI_S32 s32Handle);
```

[Parameter]

Parameter	Description	Input/Output
s32Handle	Registration handle returned when a decoder is registered	Input



[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header files: hi_comm_adec.h, mpi_adec.h
- Library file: libmpi.a

[Note]

Typically, decoders do not need to be deregistered.

[Example]

None

[See Also]

None

HI_MPI_ADEC_GetFrame

[Description]

Obtains the decoded audio frame data.

[Syntax]

```
HI_S32 HI_MPI_ADEC_GetFrame(ADEC_CHN AdChn, AUDIO_FRAME_INFO_S  
*pstFrmInfo, HI_BOOL bBlock);
```

[Parameter]

Parameter	Description	Input/Output
AdChn	ADEC channel ID	Input
pstFrmInfo	Structure of the audio frame data	Output
bBlock	Whether the decoded audio frame data is obtained in block mode	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."



[Requirement]

- Header files: hi_comm_adec.h, mpi_adec.h
- Library file: libmpi.a

[Note]

- This MPI can be called only after an ADEC channel is created.
- When the decoded frame data is obtained by calling this MPI, it is recommended that streams be sent by frame.
- If streams are sent by stream when the decoded frame data is obtained by calling this MPI, ensure that the decoded frame data is obtained in a timely manner. Otherwise, exceptions may occur.
- When the decoded frame data is obtained by calling this MPI, unbind the ADEC channels from the AO channels. Otherwise, the obtained frames are inconsecutive.

[Example]

None

[See Also]

None

HI_MPI_ADEC_ReleaseFrame

[Description]

Releases the buffer for storing the decoded audio frame data.

[Syntax]

```
HI_S32 HI_MPI_ADEC_ReleaseFrame(ADEC_CHN AdChn, AUDIO_FRAME_INFO_S
*pstFrmInfo);
```

[Parameter]

Parameter	Description	Input/Output
AdChn	ADEC channel ID	Input
pstFrmInfo	Structure of the audio frame data	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]



- Header files: hi_comm_adec.h, mpi_adec.h
- Library file: libmpi.a

[Note]

- This MPI can be called only after an ADEC channel is created.
- This MPI must work with [HI_MPI_ADEC_GetFrame](#).

[Example]

None

[See Also]

None

HI_MPI_ADEC_SendEndOfStream

[Description]

Sends the stream end identifier to the decoder and clears the stream buffer.

[Syntax]

```
HI_S32 HI_MPI_ADEC_SendEndOfStream (ADEC_CHN AdChn, HI_BOOL bInstant);
```

[Parameter]

Parameter	Description	Input/Output
AdChn	ADEC channel ID	Input
bInstant	Whether to clear the cached data in the decoder immediately <ul style="list-style-type: none">• HI_FALSE: no. The cached data in the decoder is not cleared and the decoding continues until the remaining buffer space is insufficient for storing a frame of data.• HI_TRUE: yes	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 9.5 "Error Codes."

[Requirement]

- Header file: mpi_adec.h
- Library file: libmpi.a

[Note]



None

[Example]

None

9.3.5 Embedded audio Codec

The embedded audio codec supports the operations of the hardware device by using the ioctl function. You do not need to call cmds of the ioctl function. For the cmds that are not called, use the default configurations when loading modules. When the ioctl function is called, the registers of the audio codec are read or written.

9.3.5.1 Common Commands

The following describes common commands:

- [ACODEC_SOFT_RESET_CTRL](#): Restores the audio codec to default settings.
- [ACODEC_SET_I2S1_FS](#): Sets the sampling rate of the I²S1 interface.
- [ACODEC_SET_INPUT_VOL](#): Sets the total input volume.
- [ACODEC_GET_INPUT_VOL](#): Obtains the total input volume.
- [ACODEC_SET_OUTPUT_VOL](#): Sets the total output volume.
- [ACODEC_GET_OUTPUT_VOL](#): Obtains the total output volume.
- [ACODEC_SET_MIXER_MIC](#): Selects the input mode.
- [ACODEC_SET_GAIN_MICL](#): Sets the analog gain of the audio-left input channel.
- [ACODEC_SET_GAIN_MICR](#): Sets the analog gain of the audio-right input channel.
- [ACODEC_SET_DACL_VOL](#): Sets the volume of the audio-left output channel.
- [ACODEC_SET_DACR_VOL](#): Sets the volume of the audio-right output channel.
- [ACODEC_SET_ADCL_VOL](#): Sets the volume of the audio-left input channel.
- [ACODEC_SET_ADCR_VOL](#): Sets the volume of the audio-right input channel.
- [ACODEC_SET_MICL_MUTE](#): Sets the mute of the audio-left input channel.
- [ACODEC_SET_MICR_MUTE](#): Sets the mute of the audio-right input channel.
- [ACODEC_SET_DACL_MUTE](#): Set the mute of the audio-left output channel.
- [ACODEC_SET_DACR_MUTE](#): Sets the mute of the audio-right output channel.
- [ACODEC_DAC_SOFT_MUTE](#): Controls the soft mute function of the DAC.
- [ACODEC_DAC_SOFT_UNMUTE](#): Controls the soft unmute function of the DAC.
- [ACODEC_ENABLE_BOOSTL](#): Controls the boost analog gain of the audio-left channel.
- [ACODEC_ENABLE_BOOSTR](#): Controls the boost analog gain of the audio-right channel.
- [ACODEC_GET_GAIN_MICL](#): Obtains the analog gain of the audio-left input channel.
- [ACODEC_GET_GAIN_MICR](#): Obtains the analog gain of the audio-right input channel.
- [ACODEC_GET_DACL_VOL](#): Obtains the volume of the audio-left output channel.
- [ACODEC_GET_DACR_VOL](#): Obtains the volume of the audio-right output channel.
- [ACODEC_GET_ADCL_VOL](#): Obtains the volume of the audio-left input channel.
- [ACODEC_GET_ADCR_VOL](#): Obtains the volume of the audio-right input channel.
- [ACODEC_SET_PD_DACL](#): Powers off the audio-left output channel.
- [ACODEC_SET_PD_DACR](#): Powers off the audio-right output channel.



- [ACODEC_SET_PD_ADCL](#): Powers off the audio-left input channel.
- [ACODEC_SET_PD_ADCR](#): Powers off the audio-right input channel.
- [ACODEC_SET_PD_LINEINL](#): Controls the power-off of the line-in input of the audio-left channel.
- [ACODEC_SET_PD_LINEINR](#): Controls the power-off of the line-in input of the audio-right channel.

ACODEC_SOFT_RESET_CTRL

[Description]

Restores the audio codec to default settings.

[Syntax]

```
int ioctl (int fd,  
          ACODEC_SOFT_RESET_CTRL);
```

[Parameter]

Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_SOFT_RESET_CTRL	ioctl number	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: acodec.h

[Note]

Call this MPI only after AI and AO devices are disabled.

[Example]

```
if (ioctl(s32Acodec_Fd, ACODEC_SOFT_RESET_CTRL))  
{  
    printf("ioctl err!\n");  
}
```

[See Also]

None



ACODEC_SET_I2S1_FS

[Description]

Sets the sampling rate of the I²S1 interface.

[Syntax]

```
int ioctl (int fd,  
          ACODEC_SET_I2S1_FS,  
          unsigned int *arg);
```

[Parameter]

Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_SET_I2S1_FS	ioctl number	Input
arg	Unsigned integer pointer	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: acodec.h

[Note]

None

[Example]

```
ACODEC_FS_E i2s_fs_sel;  
i2s_fs_sel = ACODEC_FS_48000;  
if (ioctl(s32Acodec_Fd, ACODEC_SET_I2S1_FS, &i2s_fs_sel))  
{  
    printf("ioctl err!\n");  
}
```

[See Also]

None

ACODEC_SET_INPUT_VOL

[Description]



Sets the total input volume.

[Syntax]

```
int ioctl (int fd, ACODEC_SET_INPUT_VOL, unsigned int *arg);
```

[Parameter]

Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_SET_INPUT_VOL	ioctl number	Input
arg	Signed integer pointer	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

acodec.h

[Note]

- For the Hi3516A, the input volume range is [-87, +86]. Both the analog gain and digital gain are included. A larger value indicates higher volume. For example, the value 86 indicates the maximum volume of 86 dB, and the value -87 indicates the minimum volume (muted status). The volume adjustment takes effect simultaneously in the audio-left and audio-right channels. The recommended volume range is [-10, +56]. Within this range, the noises are lowest because only the analog gain is adjusted, and the voice quality can be ensured.
- For Hi3518E V200, the input volume range is [-72, +86]. The recommended volume range is [26, 56]
- For Hi3519 V100, the input volume range is [-78, +80]. The recommended volume range is [19, 50].

[Example]

```
int iVol;  
iVol = 20;  
if (ioctl(s32Acodec_Fd, ACODEC_SET_INPUT_VOL, &iVol))  
{  
    printf("ioctl err!\n");  
}
```



ACODEC_GET_INPUT_VOL

[Description]

Obtains the total input volume.

[Syntax]

```
int ioctl (int fd, ACODEC_GET_INPUT_VOL, unsigned int *arg);
```

[Parameter]

Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_GET_INPUT_VOL	ioctl number	Input
arg	Signed integer pointer	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

acodec.h

[Note]

None

[Example]

```
int iVol;  
if (ioctl(s32Acodec_Fd, ACODEC_GET_INPUT_VOL, &iVol))  
{  
    printf("ioctl err!\n");  
}
```

ACODEC_SET_OUTPUT_VOL

[Description]

Sets the total output volume.

[Syntax]

```
int ioctl (int fd, ACODEC_SET_OUTPUT_VOL, unsigned int *arg);
```

[Parameter]



Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_SET_OUTPUT_VOL	ioctl number	Input
arg	Signed integer pointer	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

acodec.h

[Note]

The output volume range is $[-121, +6]$. A larger value indicates higher volume. For example, the value 6 indicates the maximum volume of 6 dB, and the value -121 indicates the minimum volume (muted status). The volume adjustment takes effect simultaneously in the audio-left and audio-right channels. The digital gain is adjusted by calling this MPI. It is recommended that a small value be assigned to avoid noises.

[Example]

```
int iVol;  
iVol = 0;  
if (ioctl(s32Acodec_Fd, ACODEC_SET_OUTPUT_VOL, &iVol))  
{  
    printf("ioctl err!\n");  
}
```

ACODEC_GET_OUTPUT_VOL

[Description]

Obtains the total output volume.

[Syntax]

```
int ioctl (int fd, ACODEC_GET_OUTPUT_VOL, unsigned int *arg);
```

[Parameter]

Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_GET_OUTPUT_VOL	ioctl number	Input



Parameter	Description	Input/Output
arg	Signed integer pointer	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

acodec.h

[Note]

None

[Example]

```
int iVol;  
if (ioctl(s32Acodec_Fd, ACODEC_GET_OUTPUT_VOL, &iVol))  
{  
    printf("ioctl err!\n");  
}
```

ACODEC_SET_MIXER_MIC

[Description]

Selects the input mode.

[Syntax]

```
int ioctl (int fd,  
          ACODEC_SET_MIXER_MIC,  
          unsigned int *arg);
```

[Parameter]

Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_SET_MIXER_MIC	ioctl number	Input
arg	Unsigned integer pointer	Input

[Return Value]



Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: acodec.h

[Note]

- For the Hi3516A, the value range of **arg** is [0, 1], and **ACODEC_MIXER_LINEIN** is selected as the input by default.
- For Hi3518E V200, the value range of **arg** is [3, 4], and **ACODEC_MIXER_IN** is selected as the input by default.
- For Hi3519 V100, the value range of **arg** is [0, 2], and the IN0 single-ended input is selected as the input by default. If the inputs are passive audio signals, the analog gain needs to be increased.

[Difference]

Chip	Description
Hi3516A	The MIC input and line-in input are supported. ACODEC_MIXER_MICIN indicates the MIC input, and ACODEC_MIXER_LINEIN indicates the line-in input.
Hi3518E V200	The single-ended input and differential input are supported. ACODEC_MIXER_IN indicates the single-ended input, and ACODEC_MIXER_IN_D indicates the differential input.
Hi3519 V100	The IN0 single-ended input, IN1 single-ended input, and IN_D differential input are supported.

[Example]

```
ACODEC_MIXER_E mixer_mic_ctrl;  
mixer_mic_ctrl = ACODEC_MIXER_IN0;  
if (ioctl(s32Acodec_Fd, ACODEC_SET_MIXER_MIC, &mixer_mic_ctrl))  
{  
    printf("ioctl err!\n");  
}
```

[See Also]

None

ACODEC_SET_GAIN_MICL

[Description]

Controls the analog gain of the audio-left input channel.



[Syntax]

```
int ioctl (int fd,  
          ACODEC_SET_GAIN_MICL,  
          unsigned int *arg);
```

[Parameter]

Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_SET_GAIN_MICL	ioctl number	Input
arg	Unsigned integer pointer	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: acodec.h

[Note]

- If the inputs are passive audio signals, the analog gain needs to be increased.
- For Hi3516A, the value range of **arg** is [0, 31]. When **arg** increases, the gain is incremented by 1.5 dB. The value 0 corresponds to the minimum gain of -16.5 dB, and the value 31 corresponds to the maximum gain of 30 dB.
- For the Hi3518E V200/Hi3519 V100, the value range of the analog gain parameter is [0, 16]. When the parameter value is incremented by 1 from 0 to 15, the analog gain is incremented by 2 dB. The value 0 corresponds to the analog gain of 0 dB, the value 15 corresponds to the maximum analog gain of 30 dB, and the value 16 corresponds to the analog gain of -1.5 dB.

[Example]

```
unsigned int gain_mac;  
gain_mic = 0x18;  
if (ioctl(s32Acodec_Fd, ACODEC_SET_GAIN_MICL, &gain_mic))  
{  
    printf("ioctl err!\n");  
}
```

[See Also]

None



ACODEC_SET_GAIN_MICR

[Description]

Sets the analog gain of the audio-right input channel.

[Syntax]

```
int ioctl (int fd,  
          ACODEC_SET_GAIN_MICR,  
          unsigned int *arg);
```

[Parameter]

Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_SET_GAIN_MICR	ioctl number	Input
arg	Unsigned integer pointer	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: acodec.h

[Note]

- If the inputs are passive audio signals, the analog gain needs to be increased.
- For Hi3516A, the value range of **arg** is [0, 31]. When **arg** increases, the gain is increment by 1.5 dB. The value 0 corresponds to the minimum gain of -16.5 dB, and the value 31 corresponds to the maximum gain of 30 dB.
- For the Hi3518E V200/Hi3519 V100, the value range of the analog gain parameter is [0, 16]. When the parameter value is incremented by 1 from 0 to 15, the analog gain is incremented by 2 dB. The value 0 corresponds to the analog gain of 0 dB, the value 15 corresponds to the maximum analog gain of 30 dB, and the value 16 corresponds to the analog gain of -1.5 dB.

[Example]

```
unsigned int gain_mic;  
gain_mic = 0x18;  
if (ioctl(s32Acodec_Fd, ACODEC_SET_GAIN_MICR, &gain_mic))  
{  
    printf("ioctl err!\n");
```



}

[See Also]

None

ACODEC_SET_DACL_VOL

[Description]

Set the volume of the audio-left output channel.

[Syntax]

```
int ioctl (int fd,  
          ACODEC_SET_DACL_VOL,  
          ACODEC_VOL_CTRL *arg);
```

[Parameter]

Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_SET_DACL_VOL	ioctl number	Input
arg	Pointer to ACODEC_VOL_CTRL	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: acodec.h

[Note]

The output volume range of the audio-left channel is [0, 127]. A larger value indicates low volume. For example, the value **0** indicates the maximum volume of 6 dB, and the value **127** indicates the minimum volume (muted status).

[Example]

```
ACODEC_VOL_CTRL vol_ctrl;  
vol_ctrl.vol_ctrl_mute = 0x0;  
vol_ctrl.vol_ctrl = 0x06;  
if (ioctl(s32Acodec_Fd, ACODEC_SET_DACL_VOL, &vol_ctrl))  
{  
    printf("ioctl err!\n");
```



}

[See Also]

None

ACODEC_SET_DACR_VOL

[Description]

Sets the output volume of the audio-right channel.

[Syntax]

```
int ioctl (int fd,  
          ACODEC_SET_DACR_VOL,  
          ACODEC_VOL_CTRL *arg);
```

[Parameter]

Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_SET_DACL_VOL	ioctl number	Input
arg	Pointer to ACODEC_VOL_CTRL	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: acodec.h

[Note]

The output volume range of the audio-right channel is [0, 127]. A larger value indicates low volume. For example, the value **0** indicates the maximum volume of 6 dB, and the value **127** indicates the minimum volume (muted status).

[Example]

```
ACODEC_VOL_CTRL vol_ctrl;  
vol_ctrl.vol_ctrl_mute = 0x0;  
vol_ctrl.vol_ctrl = 0x06;  
if (ioctl(s32Acodec_Fd, ACODEC_SET_DACR_VOL, &vol_ctrl))  
{  
    printf("ioctl err!\n");
```



}

[See Also]

None

ACODEC_SET_ADCL_VOL

[Description]

Sets the volume of the audio-left input channel.

[Syntax]

```
int ioctl (int fd,  
          ACODEC_SET_ADCL_VOL,  
          ACODEC_VOL_CTRL *arg);
```

[Parameter]

Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_SET_ADCL_VOL	ioctl number	Input
arg	Pointer to ACODEC_VOL_CTRL	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: acodec.h

[Note]

The input volume range of the audio-left channel is [0, 127]. A larger value indicates low volume. For example, the value **0** indicates the maximum volume of 30 dB, and the value **127** indicates the minimum volume of -97 dB.

[Example]

```
ACODEC_VOL_CTRL vol_ctrl;  
vol_ctrl.vol_ctrl_mute = 0x0;  
vol_ctrl.vol_ctrl = 0x06;  
if (ioctl(s32Acodec_Fd, ACODEC_SET_ADCL_VOL, &vol_ctrl))  
{  
    printf("ioctl err!\n");
```



}

[See Also]

None

ACODEC_SET_ADCR_VOL

[Description]

Sets the volume of the audio-right input channel.

[Syntax]

```
int ioctl (int fd,  
          ACODEC_SET_ADCR_VOL,  
          ACODEC_VOL_CTRL *arg);
```

[Parameter]

Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_SET_ADCR_VOL	ioctl number	Input
arg	Pointer to ACODEC_VOL_CTRL	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: acodec.h

[Note]

The input volume range of the audio-right channel is [0, 127]. A larger value indicates low volume. For example, the value **0** indicates the maximum volume of 30 dB, and the value **127** indicates the minimum volume of -97 dB.

[Example]

```
ACODEC_VOL_CTRL vol_ctrl;  
vol_ctrl.vol_ctrl_mute = 0x0;  
vol_ctrl.vol_ctrl = 0x06;  
if (ioctl(s32Acodec_Fd, ACODEC_SET_ADCR_VOL, &vol_ctrl))  
{  
    printf("ioctl err!\n");
```



}

[See Also]

None

ACODEC_SET_MICL_MUTE

[Description]

Sets the mute of the audio-left input channel.

[Syntax]

```
int ioctl (int fd,  
          ACODEC_SET_MICL_MUTE,  
          unsigned int *arg);
```

[Parameter]

Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_SET_MICL_MUTE	ioctl number	Input
arg	Unsigned integer pointer	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: acodec.h

[Note]

The value range of **arg** is [0, 1].

[Example]

```
unsigned int mute_ctrl;  
mute_ctrl = 0x1;  
if (ioctl(s32Acodec_Fd, ACODEC_SET_MICL_MUTE, &mute_ctrl))  
{  
    printf("ioctl err!\n");  
}
```

[See Also]



None

ACODEC_SET_MICR_MUTE

[Description]

Sets the mute of the audio-right input channel.

[Syntax]

```
int ioctl (int fd,  
          ACODEC_SET_MICR_MUTE,  
          unsigned int *arg);
```

[Parameter]

Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_SET_MICR_MUTE	ioctl number	Input
arg	Unsigned integer pointer	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: acodec.h

[Note]

The value range of **arg** is [0, 1].

[Example]

```
unsigned int mute_ctrl;  
mute_ctrl = 0x1;  
if (ioctl(s32Acodec_Fd, ACODEC_SET_MICR_MUTE, &mute_ctrl))  
{  
    printf("ioctl err!\n");  
}
```

[See Also]

None



ACODEC_SET_DACL_MUTE

[Description]

Sets the mute of the audio-left output channel.

[Syntax]

```
int ioctl (int fd,  
          ACODEC_SET_DACL_MUTE,  
          unsigned int *arg);
```

[Parameter]

Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_SET_DACL_MUTE	ioctl number	Input
arg	Unsigned integer pointer	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: acodec.h

[Note]

The value range of **arg** is [0, 1].

[Example]

```
unsigned int mute_ctrl;  
mute_ctrl = 0x1;  
if (ioctl(s32Acodec_Fd, ACODEC_SET_DACL_MUTE, &mute_ctrl))  
{  
    printf("ioctl err!\n");  
}
```

[See Also]

None

ACODEC_SET_DACR_MUTE

[Description]



Sets the mute of the audio-right output channel.

[Syntax]

```
int ioctl (int fd,  
          ACODEC_SET_DACR_MUTE,  
          unsigned int *arg);
```

[Parameter]

Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_SET_DACR_MUTE	ioctl number	Input
arg	Unsigned integer pointer	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: acodec.h

[Note]

The value range of **arg** is [0, 1].

[Example]

```
unsigned int mute_ctrl;  
mute_ctrl = 0x1;  
if (ioctl(s32Acodec_Fd, ACODEC_SET_DACR_MUTE, &mute_ctrl))  
{  
    printf("ioctl err!\n");  
}
```

[See Also]

None

ACODEC_DAC_SOFT_MUTE

[Description]

Controls the soft mute function of the DAC.

[Syntax]



```
int ioctl (int fd,
           ACODEC_DAC_SOFT_MUTE,
           unsigned int *arg);
```

[Parameter]

Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_DAC_SOFT_MUTE	ioctl number	Input
arg	Unsigned integer pointer	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: acodec.h

[Note]

- The value range of **arg** is [0, 1].
- Hi3518E V200 and Hi3519 V100 do not support this MPI.

[Example]

```
unsigned int soft_mute_ctrl;
soft_mute_ctrl = 0x1;
if (ioctl(s32Acodec_Fd, ACODEC_DAC_SOFT_MUTE, &soft_mute_ctrl))
{
    printf("ioctl err!\n");
}
```

[See Also]

None

ACODEC_DAC_SOFT_UNMUTE

[Description]

Controls the soft unmute function of the DAC.

[Syntax]

```
int ioctl (int fd,
           ACODEC_DAC_SOFT_UNMUTE,
```



```
unsigned int *arg);
```

[Parameter]

Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_DAC_SOFT_UNMUTE	ioctl number	Input
arg	Unsigned integer pointer	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: acodec.h

[Note]

- The value range of **arg** is [0, 1].
- Hi3518E V200 and Hi3519 V100 do not support this MPI.

[Example]

```
unsigned int soft_mute_ctrl;
soft_nute_ctrl = 0x1;
if (ioctl(s32Acodec_Fd, ACODEC_DAC_SOFT_UNMUTE, &soft_mute_ctrl))
{
    printf("ioctl err!\n");
}
```

[See Also]

None

ACODEC_ENABLE_BOOSTL

[Description]

Controls the boost analog gain of the audio-left channel.

[Syntax]

```
int ioctl (int fd,
           ACODEC_ENABLE_BOOSTL,
           unsigned int *arg);
```



[Parameter]

Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_ENABLE_BOOSTL	ioctl number	Input
arg	Unsigned integer pointer	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

acodec.h

[Note]

- The value range of **arg** is [0, 1]. The value 0 indicates that the analog gain is incremented by 0 dB, and the value 1 indicates that the analog gain is incremented by 20 dB.
- The Hi3516A and Hi3518E V200 do not support this MPI.

[Example]

```
unsigned int enable_boostl;
enable_boostl = 0x1;
if (ioctl(s32Acodec_Fd, ACODEC_ENABLE_BOOSTL, &enable_boostl))
{
    printf("ioctl err!\n");
}
```

ACODEC_ENABLE_BOOSTR

[Description]

Controls the boost analog gain of the audio-right channel.

[Syntax]

```
int ioctl (int fd,
           ACODEC_ENABLE_BOOSTR,
           unsigned int *arg);
```

[Parameter]



Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_ENABLE_BOOSTR	ioctl number	Input
arg	Unsigned integer pointer	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

acodec.h

[Note]

- The value range of **arg** is [0, 1]. The value 0 indicates that the analog gain is incremented by 0 dB, and the value 1 indicates that the analog gain is incremented by 20 dB.
- The Hi3516A and Hi3518E V200 do not support this MPI.

[Example]

```
unsigned int enable_boosrt;
enable_boosrt = 0x1;
if (ioctl(s32Acodec_Fd, ACODEC_ENABLE_BOOSTR, &enable_boosrt))
{
    printf("ioctl err!\n");
}
```

ACODEC_GET_GAIN_MICL

[Description]

Obtains the analog gain of the audio-left input channel.

[Syntax]

```
int ioctl (int fd,
           ACODEC_GET_GAIN_MICL,
           unsigned int *arg);
```

[Parameter]



Parameter	Description	Input/Output
Fd	File descriptor of the audio codec	Input
ACODEC_GET_GAIN_MICL	ioctl number	Input
arg	Unsigned integer pointer	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: acodec.h

[Note]

None

[Example]

```
unsigned int gain_mic;
if (ioctl(s32Acodec_Fd, ACODEC_GET_GAIN_MICL, &gain_mic))
{
    printf("ioctl err!\n");
}
```

[See Also]

None

ACODEC_GET_GAIN_MICR

[Description]

Obtains the analog gain of the audio-right input channel.

[Syntax]

```
int ioctl (int fd,
           ACODEC_GET_GAIN_MICR,
           unsigned int *arg);
```

[Parameter]

Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input



Parameter	Description	Input/Output
ACODEC_GET_GAIN_MICR	ioctl number	Input
arg	Unsigned integer pointer	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: acodec.h

[Note]

None

[Example]

```
unsigned int gain_mic;
if (ioctl(s32Acodec_Fd, ACODEC_GET_GAIN_MICR, &gain_mic))
{
    printf("ioctl err!\n");
}
```

[See Also]

None

ACODEC_GET_DACL_VOL

[Description]

Obtains the volume of the audio-left output channel.

[Syntax]

```
int ioctl (int fd,
           ACODEC_GET_DACL_VOL,
           ACODEC_VOL_CTRL *arg);
```

[Parameter]

Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_GET_DACL_VOL	ioctl number	Input



Parameter	Description	Input/Output
arg	Pointer to ACODEC_VOL_CTRL	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: acodec.h

[Note]

None

[Example]

```
ACODEC_VOL_CTRL vol_ctrl;
if (ioctl(s32Acodec_Fd, ACODEC_GET_DACL_VOL, &vol_ctrl))
{
    printf("ioctl err!\n");
}
```

[See Also]

None

ACODEC_GET_DACR_VOL

[Description]

Obtains the volume of the audio-right output channel.

[Syntax]

```
int ioctl (int fd,
           ACODEC_GET_DACR_VOL,
           ACODEC_VOL_CTRL *arg);
```

[Parameter]

Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_GET_DACR_VOL	ioctl number	Input
arg	Pointer to ACODEC_VOL_CTRL	Output



[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: acodec.h

[Note]

None

[Example]

```
ACODEC_VOD_CTRL vol_ctrl;
if (ioctl(s32Acodec_Fd, ACODEC_GET_DACR_VOL, &vol_ctrl))
{
    printf("ioctl err!\n");
}
```

[See Also]

None

ACODEC_GET_ADCL_VOL

[Description]

Obtains the volume of the audio-left input channel.

[Syntax]

```
int ioctl (int fd,
           ACODEC_GET_ ADCL_VOL,
           ACODEC_VOL_CTRL *arg);
```

[Parameter]

Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_GET_ADCL_VOL	ioctl number	Input
arg	Pointer to ACODEC_VOL_CTRL	Output

[Return Value]



Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: acodec.h

[Note]

None

[Example]

```
ACODEC_VOL_CTRL vol_ctrl;
if (ioctl(s32Acodec_Fd, ACODEC_GET_ADCL_VOL, &vol_ctrl))
{
    printf("ioctl err!\n");
}
```

[See Also]

None

ACODEC_GET_ADCR_VOL

[Description]

Obtains the volume of the audio-right input channel.

[Syntax]

```
int ioctl (int fd,
           ACODEC_GET_ADCR_VOL,
           ACODEC_VOL_CTRL *arg);
```

[Parameter]

Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_GET_ADCR_VOL	ioctl number	Input
arg	Pointer to ACODEC_CTRL	Output

[Return Value]

Return Value	Description
0	Success



Return Value	Description
Other values	Failure

[Requirement]

Header file: acodec.h

[Note]

None

[Example]

```
ACODEC_VOL_CTRL vol_ctrl;
if (ioctl(s32Acodec_Fd, ACODEC_GET_ADCR_VOL, &vol_ctrl))
{
    printf("ioctl err!\n");
}
```

[See Also]

None

ACODEC_SET_PD_DACL

[Description]

Powers off the audio-left output channel.

[Syntax]

```
int ioctl (int fd,
           ACODEC_SET_PD_DACL,
           unsigned int *arg);
```

[Parameter]

Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_SET_PD_DACL	ioctl number	Input
arg	Unsigned integer pointer	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure



[Requirement]

Header file: acodec.h

[Note]

The DACL can be powered off by calling this MPI when the DAC is not used. The value range of **arg** is [0, 1].

[Example]

```
unsigned int pd_ctrl;  
pd_ctrl = 0x01;  
if (ioctl(s32Accodec_Fd, ACODEC_SET_PD_DACL, &pd_ctrl))  
{  
    printf("ioctl err!\n");  
}
```

[See Also]

None

ACODEC_SET_PD_DACR

[Description]

Powers off the audio-right output channel.

[Syntax]

```
int ioctl (int fd,  
          ACODEC_SET_PD_DACR,  
          unsigned int *arg);
```

[Parameter]

Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_SET_PD_DACR	ioctl number	Input
arg	Unsigned integer pointer	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure



[Requirement]

Header file: acodec.h

[Note]

The DACR can be powered off by calling this MPI when the DAC is not used. The value range of **arg** is [0, 1].

[Example]

```
unsigned int pd_ctrl;  
pd_ctrl = 0x01  
if (ioctl(s32Acodec_Fd, ACODEC_SET_PD_DACR, &pd_ctrl))  
{  
    printf("ioctl err!\n");  
}
```

[See Also]

None

ACODEC_SET_PD_ADCL

[Description]

Powers off the audio-left input channel.

[Syntax]

```
int ioctl (int fd,  
          ACODEC_SET_PD_ADCL,  
          unsigned int *arg);
```

[Parameter]

Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_SET_PD_ADCL	ioctl number	Input
arg	Unsigned integer pointer	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]



Header file: acodec.h

[Note]

The ADCL can be powered off by calling this MPI when the ADC is not used. The value range of **arg** is [0, 1].

[Example]

```
unsigned int pd_ctrl;  
pd_ctrl = 0x01;  
if (ioctl(s32Acodec_Fd, ACODEC_SET_PD_ADCL, &pd_ctrl))  
{  
    printf("ioctl err!\n");  
}
```

[See Also]

None

ACODEC_SET_PD_ADCR

[Description]

Powers off the audio-right input channel.

[Syntax]

```
int ioctl (int fd,  
          ACODEC_SET_PD_ADCR,  
          unsigned int *arg);
```

[Parameter]

Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_SET_PD_ADCR	ioctl number	Input
arg	Unsigned integer pointer	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: acodec.h



[Note]

The ADCR can be powered off by calling this MPI when the ADC is not used. The value range of **arg** is [0, 1].

[Example]

```
unsigned int pd_ctrl;  
pd_ctrl = 0x01;  
if (ioctl(s32Acodec_Fd, ACODEC_SET_PD_ADCR, &pd_ctrl))  
{  
    printf("ioctl err!\n");  
}
```

[See Also]

None

ACODEC_SET_PD_LINEINL

[Description]

Controls the power-off of the line-in input of the audio-left channel.

[Syntax]

```
int ioctl (int fd,  
          ACODEC_SET_PD_LINEINL,  
          unsigned int *arg);
```

[Parameter]

Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_SET_PD_LINEINL	ioctl number	Input
arg	Unsigned integer pointer	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

acodec.h

[Note]



- The LINEINL can be powered off by calling this MPI when the line-in input is not used.
The value range of **arg** is [0, 1].
- The Hi3516A and Hi3518E V200 do not support this MPI.

[Example]

```
unsigned int lineinl_pd_ctrl;  
lineinl_pd_ctrl = 0x01;  
if (ioctl(s32Acodec_Fd, ACODEC_SET_PD_LINEINL, &lineinl_pd_ctrl))  
{  
    printf("ioctl err!\n");  
}
```

ACODEC_SET_PD_LINEINR

[Description]

Controls the power-off of the line-in input of the audio-right channel.

[Syntax]

```
int ioctl (int fd,  
          ACODEC_SET_PD_LINEINR,  
          unsigned int *arg);
```

[Parameter]

Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_SET_PD_LINEINR	ioctl number	Input
arg	Unsigned integer pointer	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

acodec.h

[Note]

- The LINEINR can be powered off by calling this MPI when the line-in input is not used.
The value range of **arg** is [0, 1].
- The Hi3516A and Hi3518E V200 do not support this MPI.



[Example]

```
unsigned int lineinr_pd_ctrl;  
lineinr_pd_ctrl = 0x01;  
if (ioctl(s32Acodec_Fd, ACODEC_SET_PD_LINEINR, &lineinr_pd_ctrl))  
{  
    printf("ioctl err!\n");  
}
```

9.3.5.2 Extended Commands

The following are extended commands:

- [ACODEC_SEL_DAC_CLK](#): Sets the clock edges of the DAC to same or opposite.
- [ACODEC_SEL_ADC_CLK](#): Sets the clock edges of the ADC to the same edges or reversed edges.
- [ACODEC_SEL_ANA_MCLK](#): Sets the clock edges of the analog part and digital part to same or opposite.
- [ACODEC_DACL_SEL_TRACK](#): Sets the DACL for selecting the audio channel.
- [ACODEC_DACR_SEL_TRACK](#): Sets the DACR for selecting the audio channel.
- [ACODEC_ADCL_SEL_TRACK](#): Sets the ADCL for selecting the audio channel.
- [ACODEC_ADCR_SEL_TRACK](#): Sets the ADCR for selecting the audio channel.
- [ACODEC_SET_DAC_DE_EMPHASIS](#): Controls the de-emphasis filtering function of the DAC.
- [ACODEC_SET_ADC_HP_FILTER](#): Controls the high-pass filtering function of the ADC.
- [ACODEC_DAC_POP_FREE](#): Deletes the POP noise of the DAC.
- [ACODEC_DAC_SOFT_MUTE_RATE](#): Controls the rate for soft mute of the DAC.
- [ACODEC_DAC_SEL_I2S](#): Selects the I²S interface for the DAC.
- [ACODEC_ADC_SEL_I2S](#): Selects the I²S interface for the ADC.
- [ACODEC_SET_I2S1_DATAWIDTH](#): Sets the data width of the I²S1 interface
- [ACODEC_SET_I2S2_DATAWIDTH](#): Sets the data width of the I²S2 interface
- [ACODEC_SET_I2S2_FS](#): Sets the sampling rate of the I²S2 interface.
- [ACODEC_SET_DACR2DACL_VOL](#): Mixes the volume of the DACR with that of the DACL.
- [ACODEC_SET_DACL2DACR_VOL](#): Mixes the volume of the DACL with that of the DACR.
- [ACODEC_SET_ADCL2DACL_VOL](#): Mixes the volume of the ADCL with that of the DACL.
- [ACODEC_SET_ADCR2DACL_VOL](#): Mixes the volume of the ADCR with that of the DACL.
- [ACODEC_SET_ADCL2DACR_VOL](#): Mixes the volume of the ADCL with that of the DACR.
- [ACODEC_SET_ADCR2DACL_VOL](#): Mixes the volume of the ADCR with that of the DACL.



ACODEC_SEL_DAC_CLK

[Description]

Sets the clock edges of the DAC to same or opposite.

[Syntax]

```
int ioctl (int fd,  
          ACODEC_SEL_DAC_CLK,  
          unsigned int *arg);
```

[Parameter]

Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_SEL_DAC_CLK	ioctl number	Input
arg	Unsigned integer pointer	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: acodec.h

[Note]

The value range of **arg** is [0, 1].

[Example]

```
unsigned int clk_sel;  
clk_sel = 0x1;  
if (ioctl(s32Acodec_Fd, ACODEC_SEL_DAC_CLK, &clk_sel))  
{  
    printf("ioctl err!\n");  
}
```

[See Also]

None

ACODEC_SEL_ADC_CLK

[Description]



Sets the clock edges of the ADC to the same edges or reversed edges.

[Syntax]

```
int ioctl (int fd,  
          ACODEC_SEL_ADC_CLK,  
          unsigned int *arg);
```

[Parameter]

Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_SEL_ADC_CLK	ioctl number	Input
arg	Unsigned integer pointer	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

acodec.h

[Note]

- The value range of **arg** is [0, 1].
- The Hi3516A and Hi3518E V200 do not support this MPI.

[Example]

```
unsigned int clk_sel;  
clk_sel = 0x1;  
if (ioctl(s32Acodec_Fd, ACODEC_SEL_ADC_CLK, &clk_sel))  
{  
    printf("ioctl err!\n");  
}
```

ACODEC_SEL_ANA_MCLK

[Description]

Sets the clock edges of the analog part and digital part to same or opposite.

[Syntax]

```
int ioctl (int fd,  
          ACODEC_SEL_ANA_MCLK,
```



```
unsigned int *arg);
```

[Parameter]

Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_SEL_ANA_MCLK	ioctl number	Input
arg	Unsigned integer pointer	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: acodec.h

[Note]

The value range of **arg** is [0, 1].

[Example]

```
unsigned int clk_sel;
clk_sel = 0x1;
if (ioctl(s32Acodec_Fd, ACODEC_SEL_ANA_MCLK, &clk_sel))
{
    printf("ioctl err!\n");
}
```

[See Also]

None

ACODEC_DACL_SEL_TRACK

[Description]

Sets the DACL for selecting the audio channel.

[Syntax]

```
int ioctl (int fd,
           ACODEC_DACL_SEL_TRACK,
           unsigned int *arg);
```

[Parameter]



Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_DACL_SEL_TRACK	ioctl number	Input
arg	Unsigned integer pointer	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: acodec.h

[Note]

Hi3518E V200 and Hi3519 V100 do not support this MPI.

[Example]

```
unsigned int track_select;
track_select = 0x1;
if (ioctl(s32Acodec_Fd, ACODEC_DACL_SEL_TRACK, &track_select))
{
    printf("ioctl err!\n");
}
```

[See Also]

None

ACODEC_DACR_SEL_TRACK

[Description]

Sets the DACR for selecting the audio channel.

[Syntax]

```
int ioctl (int fd,
           ACODEC_DACR_SEL_TRACK,
           unsigned int *arg);
```

[Parameter]



Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_DACR_SEL_TRACK	ioctl number	Input
arg	Unsigned integer pointer	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: acodec.h

[Note]

Hi3518E V200 and Hi3519 V100 do not support this MPI.

[Example]

```
unsigned int track_select;
track_select = 0x1;
if (ioctl(s32Acodec_Fd, ACODEC_DACR_SEL_TRACK, &track_select)
{
    printf("ioctl err!\n");
}
```

[See Also]

None

ACODEC_ADCL_SEL_TRACK

[Description]

Sets the ADCL for selecting the audio channel.

[Syntax]

```
int ioctl (int fd,
           ACODEC_ADCL_SEL_TRACK,
           unsigned int *arg);
```

[Parameter]



Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_ADCL_SEL_TRACK	ioctl number	Input
arg	Unsigned integer pointer	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: acodec.h

[Note]

Hi3518E V200 and Hi3519 V100 do not support this MPI.

[Example]

```
unsigned int track_select;
track_select = 0x1;
if (ioctl(s32Acodec_Fd, ACODEC_ADCL_SEL_TRACK, &track_select))
{
    printf("ioctl err!\n");
}
```

[See Also]

None

ACODEC_ADCR_SEL_TRACK

[Description]

Sets the ADCR for selecting the audio channel.

[Syntax]

```
int ioctl (int fd,
           ACODEC_ADCR_SEL_TRACK,
           unsigned int *arg);
```

[Parameter]



Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_ADCR_SEL_TRACK	ioctl number	Input
arg	Unsigned integer pointer	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: acodec.h

[Note]

Hi3518E V200 and Hi3519 V100 do not support this MPI.

[Example]

```
unsigned int track_select;
track_select = 0x1;
if (ioctl(s32Acodec_Fd, ACODEC_ADCR_SEL_TRACK, &track_select))
{
    printf("ioctl err!\n");
}
```

[See Also]

None

ACODEC_SET_DAC_DE_EMPHASIS

[Description]

Control the de-emphasis filtering function of the DAC.

[Syntax]

```
int ioctl (int fd,
           ACODEC_SET_DAC_DE_EMPHASIS,
           unsigned int *arg);
```

[Parameter]



Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_SET_DAC_DE_EMPHASIS	ioctl number	Input
arg	Unsigned integer pointer	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: acodec.h

[Note]

The value range of **arg** is [0, 3].

[Example]

```
unsigned int dac_deemphasis;
dac_deemphasis = 0x1;
if (ioctl(s32Acodec_Fd, ACODEC_SET_DAC_DE_EMPHASIS, &dac_deemphasis) )
{
    printf("ioctl err!\n");
}
```

[See Also]

None

ACODEC_SET_ADC_HP_FILTER

[Description]

Controls the high-pass filtering function of the ADC.

[Syntax]

```
int ioctl (int fd,
           ACODEC_SET_ADC_HP_FILTER,
           unsigned int *arg);
```

[Parameter]



Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_SET_ADC_HP_FILTER	ioctl number	Input
arg	Unsigned integer pointer	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: acodec.h

[Note]

The value range of **arg** is [0, 1].

[Example]

```
unsigned int adc_hpf;
adc_hpf = 0x1;
if (ioctl(s32Acodec_Fd, ACODEC_SET_ADC_HP_FILTER, &adc_hpf))
{
    printf("ioctl err!\n");
}
```

[See Also]

None

ACODEC_DAC_POP_FREE

[Description]

Deletes the POP noise of the DAC.

[Syntax]

```
int ioctl (int fd,
           ACODEC_DAC_POP_FREE,
           unsigned int *arg);
```

[Parameter]



Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_DAC_POP_FREE	ioctl number	Input
arg	Unsigned integer pointer	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: acodec.h

[Note]

- The value range of **arg** is [0, 1].
- Hi3518E V200 and Hi3519 V100 do not support this MPI.

[Example]

```
unsigned int dac_pop_free;
dac_pop_free = 0x1;
if (ioctl(s32Acodec_Fd, ACODEC_DAC_POP_FREE, &dac_pop_free))
{
    printf("ioctl err!\n");
}
```

[See Also]

None

ACODEC_DAC_SOFT_MUTE_RATE

[Description]

Controls the rate for soft mute of the DAC.

[Syntax]

```
int ioctl (int fd,
           ACODEC_DAC_POP_FREE,
           unsigned int *arg);
```

[Parameter]



Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_DAC_SOFT_MUTE_RATE	ioctl number	Input
arg	Unsigned integer pointer	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: acodec.h

[Note]

- The value range of **arg** is [0, 3].
- Hi3518E V200 and Hi3519 V100 do not support this MPI.

[Example]

```
unsigned int soft_mute_rate;
soft_mute_rate = 0x1;
if (ioctl(s32Acodec_Fd, ACODEC_DAC_SOFT_MUTE_RATE, &soft_mute_rate) )
{
    printf("ioctl err!\n");
}
```

[See Also]

None

ACODEC_DAC_SEL_I2S

[Description]

Selects the I²S interface for the DAC.

[Syntax]

```
int ioctl (int fd,
           ACODEC_DAC_SEL_I2S,
           unsigned int *arg);
```

[Parameter]



Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_DAC_SEL_I2S	ioctl number	Input
arg	Unsigned integer pointer	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: acodec.h

[Note]

- The value range of **arg** is [0, 1].
- Hi3518E V200 and Hi3519 V100 do not support this MPI.

[Example]

```
unsigned int i2s_sel;  
i2s_sel = 0x0;  
if (ioctl(s32Acodec_Fd, ACODEC_DAC_SOFT_MUTE_RATE, &i2s_sel))  
{  
    printf("ioctl err!\n");  
}
```

[See Also]

None

ACODEC_ADC_SEL_I2S

[Description]

Selects the I²S interface for the ADC.

[Syntax]

```
int ioctl (int fd,  
          ACODEC_ADC_SEL_I2S,  
          unsigned int *arg);
```

[Parameter]



Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_ADC_SEL_I2S	ioctl number	Input
arg	Unsigned integer pointer	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: acodec.h

[Note]

- The value range of **arg** is [0, 1].
- Hi3518E V200 and Hi3519 V100 do not support this MPI.

[Example]

```
unsigned int i2s_sel;  
i2s_sel = 0x0;  
if (ioctl(s32Acodec_Fd, ACODEC_ADC_SEL_I2S, &i2s_sel))  
{  
    printf("ioctl err!\n");  
}
```

[See Also]

None

ACODEC_SET_I2S1_DATAWIDTH

[Description]

Sets the data width of the I²S1 interface.

[Syntax]

```
int ioctl (int fd,  
          ACODEC_SET_I2S1_DATAWIDTH,  
          unsigned int *arg);
```

[Parameter]



Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_SET_I2S1_DATAWIDTH	ioctl number	Input
arg	Unsigned integer pointer	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: acodec.h

[Note]

The value range of **arg** is [0, 3].

[Example]

```
unsigned int i2s_datawidth;
i2s_datawidth = 0x0;
if (ioctl(s32Acodec_Fd, ACODEC_SET_I2S1_DATAWIDTH, &i2s_datawidth))
{
    printf("ioctl err!\n");
}
```

[See Also]

None

ACODEC_SET_I2S2_DATAWIDTH

[Description]

Sets the data width of the I²S2 interface.

[Syntax]

```
int ioctl (int fd,
           ACODEC_SET_I2S2_DATAWIDTH,
           unsigned int *arg);
```

[Parameter]



Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_SET_I2S2_DATAWIDTH	ioctl number	Input
arg	Unsigned integer pointer	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: acodec.h

[Note]

- The value range of **arg** is [0, 3].
- Hi3518E V200 and Hi3519 V100 do not support this MPI.

[Example]

```
unsigned int i2s_datawidth;
i2s_datawidth = 0x0;
if (ioctl(s32Acodec_Fd, ACODEC_SET_I2S2_DATAWIDTH, &i2s_datawidth))
{
    printf("ioctl err!\n");
}
```

[See Also]

None

ACODEC_SET_I2S2_FS

[Description]

Sets the sampling rate of the I²S2 interface.

[Syntax]

```
int ioctl (int fd,
           ACODEC_SET_I2S2_FS,
           unsigned int *arg);
```

[Parameter]



Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_SET_I2S2_FS	ioctl number	Input
arg	Unsigned integer pointer	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: acodec.h

[Note]

- The value range of **arg** is [0, 31].
- Hi3518E V200 and Hi3519 V100 do not support this MPI.

[Example]

```
ACODEC_FS_E i2s_fs_sel;
i2s_fs_sel = ACODEC_FS_48000;
if (ioctl(s32Acodec_Fd, ACODEC_SET_I2S2_FS, &i2s_fs_sel))
{
    printf("ioctl err!\n");
}
```

[See Also]

None

ACODEC_SET_DACR2DACL_VOL

[Description]

Mixes the volume of the DACR with that of the DACL.

[Syntax]

```
int ioctl (int fd,
           ACODEC_SET_DACR2DACL_VOL,
           ACODEC_VOL_CTRL *arg);
```

[Parameter]



Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_SET_DACR2DACL_VOL	ioctl number	Input
arg	Pointer to ACODEC_VOL_CTRL	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: acodec.h

[Note]

Hi3518E V200 and Hi3519 V100 do not support this MPI.

[Example]

```
ACODEC_VOL_CTRL vol_ctrl;
vol_ctrl.vol_ctrl_mute = 0x0;
vol_ctrl.vol_ctrl = 0x06;
if (ioctl(s32Acodec_Fd, ACODEC_SET_DACR2DACL, &vol_ctrl))
{
    printf("ioctl err!\n");
}
```

[See Also]

None

ACODEC_SET_DACL2DACL_VOL

[Description]

Mixes the volume of the DACL with that of the DACR.

[Syntax]

```
int ioctl (int fd,
           ACODEC_SET_DACL2DACL_VOL,
           ACODEC_VOL_CTRL *arg);
```

[Parameter]



Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_SET_DACL2DACL_VOL	ioctl number	Input
arg	Pointer to ACODEC_VOL_CTRL	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: acodec.h

[Note]

Hi3518E V200 and Hi3519 V100 do not support this MPI.

[Example]

```
ACODEC_VOL_CTRL vol_ctrl;
vol_ctrl.vol_ctrl_mute = 0x0;
vol_ctrl.vol_ctrl = 0x06;
if (ioctl(s32Acodec_Fd, ACODEC_SET_DACL2DACL, &vol_ctrl))
{
    printf("ioctl err!\n");
}
```

[See Also]

None

ACODEC_SET_ADCL2DACL_VOL

[Description]

Mixes the volume of the ADCL with that of the DACL.

[Syntax]

```
int ioctl (int fd,
           ACODEC_SET_ADCL2DACL_VOL,
           ACODEC_VOL_CTRL *arg);
```

[Parameter]



Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_SET_ADCL2DACL_VOL	ioctl number	Input
arg	Pointer to ACODEC_VOL_CTRL	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: acodec.h

[Note]

Hi3518E V200 and Hi3519 V100 do not support this MPI.

[Example]

```
ACODEC_VOL_CTRL vol_ctrl;
vol_ctrl.vol_ctrl_mute = 0x0;
vol_ctrl.vol_ctrl = 0x06;
if (ioctl(s32Acodec_Fd, ACODEC_SET_ADCL2DACL, &vol_ctrl))
{
    printf("ioctl err!\n");
}
```

[See Also]

None

ACODEC_SET_ADCR2DACL_VOL

[Description]

Mixes the volume of the ADCR with that of the DACL.

[Syntax]

```
int ioctl (int fd,
           ACODEC_SET_ADCR2DACL_VOL,
           ACODEC_VOL_CTRL *arg);
```

[Parameter]



Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_SET_ADCR2DACL_VOL	ioctl number	Input
arg	Pointer to ACODEC_VOL_CTRL	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: acodec.h

[Note]

Hi3518E V200 and Hi3519 V100 do not support this MPI.

[Example]

```
ACODEC_VOL_CTRL vol_ctrl;
vol_ctrl.vol_ctrl_mute = 0x0;
vol_ctrl.vol_ctrl = 0x06;
if (ioctl(s32Acodec_Fd, ACODEC_SET_ADCR2DACL, &vol_ctrl))
{
    printf("ioctl err!\n");
}
```

[See Also]

None

ACODEC_SET_ADCL2DACR_VOL

[Description]

Mixes the volume of the ADCL with that of the DACR.

[Syntax]

```
int ioctl (int fd,
           ACODEC_SET_ADCL2DACR_VOL,
           ACODEC_VOL_CTRL *arg);
```

[Parameter]



Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_SET_ADCL2DACR_VOL	ioctl number	Input
arg	Pointer to ACODEC_VOL_CTRL	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: acodec.h

[Note]

Hi3518E V200 and Hi3519 V100 do not support this MPI.

[Example]

```
ACODEC_VOL_CTRL vol_ctrl;
vol_ctrl.vol_ctrl_mute = 0x0;
vol_ctrl.vol_ctrl = 0x06;
if (ioctl(s32Acodec_Fd, ACODEC_SET_ADCL2DACR, &vol_ctrl))
{
    printf("ioctl err!\n");
}
```

[See Also]

None

ACODEC_SET_ADCR2DACR_VOL

[Description]

Mixes the volume of the ADCR with that of the DACR.

[Syntax]

```
int ioctl (int fd,
           ACODEC_SET_ADCR2DACR_VOL,
           ACODEC_VOL_CTRL *arg);
```

[Parameter]



Parameter	Description	Input/Output
fd	File descriptor of the audio codec	Input
ACODEC_SET_ADCR2DACR_VOL	ioctl number	Input
arg	Pointer to ACODEC_VOL_CTRL	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

Header file: acodec.h

[Note]

Hi3518E V200 and Hi3519 V100 do not support this MPI.

[Example]

```
ACODEC_VOL_CTRL vol_ctrl;
vol_ctrl.vol_ctrl_mute = 0x0;
vol_ctrl.vol_ctrl = 0x06;
if (ioctl(s32Acodec_Fd, ACODEC_SET_ADCR2DACR, &vol_ctrl))
{
    printf("ioctl err!\n");
}
```

[See Also]

None

9.4 Data Structures

9.4.1 AI and AO

The AI and AO data structures are defined as follows:

- [AIO_MAX_NUM](#): Defines the maximum number of AI and AO devices.
- [AIO_MAX_CHN_NUM](#): Defines the maximum number of channels for AI and AO devices.
- [AI_HIFIVQE_MASK_HPF](#): Defines the mask value of the HPF function for Hi-Fi VQE.



- [**AI_HIFIVQE_MASK_RNR**](#): Defines the mask value of the RNR function for Hi-Fi VQE.
- [**AI_HIFIVQE_MASK_HDR**](#): Defines the mask value of the HDR function for Hi-Fi VQE.
- [**AI_HIFIVQE_MASK_DRC**](#): Defines the mask value of the DRC function for Hi-Fi VQE.
- [**AI_HIFIVQE_MASK_PEQ**](#): Defines the mask value of the PEQ function for Hi-Fi VQE.
- [**AI_TALKVQE_MASK_HPF**](#): Defines the mask value of the HPF function for Talk VQE.
- [**AI_TALKVQE_MASK_AEC**](#): Defines the mask value of the AEC function for Talk VQE.
- [**AI_TALKVQE_MASK_HDR**](#): Defines the mask value of the HDR function for Talk VQE.
- [**AI_TALKVQE_MASK_AGC**](#): Defines the mask value of the AGC function for Talk VQE.
- [**AI_TALKVQE_MASK_EQ**](#): Defines the mask value of the EQ function for Talk VQE.
- [**AI_TALKVQE_MASK_ANR**](#): Defines the mask value of the ANR function for Talk VQE.
- [**AO_VQE_MASK_HPF**](#): Defines the mask value of the AO VQE HPF function.
- [**AO_VQE_MASK_ANR**](#): Defines the mask value of the AO VQE ANR function.
- [**AO_VQE_MASK_AGC**](#): Defines the mask value of the AO VQE AGC function.
- [**AO_VQE_MASK_EQ**](#): Defines the mask value of the AO VQE EQ function.
- [**MAX_AUDIO_FILE_PATH_LEN**](#): Defines the maximum length of the path for the saved audio file.
- [**MAX_AUDIO_FILE_NAME_LEN**](#): Defines the maximum length of the name for the saved audio file.
- [**AUDIO_SAMPLE_RATE_E**](#): Defines an audio sampling rate.
- [**AUDIO_BIT_WIDTH_E**](#): Defines the audio sampling precision.
- [**AIO_MODE_E**](#): Defines the AI/AO working mode.
- [**AUDIO_SOUND_MODE_E**](#): Defines the audio sound-channel mode.
- [**AIO_ATTR_S**](#): Defines the attribute structure of an AI/AO device.
- [**AI_CHN_PARAM_S**](#): Defines the channel parameters.
- [**AUDIO_FRAME_S**](#): Defines the structure of an audio frame.
- [**AEC_FRAME_S**](#): Defines the structure of a reference echo cancellation frame.
- [**AUDIO_AGC_CONFIG_S**](#): Defines the configuration information about audio automatic gain control.
- [**AI_AEC_CONFIG_S**](#): Defines the configuration information about echo cancellation.
- [**AUDIO_ANR_CONFIG_S**](#): Defines the configuration information about audio noise reduction.
- [**AUDIO_HPF_FREQ_E**](#): Defines the cut-off frequency of audio high-pass filtering.
- [**AUDIO_HPF_CONFIG_S**](#): Defines the configuration information about audio high-pass filtering.
- [**AI_RNR_CONFIG_S**](#): Defines the configuration information about recording noise reduction.
- [**VQE_WORKSTATE_E**](#): Defines the VQE working mode.



- [VQE_EQ_BAND_NUM](#): Defines the number of frequency bands that can be adjusted by the equalizer.
- [AUDIO_EQ_CONFIG_S](#): Defines the configuration information about the audio equalizer.
- [AI_HDR_CONFIG_S](#): Defines the configuration information about the audio HDR function.
- [VQE_DRC_SECNUM](#): Defines the number of configurable levels for the DRC dynamic curve.
- [AI_DRC_CONFIG_S](#): Defines the configuration information structure of audio DRC function.
- [VQE_PEQ_BAND_NUM](#): Defines the maximum number of frequency bands that can be adjusted by the PEQ.
- [AI_PEQ_CONFIG_S](#): Defines the structure of the configuration information for the audio parametric equalizer.
- [AI_VQE_CONFIG_S](#): Defines the configuration information about AI VQE.
- [AI_HIFIVQE_CONFIG_S](#): Defines the configuration information structure of AI VQE (Hi-Fi).
- [AI_TALKVQE_CONFIG_S](#): Defines the configuration information structure of AI VQE (Talk).
- [AO_VQE_CONFIG_S](#): Defines the configuration information about AO VQE.
- [AUDIO_STREAM_S](#): Defines the structure of an audio stream.
- [AO_CHN_STATE_S](#): Defines the status of the data buffer on the AO channel.
- [AUDIO_TRACK_MODE_E](#): Defines the track mode of an audio device.
- [AUDIO_FADE_RATE_E](#): Defines the fade-in/fade-out rate of an AO device.
- [AUDIO_FADE_S](#): Defines the fade-in/fade-out settings of an AO device.
- [G726_BPS_E](#): Defines the bit rate for the G.726 encoding/decoding protocol.
- [ADPCM_TYPE_E](#): Defines the type of the ADPCM encoding/decoding protocol.
- [AUDIO_SAVE_FILE_INFO_S](#): Defines the configuration information about the function of saving audio files.
- [AUDIO_FILE_STATUS_S](#): Defines the audio file saving status structure.

AIO_MAX_NUM

[Description]

Defines the maximum number of AI and AO devices.

[Syntax]

```
#define AIO_MAX_NUM           1
```

[Note]

None

[See Also]

None



AIO_MAX_CHN_NUM

[Description]

Defines the maximum number of channels for AI and AO devices.

[Syntax]

```
#define AIO_MAX_CHN_NUM      16
```

[Note]

None

[See Also]

None

AI_HIFIVQE_MASK_HPF

[Description]

Defines the mask value of the HPF function for Hi-Fi VQE.

[Syntax]

```
#define AI_HIFIVQE_MASK_HPF      0x1
```

[Note]

Hi3518E V200 does not use the definition of this macro.

[See Also]

If this macro definition is assigned to the **u32OpenMask** member of the [AI_HIFIVQE_CONFIG_S](#) structure, the HPF function is enabled. For example, when the value is assigned to **u32OpenMask** as follows: `u32OpenMask = AI_HIFIVQE_MASK_RNR | AI_HIFIVQE_MASK_HPF`, the RNR and HPF functions are enabled.

AI_HIFIVQE_MASK_RNR

[Description]

Defines the mask value of the RNR function for Hi-Fi VQE.

[Syntax]

```
#define AI_HIFIVQE_MASK_RNR      0x2
```

[Note]

This macro definition is not supported by Hi3518E V200.

[See Also]

If this macro definition is assigned to the **u32OpenMask** member of the [AI_HIFIVQE_CONFIG_S](#) structure, the RNR function is enabled. For example, when the value is assigned to **u32OpenMask** as follows: `u32OpenMask = AI_HIFIVQE_MASK_RNR | AI_HIFIVQE_MASK_HPF`, the RNR and HPF functions are enabled.



AI_HIFIVQE_MASK_HDR

[Description]

Defines the mask value of the HDR function for Hi-Fi VQE.

[Syntax]

```
#define AI_HIFIVQE_MASK_HDR      0x4
```

[Note]

Hi3518E V200 does not use the definition of this macro.

[See Also]

If this macro definition is assigned to the **u32OpenMask** member of the [AI_HIFIVQE_CONFIG_S](#) structure, the HDR function is enabled. For example, when the value is assigned to **u32OpenMask** as follows: `u32OpenMask = AI_HIFIVQE_MASK_RNR | AI_HIFIVQE_MASK_HDR`, the RNR and HDR functions are enabled.

AI_HIFIVQE_MASK_DRC

[Description]

Defines the mask value of the DRC function for Hi-Fi VQE.

[Syntax]

```
#define AI_HIFIVQE_MASK_DRC      0x8
```

[Note]

Hi3518E V200 does not use the definition of this macro.

[See Also]

If this macro definition is assigned to the **u32OpenMask** member of the [AI_HIFIVQE_CONFIG_S](#) structure, the DRC function is enabled. For example, when the value is assigned to **u32OpenMask** as follows: `u32OpenMask = AI_HIFIVQE_MASK_RNR | AI_HIFIVQE_MASK_DRC`, the RNR and DRC functions are enabled.

AI_HIFIVQE_MASK_PEQ

[Description]

Defines the mask value of the PEQ function for Hi-Fi VQE.

[Syntax]

```
#define AI_HIFIVQE_MASK_PEQ      0x10
```

[Note]

This macro definition is not supported by Hi3518E V200.

[See Also]

If this macro definition is assigned to the **u32OpenMask** member of the [AI_HIFIVQE_CONFIG_S](#) structure, the PEQ function is enabled. For example, when the



value is assigned to **u32OpenMask** as follows: $u32OpenMask = AI_HIFIVQE_MASK_RNR | AI_HIFIVQE_MASK_PEQ$, the RNR and PEQ functions are enabled.

AI_TALKVQE_MASK_HPF

[Description]

Defines the mask value of the HPF function for Talk VQE.

[Syntax]

```
#define AI_TALKVQE_MASK_HPF      0x1
```

[Note]

This macro definition is not supported by Hi3518E V200 and Hi3516A.

[See Also]

If this macro definition is assigned to the **u32OpenMask** member of the **AI_TALKVQE_CONFIG_S** structure, the HPF function is enabled. For example, when the value is assigned to **u32OpenMask** as follows: $u32OpenMask = AI_TALKVQE_MASK_AEC | AI_TALKVQE_MASK_HPF$, the AEC and HPF functions are enabled.

AI_TALKVQE_MASK_AEC

[Description]

Defines the mask value of the AEC function for Talk VQE.

[Syntax]

```
#define AI_TALKVQE_MASK_AEC      0x2
```

[Note]

This macro definition is not supported by Hi3518E V200 and Hi3516A.

[See Also]

If this macro definition is assigned to the **u32OpenMask** member of the **AI_TALKVQE_CONFIG_S** structure, the AEC function is enabled. For example, when the value is assigned to **u32OpenMask** as follows: $u32OpenMask = AI_TALKVQE_MASK_AEC | AI_TALKVQE_MASK_HPF$, the AEC and HPF functions are enabled.

AI_TALKVQE_MASK_HDR

[Description]

Defines the mask value of the HDR function for Talk VQE.

[Syntax]

```
#define AI_TALKVQE_MASK_HDR      0x4
```

[Note]

This macro definition is not supported by Hi3518E V200 and Hi3516A.



[See Also]

If this macro definition is assigned to the **u32OpenMask** member of the [AI_TALKVQE_CONFIG_S](#) structure, the HDR function is enabled. For example, when the value is assigned to **u32OpenMask** as follows: u32OpenMask = AI_TALKVQE_MASK_AEC | AI_TALKVQE_MASK_HDR, the AEC and HDR functions are enabled.

AI_TALKVQE_MASK_AGC

[Description]

Defines the mask value of the AGC function for Talk VQE.

[Syntax]

```
#define AI_TALKVQE_MASK_AGC      0x8
```

[Note]

This macro definition is not supported by Hi3518E V200 and Hi3516A.

[See Also]

If this macro definition is assigned to the **u32OpenMask** member of the [AI_TALKVQE_CONFIG_S](#) structure, the AGC function is enabled. For example, when the value is assigned to **u32OpenMask** as follows: u32OpenMask = AI_TALKVQE_MASK_AEC | AI_TALKVQE_MASK_AGC, the AEC and AGC functions are enabled.

AI_TALKVQE_MASK_EQ

[Description]

Defines the mask value of the EQ function for Talk VQE.

[Syntax]

```
#define AI_TALKVQE_MASK_EQ      0x10
```

[Note]

This macro definition is not supported by Hi3518E V200 and Hi3516A.

[See Also]

If this macro definition is assigned to the **u32OpenMask** member of the [AI_TALKVQE_CONFIG_S](#) structure, the EQ function is enabled. For example, when the value is assigned to **u32OpenMask** as follows: u32OpenMask = AI_TALKVQE_MASK_AEC | AI_TALKVQE_MASK_EQ, the AEC and EQ functions are enabled.

AI_TALKVQE_MASK_ANR

[Description]

Defines the mask value of the ANR function for Talk VQE.

[Syntax]



```
#define AI_TALKVQE_MASK_ANR      0x20
```

[Note]

This macro definition is not supported by Hi3518E V200 and Hi3516A.

[See Also]

If this macro definition is assigned to the **u32OpenMask** member of the [AI_TALKVQE_CONFIG_S](#) structure, the ANR function is enabled. For example, when the value is assigned to **u32OpenMask** as follows: **u32OpenMask = AI_TALKVQE_MASK_AEC | AI_TALKVQE_MASK_ANR**, the AEC and ANR functions are enabled.

AO_VQE_MASK_HPF

[Description]

Defines the mask value of the AO VQE HPF function.

[Syntax]

```
#define AO_VQE_MASK_HPF      0x1
```

[Note]

Hi3518E V200 and the Hi3516A do not have this macro definition.

[See Also]

If this macro definition is assigned to the **u32OpenMask** member of the [AO_VQE_CONFIG_S](#) structure, the HPF function is enabled. For example, when the value is assigned to **u32OpenMask** as follows: **u32OpenMask = AO_VQE_MASK_AGC | AO_VQE_MASK_HPF**, the AGC and HPF functions are enabled.

AO_VQE_MASK_ANR

[Description]

Defines the mask value of the AO VQE ANR function.

[Syntax]

```
#define AI_TALKVQE_MASK_ANR      0x2
```

[Note]

Hi3518E V200 and the Hi3516A do not have this macro definition.

[See Also]

If this macro definition is assigned to the **u32OpenMask** member of the [AO_VQE_CONFIG_S](#) structure, the ANR function is enabled. For example, when the value is assigned to **u32OpenMask** as follows: **u32OpenMask = AO_VQE_MASK_AGC | AO_VQE_MASK_ANR**, the AGC and ANR functions are enabled.

AO_VQE_MASK_AGC

[Description]



Defines the mask value of the AO VQE AGC function.

[Syntax]

```
#define AO_VQE_MASK_AGC      0x4
```

[Note]

Hi3518E V200 and the Hi3516A do not have this macro definition.

[See Also]

If this macro definition is assigned to the **u32OpenMask** member of the **AO_VQE_CONFIG_S** structure, the AGC function is enabled. For example, when the value is assigned to **u32OpenMask** as follows: `u32OpenMask = AO_VQE_MASK_AGC | AO_VQE_MASK_EQ`, the AGC and EQ functions are enabled.

AO_VQE_MASK_EQ

[Description]

Defines the mask value of the AO VQE EQ function.

[Syntax]

```
#define AO_VQE_MASK_EQ      0x8
```

[Note]

Hi3518E V200 and the Hi3516A do not have this macro definition.

[See Also]

If this macro definition is assigned to the **u32OpenMask** member of the **AO_VQE_CONFIG_S** structure, the EQ function is enabled. For example, when the value is assigned to **u32OpenMask** as follows: `u32OpenMask = AO_VQE_MASK_AGC | AO_VQE_MASK_EQ`, the AGC and EQ functions are enabled.

MAX_AUDIO_FILE_PATH_LEN

[Description]

Defines the maximum length of the path for the saved audio file.

[Syntax]

```
#define MAX_AUDIO_FILE_PATH_LEN      256
```

[Note]

None

[See Also]

[AUDIO_SAVE_FILE_INFO_S](#)

MAX_AUDIO_FILE_NAME_LEN

[Description]

Defines the maximum length of the name for the saved audio file.



[Syntax]

```
#define MAX_AUDIO_FILE_NAME_LEN      256
```

[Note]

None

[See Also]

[AUDIO_SAVE_FILE_INFO_S](#)

AUDIO_SAMPLE_RATE_E

[Description]

Defines an audio sampling rate.

[Syntax]

```
typedef enum hiAUDIO_SAMPLE_RATE_E
{
    AUDIO_SAMPLE_RATE_8000 =8000,           /* 8kHz sampling rate */
    AUDIO_SAMPLE_RATE_12000=12000,          /* 12kHz sampling rate */
    AUDIO_SAMPLE_RATE_11025=11025,          /* 11.025kHz sampling rate */
    AUDIO_SAMPLE_RATE_16000=16000,          /* 16kHz sampling rate */
    AUDIO_SAMPLE_RATE_22050=22050,          /* 22.050kHz sampling rate */
    AUDIO_SAMPLE_RATE_24000=24000,          /* 24kHz sampling rate */
    AUDIO_SAMPLE_RATE_32000=32000,          /* 32kHz sampling rate */
    AUDIO_SAMPLE_RATE_44100=44100,          /* 44.1kHz sampling rate */
    AUDIO_SAMPLE_RATE_48000=48000,          /* 48kHz sampling rate */
    AUDIO_SAMPLE_RATE_64000=64000,          /* 64K samplerate*/
    AUDIO_SAMPLE_RATE_96000=96000,          /* 96K samplerate*/
    AUDIO_SAMPLE_RATE_BUTT,
}AUDIO_SAMPLE_RATE_E;
```

[Member]

Member	Description
AUDIO_SAMPLE_RATE_8000	Sampling rate of 8 kHz
AUDIO_SAMPLE_RATE_12000	Sampling rate of 12 kHz
AUDIO_SAMPLE_RATE_11025	Sampling rate of 11025 kHz
AUDIO_SAMPLE_RATE_16000	Sampling rate of 16 kHz
AUDIO_SAMPLE_RATE_22050	Sampling rate of 22050 kHz
AUDIO_SAMPLE_RATE_24000	Sampling rate of 24 kHz
AUDIO_SAMPLE_RATE_32000	Sampling rate of 32 kHz
AUDIO_SAMPLE_RATE_44100	Sampling rate of 441 kHz



Member	Description
AUDIO_SAMPLE_RATE_48000	Sampling rate of 48 kHz
AUDIO_SAMPLE_RATE_64000	Sampling rate of 64 kHz
AUDIO_SAMPLE_RATE_96000	Sampling rate of 96 kHz

[Note]

- The value is not enumerated from 0. Instead, the value is consistent with the actual sampling rate.
- The data with the sampling rate of 64 kHz or 96 kHz applies only to AI capturing and AO playback, and such data cannot be used for resampling and VQE processing.

[See Also]

[AIO_ATTR_S](#)

AUDIO_BIT_WIDTH_E

[Description]

Defines the audio sampling precision.

[Syntax]

```
typedef enum hiAUDIO_BIT_WIDTH_E
{
    AUDIO_BIT_WIDTH_8      =0,      /* 8bit width */
    AUDIO_BIT_WIDTH_16     =1,      /* 16bit width */
    AUDIO_BIT_WIDTH_24     =2,      /* 24bit width */
    AUDIO_BIT_WIDTH_BUTT,
}AUDIO_BIT_WIDTH_E;
```

[Member]

Member	Description
AUDIO_BIT_WIDTH_8	Sampling precision of 8 bits
AUDIO_BIT_WIDTH_16	Sampling precision of 16 bits
AUDIO_BIT_WIDTH_24	Sampling precision of 24 bits

[Note]

Only 16-bit sampling precision is supported currently.

[See Also]

[AIO_ATTR_S](#)



AIO_MODE_E

[Description]

Defines the AI/AO working mode.

[Syntax]

```
typedef enum hiAIO_MODE_E
{
    AIO_MODE_I2S_MASTER = 0,          /* I2S master mode */
    AIO_MODE_I2S_SLAVE = 1,           /* I2S slave mode */
    AIO_MODE_PCM_SLAVE_STD,          /* SIO PCM slave standard mode */
    AIO_MODE_PCM_SLAVE_NSTD,         /* SIO PCM slave non-standard mode */
    AIO_MODE_PCM_MASTER_STD,         /* SIO PCM master standard mode */
    AIO_MODE_PCM_MASTER_NSTD,        /* SIO PCM master non-standard mode */
    AIO_MODE_BUTT
}AIO_MODE_E;
```

[Member]

Member	Description
AIO_MODE_I2S_MASTER	Master I ² S mode
AIO_MODE_I2S_SLAVE	Slave I ² S mode
AIO_MODE_PCM_SLAVE_STD	Slave PCM mode (standard protocol)
AIO_MODE_PCM_SLAVE_NSTD	Slave PCM mode (customized protocol)
AIO_MODE_PCM_MASTER_STD	Master PCM mode (standard protocol)
AIO_MODE_PCM_MASTER_NSTD	Master PCM mode (customized protocol)

[Note]

When the embedded audio codec of the Hi35xx is connected, only the I²S master mode is supported.

[See Also]

[AIO_ATTR_S](#)

AUDIO_SOUND_MODE_E

[Description]

Defines the audio sound-channel mode.

[Syntax]

```
typedef enum hiAIO_SOUND_MODE_E
{
    AUDIO_SOUND_MODE_MONO      =0,      /*Mono*/
    AUDIO_SOUND_MODE_STEREO    =1,      /*Stereo*/
    AUDIO_SOUND_MODE_DUAL      =2,      /*Dual*/
    AUDIO_SOUND_MODE_QUAD      =3,      /*Quad*/
}
```



```
AUDIO_SOUND_MODE_STEREO =1,      /*Stereo*/  
AUDIO_SOUND_MODE_BUTT  
}AUDIO_SOUND_MODE_E;
```

[Member]

Member	Description
AUDIO_SOUND_MODE_MONO	Mono
AUDIO_SOUND_MODE_STEREO	Stereo

[Note]

In stereo mode, the channels whose channel IDs are smaller than half of the number of channels are audio-left channels, and the other channels are audio-right channels. You need to operate only audio-left channels, because the corresponding audio-right channels are automatically operated in the SDK.

[See Also]

[AIO_ATTR_S](#)

AIO_ATTR_S

[Description]

Defines the attribute structure of an AI/AO device.

[Syntax]

```
typedef struct hiAIO_ATTR_S  
{  
    AUDIO_SAMPLE_RATE_E enSamplerate;           /*Sampling rate*/  
    AUDIO_BIT_WIDTH_E   enBitwidth;              /*Bit width*/  
    AIO_MODE_E          enWorkmode;              /*Master or slave mode*/  
    AUDIO_SOUND_MODE_E  enSoundmode;             /*Mono or stereo*/  
    HI_U32               u32EXFlag;                /*Expand 8 bits to 16bits */  
    HI_U32               u32FrmNum;                /*Number of frames in a  
buffer*/  
    HI_U32               u32PtNumPerFrm;            /*Number of samples*/  
    HI_U32               u32ChnCnt;  
    HI_U32               u32ClkSel;  
}AIO_ATTR_S;
```

[Member]

Member	Description
enSamplerate	Audio sampling rate (in slave mode, this parameter is invalid) Static attribute



Member	Description
enBitwidth	Audio sampling precision (in slave mode, this parameter must match the sampling precision of the audio AD/DA) Static attribute
enWorkmode	Working mode of an AI/AO device Static attribute
enSoundmode	Audio sound-channel mode Static attribute
u32EXFlag	The values are as follows: <ul style="list-style-type: none">• 0: not extend• 1: 8-bit to 16-bit extension flag (It is valid when the AI sampling precision is 8 bits.)• 2: crop from 24 bits to 16 bits. It can be used in the external codec scenario. Static attribute. This parameter is reserved, and is set to 1 in general.
u32FrmNum	Number of buffered frames Value range: [2, MAX_AUDIO_FRAME_NUM] Static attribute
u32PtNumPerFrm	Number of sampling points for one frame Value range: In G711, G726, and ADPCM_DVI4, the value is 80, 160, 240, 320, or 480; in ADPCM_IMA, the value is 81, 161, 241, 321, or 481. The value range of the length of an AI frame is [80, 2048]. The value range of the length of an AO frame is [80, 4096]. Static attribute
u32ChnCnt	Number of channels Value: 1, 2, 4, 8, or 16 (A maximum of 16 input channels and two output channels are supported.)
u32ClkSel	Whether the frame clock and bit stream clock of AiDev0 are multiplexed with those of AoDev0 The values are as follows: <ul style="list-style-type: none">• 0: not multiplex• 1: multiplex

[Note]

None

[See Also]

[HI_MPI_AI_SetPubAttr](#)



AI_CHN_PARAM_S

[Description]

Defines the channel parameters.

[Syntax]

```
typedef struct hiAI_CHN_PARAM_S
{
    HI_U32 u32UsrFrmDepth;
} AI_CHN_PARAM_S;
```

[Member]

Member	Description
u32UsrFrmDepth	Depth of the buffer for storing audio frames

[Note]

None

[See Also]

None

AUDIO_FRAME_S

[Description]

Defines the structure of an audio frame.

[Syntax]

```
typedef struct hiAUDIO_FRAME_S
{
    AUDIO_BIT_WIDTH_E   enBitwidth; /*Audio frame bit width*/
    AUDIO_SOUND_MODE_E enSoundmode; /*Audio frame mono or stereo mode*/
    HI_VOID             *pVirAddr[2];
    HI_U32              u32PhyAddr[2];
    HI_U64              u64TimeStamp; /*audio frame timestamp*/
    HI_U32              u32Seq;      /*audio frame seq*/
    HI_U32              u32Len;       /*data length per channel in frame*/
    HI_U32              u32PoolId[2];
}AUDIO_FRAME_S;
```

[Member]

Member	Description
enBitwidth	Audio sampling precision



Member	Description
enSoundmode	Audio sound-channel mode
pVirAddr[2]	Virtual address for an audio frame
u32PhyAddr[2]	Physical address for an audio frame
u64TimeStamp	Time stamp of an audio frame, in μ s
u32Seq	Sequence number of an audio frame
u32Len	Length of an audio frame, in byte
u32PoolId[2]	ID of the audio frame buffer

[Note]

- **u32Len** (length of an audio frame) indicates the length of data on a mono channel.
- Mono data is directly stored, for which the sampling points are specified by **ptnum** and the length of bytes is specified by **len**. Stereo data is stored in the audio-left channel and audio-right channel in sequence, for which the sampling points are specified by **ptnum** and the length in byte is specified by **len**.

[See Also]

None

AEC_FRAME_S

[Description]

Defines the structure of a reference echo cancellation frame.

[Syntax]

```
typedef struct hiAEC_FRAME_S
{
    AUDIO_FRAME_S    stRefFrame;      /* aec reference audio frame */
    HI_BOOL          bValid;         /* whether frame is valid */
    HI_BOOL          bSysBind;       /* whether is sysbind */
} AEC_FRAME_S;
```

[Member]

Member	Description
stRefFrame	Structure of a reference echo cancellation frame
bValid	Flag indicating an invalid reference frame Value range: HI_TRUE: a valid reference frame HI_FALSE: an invalid reference frame. An invalid reference frame cannot be used to perform echo cancellation.



Member	Description
bSysBind	Whether the AIU is bound to the AENC in system binding mode

[Note]

None

[See Also]

None

AUDIO_AGC_CONFIG_S

[Description]

Defines the configuration information about audio automatic gain control.

[Syntax]

```
typedef struct hiAUDIO_AGC_CONFIG_S
{
    HI_BOOL bUsrMode;
    HI_S8 s8TargetLevel;
    HI_S8 s8NoiseFloor;
    HI_S8 s8MaxGain;
    HI_S8 s8AdjustSpeed;
    HI_S8 s8ImproveSNR;
    HI_S8 s8UseHighPassFilt;
    HI_S8 s8OutputMode;
    HI_S16 s16NoiseSupSwitch;
    HI_S32 s32Reserved;
} AUDIO_AGC_CONFIG_S;
```

[Member]

Member	Description
bUsrMode	Mode select 0 (default): automatic mode 1: user mode
s8TargetLevel	Target level, maximum level threshold after AGC processing; 1 dB adjustment step Value range: [-40 dB, -1 dB]
s8NoiseFloor	Noise bottom value, 1 dB adjustment step Value range: [-65 dB, -20 dB] Note: The noise bottom value takes effect only after the output mode is enabled.



Member	Description
s8MaxGain	Maximum gain, 1 dB adjustment step Value range: [0 dB, 30 dB]
s8AdjustSpeed	Adjustment speed, 1 dB/s adjustment step Value range: [0 dB/s, 10 dB/s]
s8ImproveSNR	Enable for improving the signal-to-noise ratio (SNR) 0: no improvement 1: improved by at most 3 dB 2: improved by at most 6 dB
s8UseHighPassFilt	HPF enable. The value indicates the configured HPF cutoff frequency in the AGC module. 0: disabled 1: 80 Hz 2: 120 Hz 3: 150 Hz 4: 300 Hz 5: 500 Hz
s8OutputMode	Output mode. The outputs are muted for signals lower below NoiseFloor . 0: disabled 1: enabled Note: The output mode works with the noise bottom value. If s8OutputMode is set to 1, the outputs of the signals below the noise bottom value are muted.
s16NoiseSupSwitch	Noise suppression enable 0: disabled 1: enabled
s32Reserved	Reserved

[Note]

- The preceding advanced parameters take effect only when the user mode is enabled. If the user mode is disabled, the default parameter configurations corresponding to **enWorkstate** in [AI_VQE_CONFIG_S](#), [AI_TALKVQE_CONFIG_S](#), or [AO_VQE_CONFIG_S](#) take effect.
- During parameter configuration, the correctness of advanced parameters is checked only when the user mode is enabled. The advanced parameters can be successfully configured only when they are correct.

[See Also]

- [AI_VQE_CONFIG_S](#)
- [AO_VQE_CONFIG_S](#)



AI_AEC_CONFIG_S

[Description]

Defines the configuration information about echo cancellation.

[Syntax]

```
typedef struct hiAI_AEC_CONFIG_S
{
    HI_BOOL bUsrMode;
    HI_S8 s8CngMode;      /* cosy-noisy mode:0 close,1 open, default 1*/
    HI_S8 s8NearAllPassEnergy;
    HI_S8 s8NearCleanSupEnergy;
    HI_S16 s16DTTHnlSortQTh;
    HI_S16 s16EchoBandLow;
    HI_S16 s16EchoBandHigh;
    HI_S16 s16EchoBandLow2;
    HI_S16 s16EchoBandHigh2;
    HI_S16 s16ERLBand[6];
    HI_S16 s16ERL[7];
    HI_S16 s16VioceProtectFreqL;
    HI_S16 s16VioceProtectFreqL1;
    HI_S32 s32Reserved;
} AI_AEC_CONFIG_S;
```

[Member]

Member	Description
bUsrMode	Mode select 0 (default): automatic mode 1: user mode
s8CngMode	Comfort noise mode enable 0: disabled 1: enabled
s8NearAllPassEnergy	Remote energy threshold for determining whether passthrough is implemented at the near end Default value: 1 0: -59 dBm0 1: -49 dBm0 2: -39 dBm0



Member	Description
s8NearCleanSupEnergy	Energy threshold for forcibly clearing the near-end signal Default value: 2 0: 12 dB 1: 15 dB 2: 18 dB
s16DTHnlSortQTh	Threshold for distinguishing the single talk and double talk, marked as Q15 Default value: 16384 Value range: [0, 32767]
s16EchoBandLow	Low-frequency parameter for voice processing frequency band 1. The value range is [1, 63] at the 8 kHz sampling rate, and is [1, 127] at the 16 kHz sampling rate. Default value: 10
s16EchoBandHigh	High-frequency parameter for voice processing frequency band 1. The value range is (s16EchoBandLow, 63] at the 8 kHz sampling rate, and is (s16EchoBandLow, 127] at the 16 kHz sampling rate. Default value: 41
s16EchoBandLow2	Low-frequency parameter for voice processing frequency band 2. The value range is [1, 63] at the 8 kHz sampling rate, and is [1, 127] at the 16 kHz sampling rate. Default value: 47
s16EchoBandHigh2	High-frequency parameter for voice processing frequency band 2. The value range is (s16EchoBandLow2, 63] at the 8 kHz sampling rate. Default value: 63 The value range is (s16EchoBandLow2, 127] at the 16 kHz sampling rate. Default value: 72
s16ERLBand[6]	Array for the echo return loss (ERL) protection frequency band. The value range of the array parameters is [1, 63] at the 8 kHz sampling rate, and is [1, 127] at the 16 kHz sampling rate. The array parameters must be in ascending order, that is, s16ERLBand[n + 1] must be greater than or equal to s16ERLBand[n]. The recommended values are {4, 6, 36, 49, 50, 51}.
s16ERL[7]	Array for the ERL frequency band protection values. This array works with s16ERLBand[6]. s16ERLBand sets the value of s16ERL by the frequency band. That is, 0–s16ERLBand[0] correspond to s16ERL[0], s16ERLBand[0]–s16ERLBand[1] correspond to s16ERL[1]. s16ERLBand[5]–MaxBand correspond to s16ERL[6]. MaxBand is 65 in 8 kHz mode, and 129 in 16 kHz mode. A smaller value indicates greater protection strength. The value range is [0, 18]. The recommended values are {7, 10, 16, 10, 18, 18, 18}.



Member	Description
s16VioceProtectFreqL	Frequency parameter for the near-end low-frequency protection region. The value range is [1, 63] at the 8 kHz sampling rate, and is [1, 127] at the 16 kHz sampling rate. Default value: 3
s16VioceProtectFreqL1	Frequency parameter 1 for the near-end low-frequency protection region. The value range is (s16VioceProtectFreqL , 63] at the 8 kHz sampling rate, and is (s16VioceProtectFreqL , 127] at the 16 kHz sampling rate. Default value: 6
s32Reserved	Reserved

[Note]

- When the user mode is enabled, all preceding parameters take effect except **UserMode**. If the user mode is disabled, configure the mode based on the default value of **enWorkstate** in [AI_VQE_CONFIG_S](#) or [AI_TALKVQE_CONFIG_S](#).
- During parameter configuration, the correctness of advanced parameters is checked only when the user mode is enabled. The advanced parameters can be successfully configured only when they are correct.

[See Also]

[AI_VQE_CONFIG_S](#)

AUDIO_ANR_CONFIG_S

[Description]

Defines the configuration information about audio noise reduction.

[Syntax]

```
typedef struct hiAUDIO_ANR_CONFIG_S
{
    HI_BOOL bUsrMode;
    HI_S16 s16NrIntensity;
    HI_S16 s16NoiseDbThr;
    HI_S8  s8SpProSwitch;
    HI_S32 s32Reserved;
} AUDIO_ANR_CONFIG_S;
```

[Member]

Member	Description
bUsrMode	Mode select 0 (default): automatic mode 1: user mode



Member	Description
s16NrIntensity	NR strength. A larger value indicates higher NR strength but more details losses/damage. Value range: [0, 25]
s16NoiseDbThr	Noise threshold. A larger value indicates weaker detection strength and smoother voice. Value range: [30, 60]
s8SpProSwitch	Music detection enable. After this function is enabled, the music details are detected. It is recommended that this function be disabled in noisy scenarios. Value: 0 or 1
s32Reserved	Reserved

[Note]

- The preceding advanced parameters take effect only when the user mode is enabled. If the user mode is disabled, the default parameter configurations corresponding to **enWorkstate** in [AI_VQE_CONFIG_S](#), [AI_TALKVQE_CONFIG_S](#), or [AO_VQE_CONFIG_S](#) take effect.
- During parameter configuration, the correctness of advanced parameters is checked only when the user mode is enabled. The advanced parameters can be successfully configured only when they are correct.

[See Also]

- [AI_VQE_CONFIG_S](#)
- [AO_VQE_CONFIG_S](#)

AUDIO_HPF_FREQ_E

[Description]

Defines the cut-off frequency of audio high-pass filtering.

[Syntax]

```
typedef enum hiAUDIO_HPF_FREQ_E
{
    AUDIO_HPF_FREQ_80    = 80,      /* 80 Hz */
    AUDIO_HPF_FREQ_120   = 120,     /* 120 Hz */
    AUDIO_HPF_FREQ_150   = 150,     /* 150 Hz */
    AUDIO_HPF_FREQ_BUTT,
} AUDIO_HPF_FREQ_E;
```

[Member]



Member	Description
AUDIO_HPF_FREQ_80	80 Hz cut-off frequency
AUDIO_HPF_FREQ_120	120 Hz cut-off frequency
AUDIO_HPF_FREQ_150	150 Hz cut-off frequency

[Note]

The default cut-off frequency is 150 Hz.

[See Also]

None

AUDIO_HPF_CONFIG_S

[Description]

Defines the configuration information about audio high-pass filtering.

[Syntax]

```
typedef struct hiAUDIO_HPF_CONFIG_S
{
    HI_BOOL bUsrMode;
    AUDIO_HPF_FREQ_E enHpfFreq; /*freq to be processed*/
} AUDIO_HPF_CONFIG_S;
```

[Member]

Member	Description
bUsrMode	Mode select 0 (default): automatic mode 1: user mode
enHpfFreq	Cut-off frequency of high-pass filtering 80: 80 Hz 120: 120 Hz 150: 150 Hz

[Note]

- The advanced parameter takes effect only when the user mode is enabled. If the user mode is disabled, configure the mode based on the default value of **enWorkstate** in [AI_VQE_CONFIG_S](#), [AI_TALKVQE_CONFIG_S](#), [AI_HIFIVQE_CONFIG_S](#), or [AO_VQE_CONFIG_S](#).
- During parameter configuration, the correctness of advanced parameters is checked only when the user mode is enabled. The advanced parameters can be successfully configured only when they are correct.



[See Also]

- [AI_VQE_CONFIG_S](#)
- [AO_VQE_CONFIG_S](#)

AI_RNR_CONFIG_S

[Description]

Defines the configuration information about recording noise reduction.

[Syntax]

```
typedef struct hiAI_RNR_CONFIG_S
{
    HI_BOOL bUsrMode;
    HI_S32 s32NrMode;
    HI_S32 s32MaxNrLevel;
    HI_S32 s32NoiseThresh;
} AI_RNR_CONFIG_S;
```

[Member]

Member	Description
bUsrMode	Mode select 0 (default): automatic mode 1: user mode
s32NrMode	NR mode 0 (default): Reduce the background noise. 1: Reduce the environment noise.
s32MaxNrLevel	Maximum NR capability, 1 dB adjustment step Value range: [2 dB, 20 dB] Default value: 15 dB
s32NoiseThresh	Noise threshold Value range: [-80 dB, -20 dB] Default value: -55 dB

[Note]

- The advanced parameter takes effect only when the user mode is enabled. If the user mode is disabled, configure the mode based on the default value of **enWorkstate** in [AI_VQE_CONFIG_S](#) or [AI_HIFIVQE_CONFIG_S](#).
- During parameter configuration, the correctness of advanced parameters is checked only when the user mode is enabled. The advanced parameters can be successfully configured only when they are correct.
- When **s32NrMode** is 1, the 48 kHz working sampling rate (**s32WorkSampleRate**) is not supported.



[See Also]

[AI_VQE_CONFIG_S](#)

VQE_WORKSTATE_E

[Description]

Defines the VQE working mode.

[Syntax]

```
typedef enum hiVQE_WORKSTATE_E
{
    VQE_WORKSTATE_COMMON = 0,
    VQE_WORKSTATE_MUSIC = 1,
    VQE_WORKSTATE_NOISY = 2
} VQE_WORKSTATE_E;
```

[Member]

Member	Description
VQE_WORKSTATE_COMMON	General mode
VQE_WORKSTATE_MUSIC	Music mode
VQE_WORKSTATE_NOISY	Noise mode

[Note]

None

[See Also]

- [AI_VQE_CONFIG_S](#)
- [AO_VQE_CONFIG_S](#)

VQE_EQ_BAND_NUM

[Description]

Defines the number of frequency bands that can be adjusted by the equalizer.

[Syntax]

```
#define VQE_EQ_BAND_NUM 10
```

[Note]

None

[See Also]

[AUDIO_EQ_CONFIG_S](#)



AUDIO_EQ_CONFIG_S

[Description]

Defines the configuration information about the audio equalizer.

[Syntax]

```
typedef struct hiAUDIO_EQ_CONFIG_S
{
    HI_S8    s8GaindB[VQE_EQ_BAND_NUM];
    HI_S32   s32Reserved;
} AUDIO_EQ_CONFIG_S;
```

[Member]

Member	Description
s8GaindB	Gain adjustment for the EQ frequency bands (100 Hz, 200 Hz, 250 Hz, 350 Hz, 500 Hz, 800 Hz, 1.2 kHz, 2.5 kHz, 4 kHz, and 8 kHz). The 8 kHz frequency band is valid only when the working sampling rate (s32WorkSampleRate) is set to 16 kHz. The value range of each frequency band is [-100, +20] dB. The adjustment step is 1 dB.
s32Reserved	Reserved

[Note]

The equalizer supports only the user mode, and does not support the automatic mode. To validate the equalizer, the corresponding advanced parameter needs to be configured.

[See Also]

None

AI_HDR_CONFIG_S

[Description]

Defines the configuration information about the audio HDR function.

[Syntax]

```
typedef struct hiAI_HDR_CONFIG_S
{
    HI_BOOL bUsrMode;
    HI_S32 s32MinGaindB;
    HI_S32 s32MaxGaindB;
    HI_S32 s32MicGaindB;
    HI_S32 s32MicGainStepdB;
    pFuncGainCallBack pcallback;
} AI_HDR_CONFIG_S;
```

[Member]



Member	Description
bUsrMode	Mode select 0: automatic mode 1: user mode The default value 0 indicates that the default built-in audio codec adjustment mode is applied.
s32MinGaindB	Minimum configured gain allowed by the codec Value range: [0, 120]
s32MaxGaindB	Maximum configured gain allowed by the codec Value range: [0, 120]
s32MicGaindB	Current configured gain of the codec Value range: [s32MinGaindB, s32MaxGaindB]
s32MicGainStepdB	Gain adjustment step Default value: 2 Value range: [1,3]
pcallback	Function pointer used to change the codec gain

[Note]

- To use the audio HDR function, you need to configure corresponding parameters based on the codec.
- The preceding advanced parameters take effect only when the user mode is enabled. If the user mode is disabled, the default parameter configurations corresponding to **enWorkstate** in **AI_VQE_CONFIG_S**, **AI_TALKVQE_CONFIG_S**, or **AI_HIFIVQE_CONFIG_S** take effect.
- During parameter configuration, the correctness of advanced parameters is checked only when the user mode is enabled. The advanced parameters can be successfully configured only when they are correct.

[See Also]

None

VQE_DRC_SECNUM

[Description]

Defines the number of configurable levels for the DRC dynamic curve.

[Syntax]

```
#define VQE_DRC_SECNUM 5
```

[Note]

None

[See Also]



AI_DRC_CONFIG_S

[Description]

Defines the configuration information structure of the audio DRC function.

[Syntax]

```
typedef struct hiAI_DRC_CONFIG_S
{
    HI_BOOL bUsrMode;
    HI_S16 s16AttackTime;
    HI_S16 s16ReleaseTime;
    HI_S16 s16OldLevDb[VQE_DRC_SECNUM];
    HI_S16 s16NewLevDb[VQE_DRC_SECNUM];
} AI_DRC_CONFIG_S;
```

[Member]

Member	Description
bUsrMode	Mode select 0: automatic mode 1: user mode
s16AttackTime	Time period for the signal to change to become weaker (ms) Value range: [10, 250]
s16ReleaseTime	Time period for the signal to become stronger (ms) Value range: [10, 250]
s16OldLevDb	Knee point level Q4 before dynamic curve adjustment. The values are stored in descending order, that is, s16OldLevDb[n] is greater than or equal to s16OldLevDb[n+1] . Besides, s16NewLevDb[n] must be greater than or equal to s16OldLevDb[n] . Value range: [-1440, 0] Default value: [0, -472, -792, -960, -1280]
s16NewLevDb	Knee point level Q4 after dynamic curve adjustment. The values are stored in descending order, that is, s16NewLevDb[n] is greater than or equal to s16NewLevDb[n+1] . Besides, s16NewLevDb[n] must be greater than or equal to s16OldLevDb[n] . Value range: [-1440, 0] Default value: [0, -174, -410, -608, -1021]

[Note]



- The preceding advanced parameters take effect only when the user mode is enabled. If the user mode is disabled, the default parameter configurations corresponding to **enWorkstate** in [AI_HIFIVQE_CONFIG_S](#) take effect.
- During parameter configuration, the correctness of advanced parameters is checked only when the user mode is enabled. The advanced parameters can be successfully configured only when they are correct.
- This function supports only the 48 kHz sampling rate.

[See Also]

None

VQE_PEQ_BAND_NUM

[Description]

Defines the maximum number of frequency bands that can be adjusted by the PEQ.

[Syntax]

```
#define VQE_PEQ_BAND_NUM 10
```

[Note]

None

[See Also]

[AUDIO_EQ_CONFIG_S](#)

AI_PEQ_CONFIG_S

[Description]

Defines the structure of the configuration information for the audio parametric equalizer.

[Syntax]

```
typedef struct hiAI_PEQ_CONFIG_S
{
    HI_BOOL bUsrMode;
    HI_U32 u32BandNum;
    HI_U8  u8FilterType[VQE_PEQ_BAND_NUM];
    HI_S8  s8GaindB[VQE_PEQ_BAND_NUM];
    HI_U16 u16Frequency[VQE_PEQ_BAND_NUM];
    HI_U16 u16Q[VQE_PEQ_BAND_NUM];
} AI_PEQ_CONFIG_S;
```

[Member]

Member	Description
bUsrMode	Mode select 0: automatic mode 1: user mode



Member	Description
u32BandNum	Number of adjusted frequency bands Value range: (0, 10]
u8FilterType	Type of filter, including high-pass (HP) filter, low-pass (LP) filter, high-pass shelf (HS) filter, low-pass shelf (LS) filter, and peak (PK) filter. <ul style="list-style-type: none">• 0: HP• 1: LS• 2: PK• 3: HS• 4: LP
s8GaindB	Gain of the PEQ frequency band (dB). The gains of the HP filter and LP filter are 0 dB. The value of other types of filter ranges from -15 to +15.
u16Frequency	Center frequency of the PEQ frequency band (Hz). The value range is [20, 4000] for the HP/LS filter, [20, 22000] for the PK filter, and [4000, 22000] for the HS/LP filter.
u16Q	Value Q. The value range is [7, 10] for the HS/LS filter, [5, 100] for the PK filter, and 7 for the HP/LP filter.

[Note]

- This function supports only the 48 kHz sampling rate.
- The preceding advanced parameters take effect only when the user mode is enabled. If the user mode is disabled, the default parameter configurations corresponding to **enWorkstate** in [AI_HIFIVQE_CONFIG_S](#) take effect.
- During parameter configuration, the correctness of advanced parameters is checked only when the user mode is enabled. The advanced parameters can be successfully configured only when they are correct.

[See Also]

None

AI_VQE_CONFIG_S

[Description]

Defines the configuration information about AI VQE.

[Syntax]

```
typedef struct hiAI_VQE_CONFIG_S
{
    HI_S32          bHpfOpen;
    HI_S32          bAecOpen;
    HI_S32          bAnrOpen;
```



```
HI_S32          bRnrOpen;
HI_S32          bAgcOpen;
HI_S32          bEqOpen;
HI_S32          bHdrOpen;
HI_S32          s32WorkSampleRate;
HI_S32          s32FrameSample;
VQE_WORKSTATE_E enWorkstate;
AUDIO_HPF_CONFIG_S stHpfCfg;
AI_AEC_CONFIG_S stAecCfg;
AUDIO_ANR_CONFIG_S stAnrCfg;
AI_RNR_CONFIG_S stRnrCfg;
AUDIO_AGC_CONFIG_S stAgcCfg;
AUDIO_EQ_CONFIG_S stEqCfg;
AI_HDR_CONFIG_S stHdrCfg;
} AI_VQE_CONFIG_S;
```

[Member]

Member	Description
bHpfOpen	HPF enable
bAecOpen	AEC enable
bAnrOpen	ANR enable
bRnrOpen	RNR enable
bAgcOpen	AGC enable
bEqOpen	EQ enable
bHdrOpen	HDR enable
s32WorkSampleRate	Working sampling rate of the internal function algorithm Value: 8 kHz, 16 kHz, or 48 kHz Default value: 8 kHz RNR/HPF/HDR supports 8 kHz, 16 kHz, and 48 kHz. Others support 8 kHz and 16 kHz.
s32FrameSample	VQE frame length (number of sampling points) Value range: [80, 4096]
enWorkstate	Working mode
stHpfCfg	HPF configuration information
stAecCfg	AEC configuration information
stAnrCfg	ANR configuration information
stRnrCfg	RNR configuration information
stAgcCfg	AGC configuration information



Member	Description
stEqCfg	EQ configuration information
stHdrCfg	HDR configuration information

[Note]

Table 9-8 describes the default parameter configurations in various scenario modes of VQE.

Table 9-8 Default parameter configurations in various scenario modes of VQE

Module	Parameter	Scenario Mode		
		Common	Music	Noisy
HPF	enHpfFreq	AUDIO_HPF_FREQ_120	AUDIO_HPF_FREQ_120	AUDIO_HPF_FREQ_120
AEC	s8CngMode	1	1	1
ANR	s16NrIntensity	15	8	15
	s16NoiseDbThr	45	60	45
	s8SpProSwitch	1	1	0
RNR	s32NrMode	1	0	1
	s32MaxNrLevel	15	15	15
	s32NoiseThresh	50	50	50
AGC	s8TargetLevel	-2	-2	-2
	s8NoiseFloor	-40	-40	-40
	s8MaxGain	15	10	15
	s8AdjustSpeed	10	5	10
	s8ImproveSNR	2	0	2
	s8UseHighPassFilt	0	0	0
	s8OutputMode	0	0	0
	s16NoiseSupSwitch	1	0	1

- Hi3518E V200 does not support the RNR, HDR function.
- Hi3519 V100 does not support this data structure.
- HDR can be configured to the automatic mode only when there is a built-in audio codec.

[See Also]

None



AI_HIFIVQE_CONFIG_S

[Description]

Defines the configuration information structure of AI VQE (Hi-Fi).

[Syntax]

```
typedef struct hiAI_HIFIVQE_CONFIG_S
{
    HI_U32          u32OpenMask;
    HI_S32          s32WorkSampleRate;
    HI_S32          s32FrameSample;
    VQE_WORKSTATE_E enWorkstate;
    AI_HDR_CONFIG_S stRnrCfg;
    AI_HDR_CONFIG_S stHdrCfg;
    VQE_DRC_SECNUM vqeDrcSecNum;
```

[Description]

Defines the number of configurable levels for the DRC dynamic curve.

[Syntax]

```
#define VQE_DRC_SECNUM 5
```

[Note]

None

[See Also]

[AI_DRC_CONFIG_S](#)

```
AI_DRC_CONFIG_S     stDrcCfg;
AI_PEQ_CONFIG_S     stPeqCfg;
} AI_HIFIVQE_CONFIG_S;
```

[Member]

Member	Description
u32OpenMask	Mask value for enabling the functions of Hi-Fi VQE, u32OpenMask cannot be 0 .
s32WorkSampleRate	Working sampling rate of the internal functional algorithm HPF/RNR supports 8 kHz, 16 kHz, and 48 kHz, while PEQ/DRC supports only 48 kHz.
s32FrameSample	VQE frame length (number of sampling points) Value range: [80, 4096]
enWorkstate	Working mode
stHpfCfg	Configuration information about high-pass filtering



Member	Description
stRnrCfg	RNR configuration information
stHdrCfg	HDR configuration information
stDrcCfg	DRC configuration information
stPeqCfg	PEQ configuration information

[Note]

Table 9-9 describes the default parameter configurations in various scenario modes of Hi-Fi VQE.

Table 9-9 Default parameter configurations in various scenario modes of Hi-Fi VQE

Module	Parameter	Scenario Mode		
		Common	Music	Noisy
HPF	enHpfFreq	AUDIO_HPF_FREQ_120	AUDIO_HPF_FREQ_120	AUDIO_HPF_FREQ_120
RNR	s32NrMode	1	0	1
	s32MaxNrLevel	15	15	15
	s32NoiseThresh	50	50	50
DRC	s16AttackTime	24	24	24
	s16ReleaseTime	100	100	100
	s16OldLevDb	{0, -472, -792, -960, -1280}	{0, -472, -792, -960, -1280}	{0, -472, -792, -960, -1280}
	s16NewLevDb	{0, -174, -410, -608, -1021}	{0, -174, -410, -608, -1021}	{0, -174, -410, -608, -1021}

- Hi3518E V200 does not support Hi-Fi VQE.
- In automatic mode, the PEQ function does not take effect (**bUsrMode** is false).
- When there is a built-in audio codec, HDR can be configured to the automatic mode.

[See Also]

None

AI_TALKVQE_CONFIG_S

[Description]

Defines the configuration information structure of AI VQE (Talk).

[Syntax]



```
typedef struct hiAI_TALKVQE_CONFIG_S
{
    HI_U32          u32OpenMask;
    HI_S32          s32WorkSampleRate;
    HI_S32          s32FrameSample;
    VQE_WORKSTATE_E enWorkstate;
    AUDIO_HPF_CONFIG_S stHpfCfg;
    AI_AEC_CONFIG_S stAecCfg;
    AUDIO_ANR_CONFIG_S stAnrCfg;
    AUDIO_AGC_CONFIG_S stAgcCfg;
    AUDIO_EQ_CONFIG_S stEqCfg;
    AI_HDR_CONFIG_S stHdrCfg;
} AI_TALKVQE_CONFIG_S
```

[Member]

Member	Description
u32OpenMask	Mask value for enabling the functions of Talk VQE, u32OpenMask cannot be 0.
s32WorkSampleRate	Working sampling rate of the internal functional algorithm Value: 8 kHz or 16 kHz Default value: 8 kHz
s32FrameSample	VQE frame length (number of sampling points) Value range: [80, 4096]
enWorkstate	Working mode
stHpfCfg	Configuration information about high-pass filtering
stAecCfg	Configuration information about echo cancellation
stAnrCfg	Configuration information about audio noise reduction
stAgcCfg	Configuration information about automatic gain control
stEqCfg	Configuration information about the equalizer
stHdrCfg	HDR configuration information

[Note]

Table 9-10 describes the default parameter configurations in various scenario modes of Talk VQE.



Table 9-10 Default parameter configurations in various scenario modes of Talk VQE

Module	Parameter	Scenario Mode		
		Common	Music	Noisy
HPF	enHpfFreq	AUDIO_HPF_FREQ_120	AUDIO_HPF_FREQ_120	AUDIO_HPF_FREQ_120
AEC	s8CngMode	1	1	1
ANR	s16NrIntensity	15	8	15
	s16NoiseDbThr	45	60	45
	s8SpProSwitch	1	1	0
AGC	s8TargetLevel	-2	-2	-2
	s8NoiseFloor	-40	-40	-40
	s8MaxGain	15	10	15
	s8AdjustSpeed	10	5	10
	s8ImproveSNR	2	0	2
	s8UseHighPassFilt	0	0	0
	s8OutputMode	0	0	0
	s16NoiseSupSwitch	1	0	1

- Hi3518E V200 and Hi3516A do not support Talk VQE.
- When there is a built-in audio codec, HDR can be configured to the automatic mode.

[See Also]

None

AO_VQE_CONFIG_S

[Description]

Defines the configuration information about AO VQE. The definition of this structure for Hi3518E V200 and the Hi3516A differs slightly from that for Hi3519 V100.

[Syntax]

Definition for Hi3518E V200 and the Hi3516A:

```
typedef struct hiAO_VQE_CONFIG_S
{
    HI_S32          bHpfOpen;
    HI_S32          bAnrOpen;
    HI_S32          bAgcOpen;
    HI_S32          bEqOpen;
    HI_S32          s32WorkSampleRate;
```



```
    HI_S32          s32FrameSample;
    VQE_WORKSTATE_E enWorkstate;
    AUDIO_HPF_CONFIG_S stHpfCfg;
    AUDIO_ANR_CONFIG_S stAnrCfg;
    AUDIO_AGC_CONFIG_S stAgcCfg;
    AUDIO_EQ_CONFIG_S  stEqCfg;
} AO_VQE_CONFIG_S;
```

Definition for Hi3519 V100:

```
typedef struct hiAO_VQE_CONFIG_S
{
    HI_U32          u32OpenMask;
    HI_S32          s32WorkSampleRate;
    HI_S32          s32FrameSample;
    VQE_WORKSTATE_E enWorkstate;
    AUDIO_HPF_CONFIG_S stHpfCfg;
    AUDIO_ANR_CONFIG_S stAnrCfg;
    AUDIO_AGC_CONFIG_S stAgcCfg;
    AUDIO_EQ_CONFIG_S  stEqCfg;
} AO_VQE_CONFIG_S;
```

[Member]

Chip	Member	Description
Hi3518E V200, Hi3516A	bHpfOpen	HPF enable
	bAnrOpen	ANR enable
	bAgcOpen	AGC enable
	bEqOpen	EQ enable
	s32WorkSampleRate	Working sampling rate of the internal function algorithm Value: 8 kHz, 16 kHz, or 48 kHz. Default value: 8 kHz Only the HPF module supports the 48 kHz frequency.
	s32FrameSample	VQE frame length (number of sampling points) Value range: [80, 4096]
	enWorkstate	Working mode
	stHpfCfg	HPF configuration information
	stAnrCfg	ANR configuration information
	stAgcCfg	AGC configuration information



Chip	Member	Description
	stEqCfg	EQ configuration information
Hi3519 V100	u32OpenMask	Mask value for each AO VQE function u32OpenMask cannot be 0 .
	s32WorkSampleRate	Working sampling rate of the internal function algorithm Value: 8 kHz, 16 kHz, or 48 kHz Default value: 8 kHz (Only HPF supports the 48 kHz working sampling rate.)
	s32FrameSample	VQE frame length (number of sampling points) Value range: [80, 4096]
	enWorkstate	Working mode
	stHpfCfg	HPF configuration information
	stAnrCfg	ANR configuration information
	stAgcCfg	AGC configuration information
	stEqCfg	EQ configuration information

[Note]

Table 9-11 describes the default parameter configurations in various scenario modes of DnVQE.

Table 9-11 Default parameter configurations in various scenario modes of DnVQE

Module	Parameter	Scenario Mode		
		Common	Music	Noisy
HPF	enHpfFreq	AUDIO_HPF_FREQ_120	AUDIO_HPF_FREQ_120	AUDIO_HPF_FREQ_120
ANR	s16NrIntensity	15	8	15
	s16NoiseDbThr	45	60	45
	s8SpProSwitch	1	1	0
AGC	s8TargetLevel	-2	-2	-2
	s8NoiseFloor	-40	-40	-40
	s8MaxGain	15	10	15
	s8AdjustSpeed	10	5	10
	s8ImproveSNR	2	0	2



Module	Parameter	Scenario Mode		
		Common	Music	Noisy
	s8UseHighPassFilt	0	0	0
	s8OutputMode	0	0	0
	s16NoiseSupSwitch	1	0	1

[See Also]

None

AUDIO_STREAM_S

[Description]

Defines the structure of an audio stream.

[Syntax]

```
typedef struct hiAUDIO_STREAM_S
{
    HI_U8 *pStream;           /*Stream virtual address*/
    HI_U32 u32PhyAddr;       /*Stream physical address*/
    HI_U32 u32Len;           /*Stream length (in byte)*/
    HI_U64 u64TimeStamp;     /*Frame timestamp*/
    HI_U32 u32Seq;           /*Frame sequence. If the stream is not a valid
frame, u32Seq is 0.*/
} AUDIO_STREAM_S;
```

[Member]

Member	Description
pStream	Pointer to an audio stream
u32PhyAddr	Physical address for an audio stream
u32Len	Length (in byte) of an audio stream
u64TimeStamp	Timestamp of an audio stream
u32Seq	Sequence number of an audio stream

[Note]

None

[See Also]

[HI_MPI_AENC_GetStream](#)



AO_CHN_STATE_S

[Description]

Defines the status of the data buffer on the AO channel.

[Syntax]

```
typedef struct hiAO_CHN_STATE_S
{
    HI_U32          u32ChnTotalNum;
    HI_U32          u32ChnFreeNum;
    HI_U32          u32ChnBusyNum;
} AO_CHN_STATE_S;
```

[Member]

Member	Description
u32ChnTotalNum	Number of buffers on an AO channel
u32ChnFreeNum	Number of available idle buffers
u32ChnBusyNum	Number of used buffers

[Note]

None

[See Also]

[HI_MPI_AO_QueryChnStat](#)

AUDIO_TRACK_MODE_E

[Description]

Defines the track mode of an audio device.

[Syntax]

```
typedef enum hiAUDIO_TRACK_MODE_E
{
    AUDIO_TRACK_NORMAL      = 0,
    AUDIO_TRACK_BOTH_LEFT   = 1,
    AUDIO_TRACK_BOTH_RIGHT  = 2,
    AUDIO_TRACK_EXCHANGE    = 3,
    AUDIO_TRACK_MIX         = 4,
    AUDIO_TRACK_LEFT_MUTE   = 5,
    AUDIO_TRACK_RIGHT_MUTE  = 6,
    AUDIO_TRACK_BOTH_MUTE   = 7,
    AUDIO_TRACK_BUTT
} AUDIO_TRACK_MODE_E;
```



[Member]

Member	Description
AUDIO_TRACK_NORMAL	Audio is not processed (normal mode).
AUDIO_TRACK_BOTH_LEFT	The audio in two channels is audio-left channel audio.
AUDIO_TRACK_BOTH_RIGHT	The audio in two channels is audio-right channel audio.
AUDIO_TRACK_EXCHANGE	The audio in audio-left channel and audio-right channel is exchanged. To be specific, the audio in the audio-left channel is changed to audio-right channel audio, and the audio in the audio-right channel is changed to audio-left channel audio.
AUDIO_TRACK_MIX	The audio in two channels is mixed and then output.
AUDIO_TRACK_LEFT_MUTE	The audio-left channel is muted, and the audio-right channel plays its original audio.
AUDIO_TRACK_RIGHT_MUTE	The audio-right channel is muted, and the audio-left channel plays its original audio.
AUDIO_TRACK_BOTH_MUTE	Both the audio-left channel and audio-right channel are muted.

[Note]

None

[See Also]

- [HI_MPI_AI_SetTrackMode](#)
- [HI_MPI_AO_SetTrackMode](#)

AUDIO_FADE_RATE_E

[Description]

Defines the fade-in/fade-out rate of an AO device.

[Syntax]

```
typedef enum hiAUDIO_FADE_RATE_E
{
    AUDIO_FADE_RATE_1    = 0,
    AUDIO_FADE_RATE_2    = 1,
    AUDIO_FADE_RATE_4    = 2,
    AUDIO_FADE_RATE_8    = 3,
    AUDIO_FADE_RATE_16   = 4,
    AUDIO_FADE_RATE_32   = 5,
    AUDIO_FADE_RATE_64   = 6,
    AUDIO_FADE_RATE_128  = 7,
```



```
AUDIO_FADE_RATE_BUTT  
} AUDIO_FADE_RATE_E;
```

[Member]

Member	Description
AUDIO_FADE_RATE_1	The fade-in/fade-out rate changes once every one sampling point.
AUDIO_FADE_RATE_2	The fade-in/fade-out rate changes once every two sampling points.
AUDIO_FADE_RATE_4	The fade-in/fade-out rate changes once every four sampling points.
AUDIO_FADE_RATE_8	The fade-in/fade-out rate changes once every eight sampling points.
AUDIO_FADE_RATE_16	The fade-in/fade-out rate changes once every 16 sampling points.
AUDIO_FADE_RATE_32	The fade-in/fade-out rate changes once every 32 sampling points.
AUDIO_FADE_RATE_64	The fade-in/fade-out rate changes once every 64 sampling points.
AUDIO_FADE_RATE_128	The fade-in/fade-out rate changes once every 128 sampling points.

[Note]

None

[See Also]

None

AUDIO_FADE_S

[Description]

Defines the fade-in/fade-out settings of an AO device.

[Syntax]

```
typedef struct hiAUDIO_FADE_S  
{  
    HI_BOOL          bFade;  
    AUDIO_FADE_RATE_E enFadeInRate;  
    AUDIO_FADE_RATE_E enFadeOutRate;  
} AUDIO_FADE_S;
```

[Member]



Member	Description
bFade	Fade-in/Fade-out enable HI_TRUE: enabled HI_FALSE: disabled
enFadeInRate	Volume fade-in rate of an AO device
enFadeOutRate	Volume fade-out rate of an AO device

[Note]

None

[See Also]

[HI_MPI_AO_SetMute](#)

G726_BPS_E

[Description]

Defines the bit rate for the G.726 encoding/decoding protocol.

[Syntax]

```
typedef enum hiG726_BPS_E
{
    G726_16K = 0,
    G726_24K,
    G726_32K,
    G726_40K,
    MEDIA_G726_16K,      /* G726 16kbit/s for ASF */
    MEDIA_G726_24K,      /* G726 24kbit/s for ASF */
    MEDIA_G726_32K,      /* G726 32kbit/s for ASF */
    MEDIA_G726_40K,      /* G726 40kbit/s for ASF */
    G726_BUTT,
} G726_BPS_E;
```

[Member]

Member	Description
G726_16K	16 kbit/s G.726. See 4.5.4 G72616 in the <i>RFC3551</i> .
G726_24K	24 kbit/s G.726. See 4.5.4 G72624 in the <i>RFC3551</i> .
G726_32K	32 kbit/s G.726. See 4.5.4 G72632 in the <i>RFC3551</i> .
G726_40K	40 kbit/s G.726. See 4.5.4 G72640 in the <i>RFC3551</i> .
MEDIA_G726_16K	G.726 16 kbit/s for ASF



Member	Description
MEDIA_G726_24K	G.726 24 kbit/s for ASF
MEDIA_G726_32K	G.726 32 kbit/s for ASF
MEDIA_G726_40K	G.726 40 kbit/s for ASF

[Note]

None

[See Also]

None

ADPCM_TYPE_E

[Description]

Defines the type of the ADPCM encoding/decoding protocol.

[Syntax]

```
typedef enum hiADPCM_TYPE_E
{
    ADPCM_TYPE_DVI4 = 0,
    ADPCM_TYPE_IMA,
    ADPCM_TYPE_ORG_DVI4,
    ADPCM_TYPE_BUTT,
} ADPCM_TYPE_E;
```

[Member]

Member	Description
ADPCM_TYPE_DVI4	32 kbit/s ADPCM (DVI4)
ADPCM_TYPE_IMA	32 kbit/s ADPCM (IMA)
ADPCM_TYPE_ORG_DVI4	32 kbit/s ADPCM (ORG_DVI4)

[Note]

None

[See Also]

None

AUDIO_SAVE_FILE_INFO_S

[Description]



Defines the configuration information about the function of saving audio files.

[Syntax]

```
typedef struct hiAUDIO_SAVE_FILE_INFO_S
{
    HI_BOOL bCfg;
    HI_CHAR    aFilePath[MAX_AUDIO_FILE_PATH_LEN];
    HI_CHAR    aFileName[MAX_AUDIO_FILE_NAME_LEN];
    HI_U32     u32FileSize; /*in KB*/
} AUDIO_SAVE_FILE_INFO_S;
```

[Member]

Member	Description
bCfg	Configuration enable
aFilePath[MAX_AUDIO_FILE_PATH_LEN]	Path for saving the audio file
aFileName[MAX_AUDIO_FILE_NAME_LEN]	Name for saving the audio file
u32FileSize	File size. The value range is [1, 10240], and the unit is KB.

[Note]

None

[See Also]

None

AUDIO_FILE_STATUS_S

[Description]

Defines the audio file saving status structure.

[Syntax]

```
typedef struct hiAUDIO_FILE_STATUS_S
{
    HI_BOOL    bSaving;
} AUDIO_FILE_STATUS_S;
```

[Member]

Member	Description
bSaving	Whether the channel is in the state of saving files <ul style="list-style-type: none">• HI_TRUE: yes• HI_FALSE: no



[Note]

None

[See Also]

None

9.4.2 AENC

The AENC data structures are defined as follows:

- [AENC_MAX_CHN_NUM](#): Defines the maximum number of AENC channels.
- [AENC_ATTR_G711_S](#): Defines the attribute structure of the G.711 encoding protocol.
- [AENC_ATTR_G726_S](#): Defines the attribute structure of the G.726 encoding protocol.
- [AENC_ATTR_ADPCM_S](#): Defines the attribute structure of the ADPCM encoding protocol.
- [AENC_ATTR_LPCM_S](#): Defines the attribute structure of the LPCM encoding protocol.
- [AENC_CHN_ATTR_S](#): Defines the attribute structure of an AENC channel.
- [AENC_ENCODER_S](#): Defines encoder attributes.

AENC_MAX_CHN_NUM

[Description]

Defines the maximum number of AENC channels.

[Syntax]

```
#define AENC_MAX_CHN_NUM      32
```

[Note]

None

[See Also]

None

AENC_ATTR_G711_S

[Description]

Defines the attribute structure of the G.711 encoding protocol.

[Syntax]

```
typedef struct hiAENC_ATTR_G711_S
{
    HI_U32 resv;
} AENC_ATTR_G711_S;
```

[Member]



Member	Description
resv	To be extended (it is not used currently)

[Note]

None

[See Also]

None

AENC_ATTR_G726_S

[Description]

Defines the attribute structure of the G.726 encoding protocol.

[Syntax]

```
typedef struct hiAENC_ATTR_G726_S
{
    G726_BPS_E enG726bps;
} AENC_ATTR_G726_S;
```

[Member]

Member	Description
enG726bps	Bit rate of the G.726 protocol

[Note]

None

[See Also]

[G726_BPS_E](#)

AENC_ATTR_ADPCM_S

[Description]

Defines the attribute structure of the ADPCM encoding protocol.

[Syntax]

```
typedef struct hiAENC_ATTR_ADPCM_S
{
    ADPCM_TYPE_E enADPCMTYPE;
} AENC_ATTR_ADPCM_S;
```

[Member]



Member	Description
enADPCMType	ADPCM type

[Note]

None

[See Also]

[ADPCM_TYPE_E](#)

AENC_ATTR_LPCM_S

[Description]

Defines the attribute structure of the LPCM encoding protocol.

[Syntax]

```
typedef struct hiAENC_ATTR_LPCM_S
{
    HI_U32 resv;           /* reserve item */
} AENC_ATTR_LPCM_S;
```

[Member]

Member	Description
resv	To be extended (it is not used currently)

[Note]

None

[See Also]

None

AENC_CHN_ATTR_S

[Description]

Defines the attribute structure of an AENC channel.

[Syntax]

```
typedef struct hiAENC_CHN_ATTR_S
{
    PAYLOAD_TYPE_E enType;
    HI_U32 u32PtNumPerFrm;
    HI_U32 u32BufSize; /* buffer size. Unit: frame [2-
    MAX_AUDIO_FRAME_NUM] */
```



```
    HI_VOID *pValue;  
} AENC_CHN_ATTR_S;
```

[Member]

Member	Description
enType	Type of an AENC protocol Static attribute
u32PtNumPerFrm	Required frame length for the AENC protocol (if the received audio frame length is less than or equal to the required frame length, the received frame can be encoded.)
u32BufSize	Size of an AENC buffer, in frame Value range: [2, MAX_AUDIO_FRAME_NUM] Static attribute
pValue	Pointer to the attribute of a specific protocol Static attribute

[Note]

None

[See Also]

None

AENC_ENCODER_S

[Description]

Defines encoder attributes.

[Syntax]

```
typedef struct hiAENC_ENCODER_S  
{  
    PAYLOAD_TYPE_E enType;  
    HI_U32     u32MaxFrmLen;  
    HI_CHAR    aszName[16];  
    HI_S32     (*pfnOpenEncoder) (HI_VOID *pEncoderAttr, HI_VOID  
        **ppEncoder);  
    HI_S32     (*pfnEncodeFrm) (HI_VOID *pEncoder, const AUDIO_FRAME_S  
        *pstData, HI_U8 *pu8Outbuf, HI_U32 *pu32OutLen);  
    HI_S32     (*pfnCloseEncoder) (HI_VOID *pEncoder);  
} AENC_ENCODER_S;
```

[Member]



Member	Description
enType	Type of the encoding protocol
u32MaxFrmLen	Maximum stream length
aszName	Encoder name
pfnOpenEncoder	Pointer to the function for starting an encoder
pfnEncodeFrm	Pointer to encoding functions
pfnCloseEncoder	Pointer to the function for closing an encoder

[Note]

See the *Audio Components API Reference*.

[See Also]

[HI_MPI_AENC_RegeisterEncoder](#)

9.4.3 ADEC

The ADEC data structures are defined as follows:

- [MAX_AUDIO_FRAME_NUM](#): Defines the maximum number of buffered ADEC frames.
- [ADEC_MAX_CHN_NUM](#): Defines the maximum number of ADEC channels.
- [ADEC_ATTR_G711_S](#): Defines the attribute structure of the G.711 decoding protocol.
- [ADEC_ATTR_G726_S](#): Defines the attribute structure of the G.726 decoding protocol.
- [ADEC_ATTR_ADPCM_S](#): Defines the attribute structure of the ADPCM decoding protocol.
- [ADEC_ATTR_LPCM_S](#): Defines the attribute structure of the LPCM decoding protocol.
- [ADEC_MODE_E](#): Defines a decoding mode.
- [ADEC_CHN_ATTR_S](#): Defines the attribute structure of an ADEC channel.
- [ADEC_DECODER_S](#): Defines decoder attributes.
- [AUDIO_FRAME_INFO_S](#): Defines the information about the decoded audio frame.

MAX_AUDIO_FRAME_NUM

[Description]

Defines the maximum number of buffered ADEC frames.

[Syntax]

```
#define MAX_AUDIO_FRAME_NUM      300
```

[Note]

None

[See Also]



None

ADEC_MAX_CHN_NUM

[Description]

Defines the maximum number of ADEC channels.

[Syntax]

```
#define ADEC_MAX_CHN_NUM      32
```

[Note]

None

[See Also]

None

ADEC_ATTR_G711_S

[Description]

Defines the attribute structure of the G.711 decoding protocol.

[Syntax]

```
typedef struct hiADEC_ATTR_G711_S
{
    HI_U32    resv;
}ADEC_ATTR_G711_S;
```

[Member]

Member	Description
resv	To be extended (it is not used currently)

[Note]

None

[See Also]

None

ADEC_ATTR_G726_S

[Description]

Defines the attribute structure of the G.726 decoding protocol.

[Syntax]

```
typedef struct hiADEC_ATTR_G726_S
{
```



```
G726_BPS_E enG726bps;  
}ADEC_ATTR_G726_S;
```

[Member]

Member	Description
enG726bps	Bit rate of the G.726 protocol

[Note]

None

[See Also]

[G726_BPS_E](#)

ADEC_ATTR_ADPCM_S

[Description]

Defines the attribute structure of the ADPCM decoding protocol.

[Syntax]

```
typedef struct hiADEC_ATTR_ADPCM_S  
{  
    ADPCM_TYPE_E     enADPCMType;  
}ADEC_ATTR_ADPCM_S;
```

[Member]

Member	Description
enADPCMType	ADPCM type

[Note]

None

[See Also]

[ADPCM_TYPE_E](#)

ADEC_ATTR_LPCM_S

[Description]

Defines the attribute structure of the LPCM decoding protocol.

[Syntax]

```
typedef struct hiADEC_ATTR_LPCM_S  
{
```



```
    HI_U32 resv;  
}ADEC_ATTR_LPCM_S;
```

[Member]

Member	Description
resv	To be extended (it is not used currently)

[Note]

None

[See Also]

None

ADEC_MODE_E

[Description]

Defines a decoding mode.

[Syntax]

```
typedef enum hiADEC_MODE_E  
{  
    ADEC_MODE_PACK = 0, /*require input is valid dec pack(a  
                        complete frame encode result),  
                        e.g.the stream get from AENC is a  
                        valid dec pack, the stream know actually  
                        pack len from file is also a dec pack.  
                        this mode is a high-performance mode*/  
    ADEC_MODE_STREAM ,/*input is stream, low-performance,  
                        if you cannot find out whether a stream is  
                        valid dec pack, you could use  
                        this mode*/  
    ADEC_MODE_BUTT  
}ADEC_MODE_E;
```

[Member]

Member	Description
ADEC_MODE_PACK	Pack mode
ADEC_MODE_STREAM	Stream mode

[Note]



- The Pack mode is used when users determine that the current stream is exactly one frame. The decoder directly decodes the stream. If the stream is not one frame, an error occurs in the decoder. This mode enables high efficiency. If a stream encoded by an AENC module is not corrupted, this mode can be used to decode the stream.
- The Stream mode is used when users cannot determine whether the current stream is one frame. The decoder must identify and buffer the stream. Due to low efficiency, this mode is used to read streams in a file to be decoded or streams whose boundaries are uncertain. The pack mode is recommended because the length of the voice encoding stream is fixed; therefore, the boundary of a frame is easily determined.

[See Also]

None

ADEC_CHN_ATTR_S

[Description]

Defines the attribute structure of an ADEC channel.

[Syntax]

```
typedef struct hiADEC_CH_ATTR_S
{
    PAYLOAD_TYPE_E enType;
    HI_U32         u32BufSize;
    ADEC_MODE_E    enMode;
    HI_VOID        *pValue;
}ADEC_CHN_ATTR_S;
```

[Member]

Member	Description
enType	Type of an ADEC protocol Static attribute
u32BufSize	ADEC buffer size, in frame Value range: [2, MAX_AUDIO_FRAME_NUM] Static attribute
enMode	Decoding mode Static attribute
pValue	Pointer to the attribute of a specific protocol

[Note]

None

[See Also]

None



ADEC_DECODER_S

[Description]

Defines decoder attributes.

[Syntax]

```
typedef struct hiADEC_DECODER_S
{
    PAYLOAD_TYPE_E enType;
    HI_CHAR         aszName[16];
    HI_S32          (*pfnOpenDecoder)(HI_VOID *pDecoderAttr, HI_VOID
**ppDecoder);
    HI_S32          (*pfnDecodeFrm)(HI_VOID *pDecoder, HI_U8
**pu8Inbuf, HI_S32 *ps32LeftByte, HI_U16 *pu16Outbuf, HI_U32
*pu32OutLen, HI_U32 *pu32Chns);
    HI_S32          (*pfnGetFrmInfo)(HI_VOID *pDecoder, HI_VOID *pInfo);
    HI_S32          (*pfnCloseDecoder)(HI_VOID *pDecoder);
    HI_S32          (*pfnResetDecoder)(HI_VOID *pDecoder);
} ADEC_DECODER_S;
```

[Member]

Member	Description
enType	Type of the decoding protocol
aszName	Decoder name
pfnOpenDecoder	Pointer to the function for starting a decoder
pfnDecodeFrm	Pointer to decoding functions
pfnGetFrmInfo	Pointer to the function for obtaining audio frame information
pfnCloseDecoder	Pointer to the function for closing a decoder
pfnResetDecoder	Member used for clearing the buffer and reset the decoder

[Note]

See the *Audio Components API Reference*.

[See Also]

[HI_MPI_ADEC_RegeisterDecoder](#)

AUDIO_FRAME_INFO_S

[Description]

Defines the information about the decoded audio frame.

[Syntax]



```
typedef struct hiAUDIO_FRAME_INFO_S
{
    AUDIO_FRAME_S *pstFrame;
    HI_U32         u32Id;
} AUDIO_FRAME_INFO_S;
```

[Member]

Member	Description
pstFrame	Pointer to the audio frame
u32Id	Index of the audio frame Value range: [0, 49]

[Note]

See the *Audio Components API Reference*.

[See Also]

- [HI_MPI_ADEC_GetFrame](#)
- [HI_MPI_ADEC_ReleaseFrame](#)

9.4.4 Embedded audio Codec

The following are the data structures related to the embedded audio codec:

- [ACODEC_FS_E](#): Defines the sampling rate of the I²S interface.
- [ACODEC_MIXER_E](#): Defines the input mode of the audio codec.
- [ACODEC_VOL_CTRL](#): Defines the volume control structure of the audio codec.

ACODEC_FS_E

[Description]

Defines the sampling rate of the I²S interface.

[Syntax]

```
typedef enum hiACODEC_FS_E {
    ACODEC_FS_48000 = 0x1a,
    ACODEC_FS_24000 = 0x19,
    ACODEC_FS_12000 = 0x18,
    ACODEC_FS_44100 = 0x1a,
    ACODEC_FS_22050 = 0x19,
    ACODEC_FS_11025 = 0x18,
    ACODEC_FS_32000 = 0x1a,
    ACODEC_FS_16000 = 0x19,
    ACODEC_FS_8000 = 0x18,
    ACODEC_FS_64000 = 0x1b,
```



```
ACODEC_FS_96000 = 0x1b,  
ACODEC_FS_BUTT = 0x1c,  
} ACODEC_FS_E;
```

[Member]

Member	Description
ACODEC_FS_48000	48 kHz
ACODEC_FS_24000	24 kHz
ACODEC_FS_12000	12 kHz
ACODEC_FS_44100	44.1 kHz
ACODEC_FS_22050	22.05 kHz
ACODEC_FS_11025	11.025 kHz
ACODEC_FS_32000	32 kHz
ACODEC_FS_16000	16 kHz
ACODEC_FS_8000	8 kHz
ACODEC_FS_64000	64 kHz
ACODEC_FS_96000	96 kHz

[Note]

None

[See Also]

None

ACODEC_MIXER_E

[Description]

Defines the input mode of the embedded audio codec.

[Syntax]

For Hi3516A:

```
typedef enum hiACODEC_MIXER_E {  
    /*select MICIN or LINEIN*/  
    ACODEC_MIXER_LINEIN = 0x0,  
    ACODEC_MIXER_MICIN = 0x1,  
    ACODEC_MIXER_BUTT,  
} ACODEC_MIXER_E;
```

For Hi3518E V200:



```
typedef enum hiACODEC_MIXER_E {  
    /*select IN or IN_D*/  
    ACODEC_MIXER_IN      = 0x3,  
    ACODEC_MIXER_IN_D    = 0x4,  
    ACODEC_MIXER_BUTT,  
} ACODEC_MIXER_E;
```

For Hi3519 V100:

```
typedef enum hiACODEC_MIXER_E {  
  
    ACODEC_MIXER_IN0     = 0x0,  
    ACODEC_MIXER_IN1     = 0x1,  
    ACODEC_MIXER_IN_D    = 0x2,  
    ACODEC_MIXER_BUTT,  
} ACODEC_MIXER_E
```

For the Hi3516A:

Member	Description
ACODEC_MIXER_LINEIN	LINEIN input for the MICPGA
ACODEC_MIXER_MICIN	MICIN input for the MICPGA

For Hi3518E V200:

Member	Description
ACODEC_MIXER_IN	Single-ended input for the MICPGA
ACODEC_MIXER_IN_D	Differential input for the MICPGA

For Hi3519 V100:

Member	Description
ACODEC_MIXER_IN0	IN0 single-ended input for the MICPGA
ACODEC_MIXER_IN1	IN1 single-ended input for the MICPGA
ACODEC_MIXER_IN_D	IN_D differential input for the MPCPGA

[Note]

Currently the audio embedded codec input interfaces of Hi3519 V100 are classified into two groups; group 0 and group 1 (each group contains one pair of audio-left and audio-right channels). When the single-ended input is used, either of the two groups can be selected;



when the differential input is used, each of the two groups inputs one differential signal (as the audio-left channel and audio-right channel respectively).

[See Also]

None

ACODEC_VOL_CTRL

[Description]

Defines the volume control structure of the embedded audio codec.

[Syntax]

```
typedef struct {
    /*volume control, 0x00~0x7e, 0x7F:mute*/
    unsigned int vol_ctrl;
    /*adc/dac mute control, 1:mute, 0:unmute*/
    unsigned int vol_ctrl_mute;
} ACODEC_VOL_CTRL;
```

[Member]

Member	Description
vol_ctrl	Volume
vol_ctrl_mute	Mute control

[Note]

None

[See Also]

None

9.5 Error Codes

AI Error Codes

[Table 9-12](#) describes the error codes for the AI MPIS.

Table 9-12 Error codes for the AI MPIS

Error Code	Macro Definition	Description
0xA0158001	HI_ERR_AI_INVALID_DEVID	The AI device ID is invalid.
0xA0158002	HI_ERR_AI_INVALID_CHNID	The AI channel ID is invalid.



Error Code	Macro Definition	Description
0xA0158003	HI_ERR_AI_ILLEGAL_PARAM	The settings of the AI parameters are invalid.
0xA0158005	HI_ERR_AI_NOT_ENABLED	The AI device or AI channel is not enabled.
0xA0158006	HI_ERR_AI_NULL_PTR	The input parameter pointer is null.
0xA0158007	HI_ERR_AI_NOT_CONFIG	The attributes of an AI device are not set.
0xA0158008	HI_ERR_AI_NOT_SUPPORT	The operation is not supported.
0xA0158009	HI_ERR_AI_NOT_PERM	The operation is forbidden.
0xA015800C	HI_ERR_AI_NOMEM	The memory fails to be allocated.
0xA015800D	HI_ERR_AI_NOBUF	The AI buffer is insufficient.
0xA015800E	HI_ERR_AI_BUF_EMPTY	The AI buffer is empty.
0xA015800F	HI_ERR_AI_BUF_FULL	The AI buffer is full.
0xA0158010	HI_ERR_AI_SYS_NOTREADY	The AI system is not initialized.
0xA0158012	HI_ERR_AI_BUSY	The AI system is busy.
0xA0158041	HI_ERR_AI_VQE_ERR	A VQE processing error occurs in the AI channel.

AO Error Codes

Table 9-13 describes the error codes for the AO MPIs.

Table 9-13 Error codes for the AO MPIs

Error Code	Macro Definition	Description
0xA0168001	HI_ERR_AO_INVALID_DEVID	The AO device ID is invalid.
0xA0168002	HI_ERR_AO_INVALID_CHNID	The AO channel ID is invalid.
0xA0168003	HI_ERR_AO_ILLEGAL_PARAM	The settings of the AO parameters are invalid.
0xA0168005	HI_ERR_AO_NOT_ENABLED	The AO device or AO channel is not enabled.
0xA0168006	HI_ERR_AO_NULL_PTR	The output parameter pointer is null.
0xA0168007	HI_ERR_AO_NOT_CONFIG	The attributes of an AO device are not set.



Error Code	Macro Definition	Description
0xA0168008	HI_ERR_AO_NOT_SUPPORT	The operation is not supported.
0xA0168009	HI_ERR_AO_NOT_PERM	The operation is forbidden.
0xA016800C	HI_ERR_AO_NOMEM	The system memory is insufficient.
0xA016800D	HI_ERR_AO_NOBUF	The AO buffer is insufficient.
0xA016800E	HI_ERR_AO_BUF_EMPTY	The AO buffer is empty.
0xA016800F	HI_ERR_AO_BUF_FULL	The AO buffer is full.
0xA0168010	HI_ERR_AO_SYS_NOTREADY	The AO system is not initialized.
0xA0168012	HI_ERR_AO_BUSY	The AO system is busy.
0xA0168041	HI_ERR_AO_VQE_ERR	A VQE processing error occurs in the AO channel.

AENC Error Codes

Table 9-14 describes the error codes for the AENC MPIS.

Table 9-14 Error codes for the AENC MPIS

Error Code	Macro Definition	Description
0xA0178001	HI_ERR_AENC_INVALID_DEVID	The AENC device ID is invalid.
0xA0178002	HI_ERR_AENC_INVALID_CHNID	The AENC channel ID is invalid.
0xA0178003	HI_ERR_AENC_ILLEGAL_PARAM	The settings of the AENC parameters are invalid.
0xA0178004	HI_ERR_AENC_EXIST	An AENC channel is created.
0xA0178005	HI_ERR_AENC_UNEXIST	An AENC channel is not created.
0xA0178006	HI_ERR_AENC_NULL_PTR	The input parameter pointer is null.
0xA0178007	HI_ERR_AENC_NOT_CONFIG	The AENC channel is not configured.
0xA0178008	HI_ERR_AENC_NOT_SUPPORT	The operation is not supported.
0xA0178009	HI_ERR_AENC_NOT_PERM	The operation is forbidden.



Error Code	Macro Definition	Description
0xA017800C	HI_ERR_AENC_NOMEM	The system memory is insufficient.
0xA017800D	HI_ERR_AENC_NOBUF	The buffer for AENC channels fails to be allocated.
0xA017800E	HI_ERR_AENC_BUF_EMPTY	The AENC channel buffer is empty.
0xA017800F	HI_ERR_AENC_BUF_FULL	The AENC channel buffer is full.
0xA0178010	HI_ERR_AENC_SYS_NOTREADY	The system is not initialized.
0xA0178040	HI_ERR_AENC_ENCODER_ERR	An AENC data error occurs.
0xA0178041	HI_ERR_AENC_VQE_ERR	A VQE processing error occurs in the AENC channel.

ADEC Error Codes

[Table 9-15](#) describes the error codes for the ADEC MPIs.

Table 9-15 Error codes for the ADEC MPIs

Error Code	Macro Definition	Description
0xA0188001	HI_ERR_ADEC_INVALID_DEVID	The ADEC device is invalid.
0xA0188002	HI_ERR_ADEC_INVALID_CHNID	The ADEC channel ID is invalid.
0xA0188003	HI_ERR_ADEC_ILLEGAL_PARAM	The settings of the ADEC parameters are invalid.
0xA0188004	HI_ERR_ADEC_EXIST	An ADEC channel is created.
0xA0188005	HI_ERR_ADEC_UNEXIST	An ADEC channel is not created.
0xA0188006	HI_ERR_ADEC_NULL_PTR	The input parameter pointer is null.
0xA0188007	HI_ERR_ADEC_NOT_CONFIG	The attributes of an ADEC channel are not set.
0xA0188008	HI_ERR_ADEC_NOT_SUPPORT	The operation is not supported.
0xA0188009	HI_ERR_ADEC_NOT_PERM	The operation is forbidden.
0xA018800C	HI_ERR_ADEC_NOMEM	The system memory is insufficient.



Error Code	Macro Definition	Description
0xA018800D	HI_ERR_ADEC_NOBUF	The buffer for ADEC channels fails to be allocated.
0xA018800E	HI_ERR_ADEC_BUF_EMPTY	The ADEC channel buffer is empty.
0xA018800F	HI_ERR_ADEC_BUF_FULL	The ADEC channel buffer is full.
0xA0188010	HI_ERR_ADEC_SYS_NOTREADY	The system is not initialized.
0xA0188040	HI_ERR_ADEC_DECODER_ERR	An ADEC data error occurs.
0xA0188041	HI_ERR_ADEC_BUF_LACK	An insufficient buffer occurs in the ADEC channel.



Contents

10 VGS.....	10-1
10.1 Overview	10-1
10.2 Function Description	10-1
10.2.1 Basic Concepts.....	10-1
10.2.2 Functions	10-2
10.3 API Reference	10-4
10.4 Data Structures	10-14
10.5 Error Codes	10-21



Tables

Table 10-1 Hi3516A VGS specifications.....	10-3
Table 10-2 Hi3518E V200 VGS specifications	10-3
Table 10-3 Hi3519 V100 VGS specifications.....	10-4
Table 10-4 Error codes of VGS MPIs.....	10-21



10 VGS

10.1 Overview

The video graphics subsystem (VGS) processes an input picture by using the functions such as scaling, format conversion, decompression, and Cover/OSD overlay.

10.2 Function Description

10.2.1 Basic Concepts

- Job
A job has multiple tasks. Tasks are performed in the sequence of adding tasks to a job, and all tasks in a job are submitted to the VGS at a time. You can specify the maximum number of jobs supported by the VGS by setting the **max_vgs_job** module parameter when loading the VGS .ko driver. The value range of the job is [20, 400], and the default value is 128.
- Task
Tasks are the operations performed on a picture, such as copying, scaling, and format conversion. You can specify the maximum number of tasks supported by the VGS by setting the **max_vgs_task** module parameter when loading the VGS .ko driver. The value range of the task is [20, 800], and the default value is 200.
- Node
The VGS hardware performs operations by node. Typically, a task corresponds to a node. Some tasks need to be split into multiple nodes (when the LDC task or rotation task is being processed). The nodes contain the information required by hardware such as the source address, destination address, and operation type. The information is organized based on hardware requirements. You can specify the maximum number of nodes supported by the VGS by setting the **max_vgs_node** module parameter when loading the VGS .ko driver. The value range of the node is [20, 800], and the default value is 200.
- Handle
Each job handle identifies a job.



10.2.2 Functions

The VGS supports scaling, OSD overlay, Cover overlay, line-drawing, format conversion, compression, and decompression.

- Scaling

The VGS can scale a picture. At most 16x zoom-in and 15x zoom-out are supported.

The VGS of the Hi3516A/Hi3518E V200/Hi3519 V100 supports single component (Y) scaling.

- OSD overlay

The VGS can overlay OSD on a picture.

Note that in the VGS, a task can at most overlay one OSD. When more OSDs need to be overlaid, multiple tasks need to be submitted. Each task takes the picture overlaid with OSD from the previous task as the source picture. The VGS offline OSD overlay operation supports only uncompressed picture inputs and outputs, or compressed picture inputs and outputs. During VGS offline OSD overlaying, the same memory is used for the input and output.

- Cover overlay

The VGS can overlay a Cover region on a picture.

Note that in the VGS, a task can at most overlay one Cover region. When more Cover regions need to be overlaid, multiple tasks need to be submitted. Each task takes the picture overlaid with the Cover region from the previous task as the source picture. The VGS offline Cover overlay supports only uncompressed picture inputs and outputs, or compressed picture inputs and outputs. During VGS offline Cover overlay, the same memory is used for input and output.

- Line-drawing

The VGS can draw lines on a picture.

Note that when the VGS draws lines, the line width ranges from 2 to 8 and must be 2-pixel-aligned.

During the line-drawing, if the horizontal coordinate of the start coordinate and that of the end coordinate are the same, the vertical coordinates remain unchanged and the horizontal coordinates extend to the right to widen the line.

During the line-drawing, if the horizontal coordinate of the start coordinate and that of the end coordinate are different, the horizontal coordinates remain unchanged and the vertical coordinates extend downward (the horizontal coordinates remain unchanged, and the vertical coordinates move downward to widen the line).

- Format conversion

The input and output pictures can be semi-planar420, semi-planar422, or single-component (Y) pictures. Conversion between semi-planar420 and semi-planar422 is supported. Besides conversion between semi-planar420 and semi-planar422, Hi3518E V200/Hi3519 V100 also supports conversion from semi-planar420/semi-planar422 to the single-component (Y) format. The Hi3516A/Hi3518E V200/Hi3519 V100 supports only the linear picture input and output.

- Compression

The input and output pictures must be linear pictures that are compressed or not compressed.

- Decompression

The input and output pictures must be linear pictures that are compressed or not compressed. Because most back-end modules (such as the VO, IVE, and VDA) support



only non-compressed linear pictures, the format of front-module(such as the VI and VPSS) pictures must be converted and compressed pictures must be decompressed before pictures are sent to the back-end modules. If you obtain pictures for debugging, the pictures are applicable only after you convert the picture format or decompress pictures by using the VGS.

- Low-power strategy

The VGS module adopts the low-power strategy, and the VGS clock is disabled when the VGS is not working. In this case, manually reading or writing VGS registers may result in read/write errors or suspension.

Table 10-1 Hi3516A VGS specifications

Item	Specifications
Data format of the video input	<ul style="list-style-type: none">• Semi-planar420, semi-planar422, or single-component (Y) picture format• Linear storage format• Compressed or non-compressed
Input picture resolution	<ul style="list-style-type: none">• Minimum resolution of 32 x 32• Maximum resolution of 2592 x 2592• 2-pixel-aligned width and height, 16-byte-aligned stride
Data format of the video output	<ul style="list-style-type: none">• Semi-planar420, semi-planar422, or single-component (Y) picture format• Linear storage format• Compressed or non-compressed
Output picture resolution	<ul style="list-style-type: none">• Minimum resolution of 32 x 32• Maximum resolution of 4096 x 4096• 2-pixel-aligned width and height, 16-byte-aligned stride
Scaling capability	<ul style="list-style-type: none">• 16x horizontal or vertical zoom-in• 15x horizontal or vertical zoom-out

Table 10-2 Hi3518E V200 VGS specifications

Item	Specifications
Data format of the video input	<ul style="list-style-type: none">• Semi-planar420, semi-planar422, or single-component (Y) picture format• Linear storage format• Compressed or non-compressed
Input picture resolution	<ul style="list-style-type: none">• Minimum resolution of 32 x 32• Maximum resolution of 2048 x 2048• 2-pixel-aligned width and height, 16-byte-aligned stride



Item	Specifications
Data format of the video output	<ul style="list-style-type: none">• Semi-planar420, semi-planar422, or single-component (Y) picture format• Linear storage format• Compressed or non-compressed
Output picture resolution	<ul style="list-style-type: none">• Minimum resolution of 32 x 32• Maximum resolution of 2048 x 2048• 2-pixel-aligned width and height, 16-byte-aligned stride
Scaling capability	<ul style="list-style-type: none">• 16x horizontal or vertical zoom-in• 15x horizontal or vertical zoom-out

Table 10-3 Hi3519 V100 VGS specifications

Item	Specifications
Data format of the video input	<ul style="list-style-type: none">• Semi-planar420, semi-planar422, or single-component (Y) picture format• Linear storage format• Compressed or uncompressed pictures
Input picture resolution	<ul style="list-style-type: none">• Minimum resolution of 32 x 32• Maximum resolution of 8192 x 4096 for the uncompressed picture and 4096 x 4096 for the compressed picture• 2-pixel-aligned width and height, 16-pixel-aligned stride
Data format of the video output	<ul style="list-style-type: none">• Semi-planar420, semi-planar422, or single-component (Y) picture format• Linear storage format• Compressed or uncompressed pictures
Output picture resolution	<ul style="list-style-type: none">• Minimum resolution of 30 x 16• Maximum resolution of 8192 x 4096 for the uncompressed picture and 4096 x 4096 for the compressed picture• 2-pixel-aligned width and height, 16-pixel-aligned stride
Scaling capability	<ul style="list-style-type: none">• 16x horizontal or vertical zoom-in• 15x horizontal or vertical zoom-out

10.3 API Reference

The VGS provides the following MPIS:

- [HI_MPI_VGS_BeginJob](#): Starts a job.



- [HI_MPI_VGS_AddScaleTask](#): Adds a scaling task to a started job.
- [HI_MPI_VGS_AddDrawLineTask](#): Adds a line-drawing task to a started job.
- [HI_MPI_VGS_AddDrawLineTaskArray](#): Adds a line-drawing batch task to a started job.
- [HI_MPI_VGS_AddCoverTask](#): Adds a Cover overlay task to a started job.
- [HI_MPI_VGS_AddCoverTaskArray](#): Adds a Cover overlay batch task to a started job.
- [HI_MPI_VGS_AddOsdTask](#): Adds an OSD overlay task to a started job.
- [HI_MPI_VGS_AddOsdTaskArray](#): Adds an OSD overlay batch task to a started job.
- [HI_MPI_VGS_EndJob](#): Submits a job (the VGS starts to process the job).
- [HI_MPI_VGS_CancelJob](#): Cancels a job.

HI_MPI_VGS_BeginJob

[Description]

Starts a job.

[Syntax]

```
HI_S32 HI_MPI_VGS_BeginJob(VGS_HANDLE *phHandle);
```

[Parameter]

Parameter	Description	Input/Output
phHandle	Returned handle	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 10.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vgs.h, mpi_vgs.h
- Library file: libmpi.a

[Note]

- Multiple jobs can be started at a time. However, the handle returned by **phHandle** can be used only when the code indicating success is returned after [HI_MPI_VGS_BeginJob](#) is called.
- The **phHandle** pointer cannot be null or invalid.

HI_MPI_VGS_AddScaleTask

[Description]



Adds a scaling task to a started job.

[Syntax]

```
HI_S32 HI_MPI_VGS_AddScaleTask(VGS_HANDLE hHandle, VGS_TASK_ATTR_S
*pstTask);
```

[Parameter]

Parameter	Description	Input/Output
hHandle	Handle of a started job	Input
pstTask	Pointer to VGS task attributes	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 10.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vgs.h, mpi_vgs.h
- Library file: libmpi.a

[Note]

- The job identified by **HHandle** must be started.
- For Hi3516A, the task attributes must meet the requirements in [0](#).
- If an error code indicating failure is returned after `HI_MPI_VGS_AddScaleTask` is called and no tasks need to be added, you can submit added tasks by calling `HI_MPI_VGS_EndJob`. If an error code indicating failure is returned after `HI_MPI_VGS_AddScaleTask` is called and there are tasks to be added, you must cancel the job identified by **hHandle** by calling `HI_MPI_VGS_CancelJob`. Otherwise, the job identified by **hHandle** cannot be reused.

[Example]

See the example of [HI_MPI_VGS_CancelJob](#).

HI_MPI_VGS_AddDrawLineTask

[Description]

Adds a line-drawing task to a started job.

[Syntax]

```
HI_S32 HI_MPI_VGS_AddDrawLineTask(VGS_HANDLE hHandle, VGS_TASK_ATTR_S
*pstTask, VGS_DRAW_LINE_S *pstVgsDrawLine);
```



[Parameter]

Parameter	Description	Input/Output
hHandle	Handle of a started job	Input
pstTask	Pointer to VGS task attributes	Input
pstVgsDrawLine	Configuration pointer to VGS line-drawing attributes	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 10.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vgs.h, mpi_vgs.h
- Library file: libmpi.a

[Note]

- The job identified by **hHandle** must be started.
- Task attributes must meet the requirement.
- If an error code indicating failure is returned after [HI_MPI_VGS_AddDrawLineTask](#) is called and no task needs to be added, you can submit added tasks by calling [HI_MPI_VGS_EndJob](#). If an error code indicating failure is returned after [HI_MPI_VGS_AddDrawLineTask](#) is called and there are tasks to be added, you must cancel the job identified by **hHandle** by calling [HI_MPI_VGS_CancelJob](#). Otherwise, the job identified by **hHandle** cannot be reused.

[Example]

See the example of [HI_MPI_VGS_CancelJob](#).

HI_MPI_VGS_AddDrawLineTaskArray

[Description]

Adds a line-drawing batch task to a started job.

[Syntax]

```
HI_S32 HI_MPI_VGS_AddDrawLineTaskArray(VGS_HANDLE hHandle,  
VGS_TASK_ATTR_S* pstTask, VGS_DRAW_LINE_S astVgsDrawLine[], HI_U32  
u32ArraySize);
```

[Parameter]



Parameter	Description	Input/Output
hHandle	Handle of a started job	Input
pstTask	Pointer to VGS task attributes	Input
astVgsDrawLine	Configuration structure array for VGS line-drawing attributes	Input
u32ArraySize	Number of configuration structures for line-drawing attributes in the configuration structure array for VGS line-drawing attributes, that is, number of lines that can be drawn at a time Value range: [1, 100]	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 10.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vgs.h, mpi_vgs.h
- Library file: libmpi.a

[Note]

- The job identified by **hHandle** must be started.
- The task attributes must meet the VGS capability.
- If an error code indicating failure is returned after [HI_MPI_VGS_AddDrawLineTaskArray](#) is called and no task needs to be added, you can submit added tasks by calling [HI_MPI_VGS_EndJob](#). If an error code indicating failure is returned after [HI_MPI_VGS_AddDrawLineTaskArray](#) is called and there are tasks to be added, you must cancel the job identified by **hHandle** by calling [HI_MPI_VGS_CancelJob](#). Otherwise, the job identified by **hHandle** cannot be reused.
- The value of **u32ArraySize** must be the same as the number of drawn lines.

[Example]

See the example of [HI_MPI_VGS_CancelJob](#).

HI_MPI_VGS_AddCoverTask

[Description]

Adds a Cover overlay task to a started job.

[Syntax]

```
HI_S32 HI_MPI_VGS_AddCoverTask(VGS\_HANDLE hHandle, VGS\_TASK\_ATTR\_S
```



```
*pstTask, VGS\_ADD\_COVER\_S *pstVgsAddCover);
```

[Parameter]

Parameter	Description	Input/Output
hHandle	Handle of a started job	Input
pstTask	Pointer to VGS task attributes	Input
pstVgsAddCover	Configuration pointer to VGS Cover overlay attributes	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 10.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vgs.h, mpi_vgs.h
- Library file: libmpi.a

[Note]

- The job identified by **hHandle** must be started.
- Task attributes must meet the requirement.
- If an error code indicating failure is returned after [HI_MPI_VGS_AddCoverTask](#) is called and no task needs to be added, you can submit added tasks by calling [HI_MPI_VGS_EndJob](#). If an error code indicating failure is returned after [HI_MPI_VGS_AddCoverTask](#) is called and there are tasks to be added, you must cancel the job identified by **hHandle** by calling [HI_MPI_VGS_CancelJob](#). Otherwise, the job identified by **hHandle** cannot be reused.

[Example]

See the example of [HI_MPI_VGS_CancelJob](#).

HI_MPI_VGS_AddCoverTaskArray

[Description]

Adds a Cover overlay batch task to a started job.

[Syntax]

```
HI_S32 HI_MPI_VGS_AddCoverTaskArray(VGS\_HANDLE hHandle, VGS\_TASK\_ATTR\_S *  
pstTask, VGS\_ADD\_COVER\_S astVgsAddCover[], HI_U32 u32ArraySize);
```

[Parameter]



Parameter	Description	Input/Output
hHandle	Handle of a started job	Input
pstTask	Pointer to VGS task attributes	Input
astVgsAddCover	Configuration structure array for VGS Cover overlay attributes	Input
u32ArraySize	Number of configuration structures for Cover overlay attributes in the configuration structure array for VGS Cover overlay attributes, that is, number of Covers that can be overlaid at a time Value range: [1, 100]	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 10.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vgs.h, mpi_vgs.h
- Library file: libmpi.a

[Note]

- The job identified by **hHandle** must be started.
- The task attributes must meet the VGS capability.
- If an error code indicating failure is returned after [HI_MPI_VGS_AddCoverTaskArray](#) is called and no task needs to be added, you can submit added tasks by calling [HI_MPI_VGS_EndJob](#). If an error code indicating failure is returned after [HI_MPI_VGS_AddCoverTaskArray](#) is called and there are tasks to be added, you must cancel the job identified by **hHandle** by calling [HI_MPI_VGS_CancelJob](#). Otherwise, the job identified by **hHandle** cannot be reused.
- The value of **u32ArraySize** must be the same as the number of overlaid Covers.

[Example]

See the example of [HI_MPI_VGS_CancelJob](#).

HI_MPI_VGS_AddOsdTask

[Description]

Adds an OSD overlay task to a started job.

[Syntax]

```
HI_S32 HI_MPI_VGS_AddOsdTask(VGS_HANDLE hHandle, VGS_TASK_ATTR_S *pstTask,
```



```
VGS_ADD OSD S *pstVgsAddOsd);
```

[Parameter]

Parameter	Description	Input/Output
hHandle	Handle of a started job	Input
pstTask	Pointer to VGS task attributes	Input
pstVgsAddOsd	Configuration pointer to VGS OSD overlay attributes	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 10.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vgs.h, mpi_vgs.h
- Library file: libmpi.a

[Note]

- The job identified by **hHandle** must be started.
- Task attributes must meet the requirement.
- If an error code indicating failure is returned after **HI_MPI_VGS_AddOsdTask** is called and no tasks need to be added, you can submit added tasks by calling **HI_MPI_VGS_EndJob**. If an error code indicating failure is returned after **HI_MPI_VGS_AddOsdTask** is called and there are tasks to be added, you must cancel the job identified by **hHandle** by calling **HI_MPI_VGS_CancelJob**. Otherwise, the job identified by **hHandle** cannot be reused.

[Example]

See the example of [HI_MPI_VGS_CancelJob](#).

HI_MPI_VGS_AddOsdTaskArray

[Description]

Adds an OSD overlay batch task to a started job.

[Syntax]

```
HI_S32 HI_MPI_VGS_AddOsdTaskArray(VGS_HANDLE hHandle, VGS_TASK_ATTR_S *  
pstTask, VGS_ADD OSD S astVgsAddOsd[], HI_U32 u32ArraySize);
```

[Parameter]



Parameter	Description	Input/Output
hHandle	Handle of a started job	Input
pstTask	Pointer to VGS task attributes	Input
astVgsAddOsd	Configuration structure array for VGS OSD overlay attributes	Input
u32ArraySize	Number of configuration structures for OSD overlay attributes in the configuration structure array for VGS OSD overlay attributes, that is, number of OSDs that can be overlaid at a time Value range: [1, 100]	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 10.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vgs.h, mpi_vgs.h
- Library file: libmpi.a

[Note]

- The job identified by **hHandle** must be started.
- The task attributes must meet the VGS capability.
- If an error code indicating failure is returned after [HI_MPI_VGS_AddOsdTaskArray](#) is called and no task needs to be added, you can submit added tasks by calling [HI_MPI_VGS_EndJob](#). If an error code indicating failure is returned after [HI_MPI_VGS_AddOsdTaskArray](#) is called and there are tasks to be added, you must cancel the job identified by **hHandle** by calling [HI_MPI_VGS_CancelJob](#). Otherwise, the job identified by **hHandle** cannot be reused.
- The value of **u32ArraySize** must be the same as the number of overlaid OSDs.

[Example]

See the example of [HI_MPI_VGS_CancelJob](#).

HI_MPI_VGS_EndJob

[Description]

Submits a job.

[Syntax]

```
HI_S32 HI_MPI_VGS_EndJob(VGS\_HANDLE hHandle);
```



[Parameter]

Parameter	Description	Input/Output
hHandle	Handle of a started job	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. The return value is an error code. For details, see section 10.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vgs.h, mpi_vgs.h
- Library file: libmpi.a

[Note]

- The job identified by **HHandle** must be started.
- If an error code indicating failure is returned after HI_MPI_VGS_EndJob is called, you must cancel the job identified by **hHandle** by calling [HI_MPI_VGS_CancelJob](#). Otherwise, the job identified by **hHandle** cannot be reused.

[Example]

See the example of [HI_MPI_VGS_BeginJob](#).

HI_MPI_VGS_CancelJob

[Description]

Cancels a job.

[Syntax]

```
HI_S32 HI_MPI_VGS_CancelJob(VGS_HANDLE hHandle);
```

[Parameter]

Parameter	Description	Input/Output
hHandle	Handle of a started job	Input

[Return Value]

Return Value	Description
0	Success



Return Value	Description
Other values	Failure. The return value is an error code. For details, see section 10.5 "Error Codes."

[Requirement]

- Header files: hi_comm_vgs.h, mpi_vgs.h
- Library file: libmpi.a

[Note]

The job identified by **HHandle** must be started.

[Example]

```
HI_S32 s32Ret = HI_SUCCESS;
VGS_HANDLE hHandle;
VGS_TASK_ATTR_S stTask;
...
...
s32Ret = HI_MPI_VGS_BeginJob(&hHandle);
if(s32Ret != HI_SUCCESS)
{
    ...
    goto HandleErr;
}
...
...
s32Ret = HI_MPI_VGS_AddScaleTask(hHandle, &stTask);
if(s32Ret != HI_SUCCESS)
{
    HI_MPI_VGS_CancelJob(hHandle);
    goto HandleErr;
}

s32Ret = HI_MPI_VGS_EndJob (hHandle);
if(s32Ret != HI_SUCCESS)
{
    HI_MPI_VGS_CancelJob(hHandle);
    goto HandleErr;
}
```

10.4 Data Structures

The VGS provides the following data structures:



- [VGS_HANDLE](#): Defines the VGS job handle.
- [VGS_TASK_ATTR_S](#): Defines VGS task attributes.
- [VGS_DRAW_LINE_S](#): Defines the configuration of the VGS line-drawing.
- [VGS_COVER_TYPE_E](#): Defines the data structure of the VGS Cover overlay.
- [VGS_QUADRANGLE_COVER_S](#): Defines the configuration of the quadrilateral used to cover a region in the VGS.
- [VGS_ADD_COVER_S](#): Defines the configuration of the VGS Cover overlay.
- [VGS_ADD OSD_S](#): Defines the configuration of the VGS OSD overlay.

VGS_HANDLE

[Description]

Defines the VGS job handle.

[Syntax]

```
typedef HI_S32      VGS_HANDLE;
```

[Note]

None

[See Also]

None

VGS_TASK_ATTR_S

[Description]

Defines VGS task attributes.

[Syntax]

```
typedef struct hiVGS_TASK_ATTR_S
{
    VIDEO_FRAME_INFO_S     stImgIn;          /*Input picture */
    VIDEO_FRAME_INFO_S     stImgOut;         /*Output picture */
    HI_U32                 au32privateData[4]; /*Task private data */
    HI_U32                 reserved;        /*Reserved*/
}VGS_TASK_ATTR_S;
```

[Member]

Member	Description
stImgIn	Input picture attribute
stImgOut	Output picture attribute
au32privateData	Task-related private data. The VGS does not use and modify the data.
reserved	Reserved



[Note]

The input and output picture attributes must meet the requirements.

[See Also]

VIDEO_FRAME_INFO_S

VGS_DRAW_LINE_S

[Description]

Defines the configuration of the VGS line-drawing.

[Syntax]

```
typedef struct hivGS_DRAW_LINE_S
{
    POINT_S          stStartPoint; /* line start point */
    POINT_S          stEndPoint;  /* line end point */
    HI_U32           u32Thick;    /* width of line */
    HI_U32           u32Color;   /* color of line */
}VGS_DRAW_LINE_S;
```

[Member]

Member	Description
stStartPoint	Coordinate of the start point of the line (2-pixel-aligned). The value ranges are as follows: Hi3516A: Horizontal position (X): [0, 2592] Vertical position (Y): [0, 2592] Hi3518E V200/Hi3519 V100: Horizontal position (X): [-8190, +8190] Vertical position (Y): [-8190, +8190]
stEndPoint	Coordinate of the end point of the line (2-pixel-aligned). The value ranges are as follows: Hi3516A: Horizontal position (X): [0, 2592] Vertical position (Y): [0, 2592] Hi3518EV200/Hi3519V100: Horizontal position (X): [-8190, +8190] Vertical position (Y): [-8190, +8190]
u32Thick	Line width (2-pixel-aligned) Value range: [2, 8]
u32Color	Line color, RGB888 format



[Note]

- During the line-drawing, if the horizontal coordinate of the start coordinate and that of the end coordinate are the same, the vertical coordinates remain unchanged and the horizontal coordinates extend to the right to widen the line. If the horizontal coordinate of the start coordinate and that of the end coordinate are different, the horizontal coordinates remain unchanged and the vertical coordinates extend downward to widen the line.
- For the Hi3516A, the drawn line cannot exceed the target picture; for Hi3518E V200/Hi3159 V100, part or all of the drawn line can exceed the target picture.
- The end point cannot overlap with the start point.

[See Also]

None

VGS_COVER_TYPE_E

[Description]

Defines the data structure of the VGS Cover overlay.

[Syntax]

```
typedef enum hiVGS_COVER_TYPE_E
{
    COVER_RECT = 0,
    COVER_QUAD_RANGLE,
    COVER_BUTT
} VGS_COVER_TYPE_E;
```

[Member]

Member	Description
COVER_RECT	Cover in a rectangle
COVER_QUAD_RANGLE	Cover in any quadrilateral

[Note]

The cover in a rectangle can only be solid, and the cover in any quadrilateral can be solid or dashed.

[See Also]

None

VGS_QUADRANGLE_COVER_S

[Description]

Defines the configuration of the quadrilateral used to cover a region in the VGS.



[Syntax]

```
typedef struct hivgs_quadrangle_cover_s
{
    HI_BOOL bSolid;
    HI_U32 u32Thick;
    POINT_S stPoint[4];
} VGS_QUADRANGLE_COVER_S;
```

[Member]

Member	Description
bSolid	Solid/Dashed Cover selection 0: dashed Cover 1: solid Cover
u32Thick	Dashed Cover: border thickness Solid Cover: invalid
stPoint[4]	Four coordinates of Cover in any quadrilateral

[Note]

- For the dashed Cover, the border thickness refers to the inward expansion length of the border.
- The four vertexes of the quadrilateral cannot overlap with each other, and cannot in the same horizontal or vertical line.
- The four coordinates of Cover in any quadrilateral must all fall within the picture. Otherwise, the Cover overlaying fails unless you overlay a rectangular Cover region that is parallel to the picture. For Hi3518E V200/Hi3519 V100, the four coordinates of a quadrilateral Cover region can be in the outside of the target picture. The Cover region takes effect when the region overlaps with the target picture, and does not take effect when there is no overlapping.
- Only the convex quadrilateral is supported. If the four coordinate points form a concave quadrilateral, a triangle is displayed.

[See Also]

None

VGS_ADD_COVER_S

[Description]

Defines the configuration of the VGS Cover overlay.

[Syntax]

```
typedef struct hivgs_add_cover_s
{
    VGS_COVER_TYPE_E enCoverType;
```



```
union
{
    RECT_S           stDstRect;
    VGS_QUADRANGLE_COVER_S  stQuadRangle;
};

HI_U32          u32Color;      /* color of cover */
}VGS_ADD_COVER_S;
```

[Member]

Member	Description
enCoverType	Cover type
stDstRect	Position, width, and height of the rectangular Cover region (2-pixel-aligned) Value ranges for the Hi3516A: Horizontal position (X): [0, 2592] Vertical position (Y): [0, 2592] Width: [2, 2592] Height: [2, 2592] Value ranges for Hi3518E V200/Hi3519 V100: Horizontal position (X): [-8190, +8190] Vertical position (Y): [-8190, +8190] Width: [2, 8190] Height: [2, 8190]
stQuadRangle	Coordinates and border thickness of the quadrilateral Cover region (2-pixel-aligned) Value ranges for the Hi3516A: Horizontal position (X): [0, 2592] Vertical position (Y): [0, 2592] Thickness of the dashed border: [2, 8] Value ranges for Hi3518E V200/Hi3519 V100: Horizontal position (X): [-8190, +8190] Vertical position (Y): [-8190, +8190] Thickness of the dashed border: [2, 8]
u32Color	Cover color, RGB888 format

[Note]

- In the VGS, a task can at most overlay one Cover region. When more Cover regions need to be overlaid, multiple tasks need to be submitted. Each task takes the picture overlaid with the Cover region from the previous task as the source picture. Because



each task can only overlay one Cover region, there is no priority. The Cover region overlaid later may overlap the previous Cover region.

- For the Hi3516A, only part of the rectangular Cover region can exceed the target picture, and the entire region is forbidden to exceed the target picture. For Hi3518E V200/Hi3519 V100, part or all of the rectangular Cover region can exceed the target picture.

[See Also]

[VGS_QUADRANGLE_COVER_S](#)

VGS_ADD OSD S

[Description]

Defines the configuration of VGS OSD overlay.

[Syntax]

```
typedef struct hiVGS_ADD OSD S
{
    RECT_S           stRect;          /* start point, width and height
of osd */
    HI_U32           u32BgColor;      /* background color of osd */
    PIXEL_FORMAT_E   enPixelFmt;      /* pixel format of osd */
    HI_U32           u32PhyAddr;
    HI_U32           u32Stride;
    HI_U32           u32BgAlpha;
    HI_U32           u32FgAlpha;
}VGS_ADD OSD S;
```

[Member]

Member	Description
stRect	Start coordinates, width, and height of the OSD (2-pixel-aligned) Value ranges for the Hi3516A: Horizontal position (X): [0, 2592] Vertical position (Y): [0, 2592] Width: [2, 2592] Height: [2, 2592] Value ranges for Hi3518E V200/Hi3519 V100: Horizontal position (X): [0, 8190] Vertical position (Y): [0, 8190] Width: [2, 4094] Height: [2, 4094]
u32BgColor	OSD background color, the format of which corresponds to the OSD pixel format



Member	Description
enPixelFmt	OSD pixel format. Currently the ARGB1555, ARGB4444, and ARGB8888 formats are supported.
u32PhyAddr	Physical address of the memory occupied by the OSD
u32Stride	OSD stride
u32BgAlpha	Background alpha value of the OSD when the pixel format is ARGB1555 Value range: [0, 255]
u32FgAlpha	Foreground alpha value of the OSD when the pixel format is ARGB1555 Value range: [0, 255]

[Note]

- In the VGS, a task can at most overlay one OSD. When more OSDs need to be overlaid, multiple tasks need to be submitted. Each task takes the picture overlaid with OSD from the previous task as the source picture. Because each task can only overlay one OSD, there is no priority. The OSD overlaid later may overlap the previous OSD.
- For the Hi3516A, only part of the OSD can exceed the target picture, and the entire OSD is forbidden to exceed the target picture. For Hi3518E V200/Hi3519 V100, part or all of the OSD can exceed the target picture.

[See Also]

None

10.5 Error Codes

Table 10-4 describes the error codes of VGS MPIs.

Table 10-4 Error codes of VGS MPIs

Error Code	Macro Definition	Description
0xA02D800E	HI_ERR_VGS_BUF_EMPTY	The VGS jobs, tasks, or nodes are used up.
0xA02D8003	HI_ERR_VGS_ILLEGAL_PARAM	The VGS parameter value is invalid.
0xA02D8006	HI_ERR_VGS_NULL_PTR	The input parameter pointer is null.
0xA02D8008	HI_ERR_VGS_NOT_SUPPORT	The operation is not supported.
0xA02D8009	HI_ERR_VGS_NOT_PERMITTED	The operation is forbidden.
0xA02D800D	HI_ERR_VGS_NOBUF	The memory fails to be allocated.



Error Code	Macro Definition	Description
0xA02D8010	HI_ERR_VGS_SYS_NOTREADY	The system is not initialized.



Contents

11 Fisheye Subsystem.....	11-1
11.1 Overview	11-1
11.2 Function Description	11-1
11.2.1 Basic Concepts	11-1
11.2.2 Function Description	11-23
11.3 Fisheye Tool Library	11-26
11.3.1 Fisheye Calibration Tool Library	11-26
11.4 API Reference	11-30
11.5 Data Structures	11-36
11.6 Error Codes	11-49



Tables

Table 11-1 Fisheye correction modes	11-4
Table 11-2 Parameter description in wall mount 180° panoramic correction mode	11-5
Table 11-3 Effect demonstration in wall mount 180° panoramic correction mode.....	11-5
Table 11-4 Parameter description in wall mount normal correction mode	11-8
Table 11-5 Effect demonstration in wall mount normal correction mode.....	11-9
Table 11-6 Parameter description in ceiling/floor mount normal correction mode	11-13
Table 11-7 Effect demonstration in ceiling/floor mount normal correction mode.....	11-14
Table 11-8 Parameter description in ceiling/floor mount 360° panoramic correction mode	11-18
Table 11-9 Effect demonstration in ceiling/floor mount 360° panoramic correction mode	11-18
Table 11-10 Hardware specifications of the fisheye subsystem of Hi3519 V100.....	11-25
Table 11-11 Hardware specifications of the fisheye subsystem of Hi3519 V101.....	11-25
Table 11-12 Error codes of the fisheye subsystem.....	11-50



11 Fisheye Subsystem

11.1 Overview

The fisheye subsystem implements fisheye correction (including 360° panoramic, 180° panoramic, and normal correction modes) and lens distortion correction (LDC) on picture frames.



CAUTION

Only Hi3519 V100 and Hi3519 V101 support the fisheye subsystem.

11.2 Function Description

11.2.1 Basic Concepts

- Job

The fisheye subsystem manages tasks. A job contains multiple tasks. Tasks are executed in the sequence that they are added to a job, and all tasks in a job are submitted to hardware for execution once. You can specify the maximum number of jobs supported by the fisheye subsystem by setting the module parameter **max_fisheye_job** when the .ko driver of the fisheye subsystem is loaded. The value range of the number of jobs is [20, 400], and the default value is 128.
- Task

A task indicates one or multiple specific operations that are performed on a picture, such as fisheye correction or scaling. You can specify the maximum number of tasks supported by the fisheye subsystem by setting the module parameter **max_fisheye_task** when the .ko driver of the fisheye subsystem is loaded. The value range of the number of tasks is [20, 800], and the default value is 200.
- Node

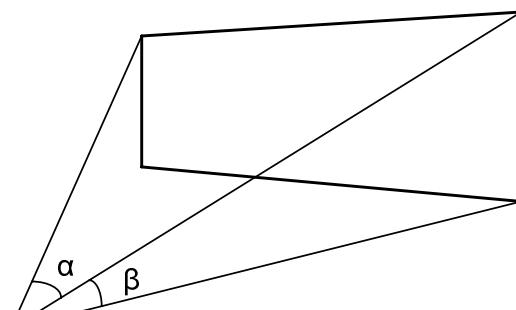
The node is the minimum execution unit of the fisheye hardware. One task corresponds to one node. The node contains the information required for hardware working, such as the source address, destination address, and operation type. The information is organized



based on hardware requirements. You can specify the maximum number of nodes supported by the fisheye subsystem by setting the module parameter **max_fisheye_node** when the .ko driver of the fisheye subsystem is loaded. The value range of the number of nodes is [20, 800], and the default value is 200.

11.2.1.1 Field Angle

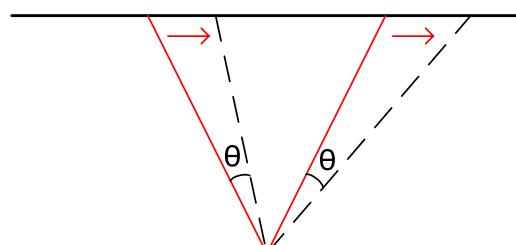
Figure 11-1 Diagram of the horizontal and vertical field angles



Horizontal field angle: α

Vertical field angle: β

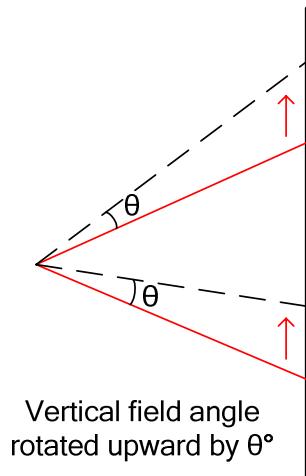
Figure 11-2 Rotation diagram of the horizontal field angle



Horizontal field angle
rotated rightward by θ°



Figure 11-3 Rotation diagram of the vertical field angle



Mount Modes

The fisheye subsystem supports three mount modes: floor mount, ceiling mount, and wall mount. The floor mount mode applies to the look-up scenario where the camera is installed on the desk or floor. The ceiling mount mode applies to the look-down scenario where the camera is installed on the ceiling. The wall mount mode applies to the scenario where the camera is installed on vertical planes such as the wall. You can select the correction mode that is appropriate to the installation scenario to achieve the optimal effect.

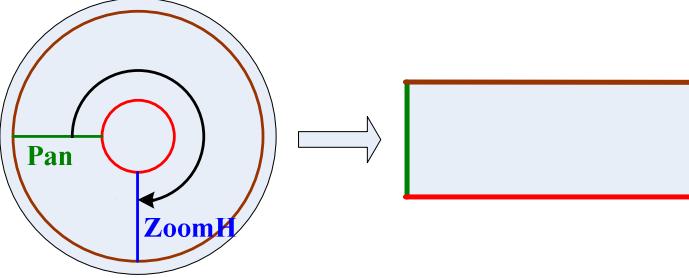
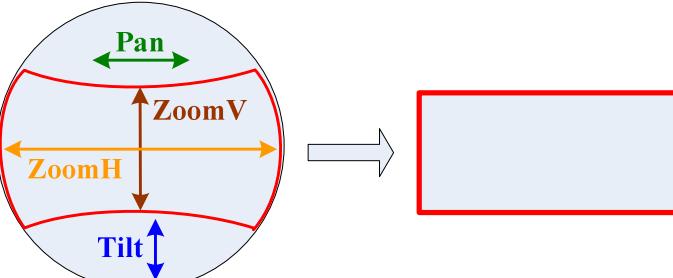
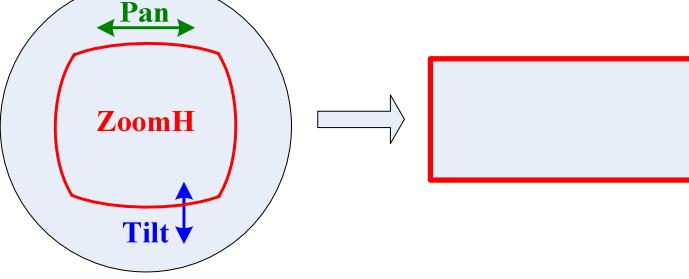
Correction Modes

The fisheye selects the correction region by setting the pan-tilt-zoom (PTZ) parameters in each correction mode to implement the electronic PTZ function. [Table 11-1](#) describes the PTZ parameters and correction models of each correction mode in typical scenarios.

NOTE

The outside radius parameter (**OutRadius**) of the fisheye is used to limit the boundary of the original picture content for the fisheye. Hardware considers the parts that exceed **OutRadius** to be invalid contents (black). **OutRadius** should be set to the actual radius of the original picture for the fisheye. The inside radius **InRadius** is valid only in 360° panoramic correction mode. **InRadius** and **OutRadius** determine the range of the correction region in the radius direction. **OutRadius** must be set to the radius of the original picture regardless of the correction mode. The correction region should be determined by the PTZ parameters.

Table 11-1 Fisheye correction modes

Correction Mode	Typical Scenario	Parameter Description	Correction Model
360° panoramic correction	Ceiling mount and floor mount	<ul style="list-style-type: none">• Pan: start position of the correction region• Tilt: movement of the correction region relative to the original picture in the radius direction• ZoomH: range of the correction region (amplitude)• ZoomV: height of the correction region	
180° panoramic correction	Wall mount	<ul style="list-style-type: none">• Pan: parameter indicating whether the field angle rotates leftward or rightward• Tilt: parameter indicating whether the field angle rotates upward or downward• ZoomH: horizontal field angle• ZoomV: vertical field angle	
Normal correction	Wall mount, ceiling mount, and floor mount	<ul style="list-style-type: none">• Pan: parameter indicating whether the field angle rotates leftward or rightward• Tilt: parameter indicating whether the field angle rotates upward or downward• ZoomH: horizontal and	



Correction Mode	Typical Scenario	Parameter Description	Correction Model
		vertical field angles • ZoomV : invalid parameter	

Application Scenarios

- Wall mount

In wall mount mode, the 180° panoramic correction mode and normal correction mode are recommended. [Table 11-2](#) to [Table 11-5](#) describe the effect demonstration and parameter description.

Table 11-2 Parameter description in wall mount 180° panoramic correction mode

Parameter	Description
Pan	Parameter indicating whether the field angle rotates leftward or rightward. If Pan is greater than 180 , the field angle rotates rightward. If Pan is less than 180 , the field angle rotates leftward. Rotation range: <ul style="list-style-type: none">Hi3519 V100: [-Width/8, Width/8]Hi3519 V101: [-Width/2, Width/2] Note: Width indicates the width of the output picture.
Tilt	Parameter indicating whether the field angle rotates upward or downward. If Tilt is greater than 180 , the field angle rotates upward. If Tilt is less than 180 , the field angle rotates downward. Rotation range: [-30°, +30°]
ZoomH	Horizontal field angle, 4095 at the maximum (180°)
ZoomV	Vertical field angle, 4095 at the maximum (90°)

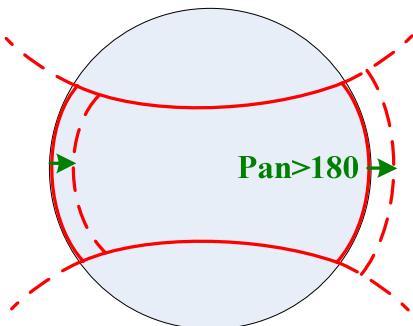
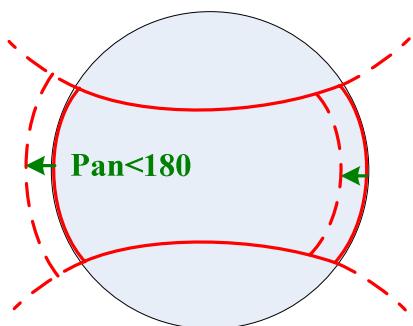
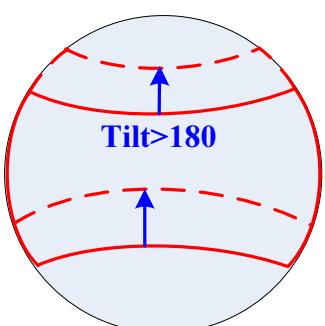
Table 11-3 Effect demonstration in wall mount 180° panoramic correction mode

Typical Parameter Setting	Diagram	Correction Effect Demonstration
Input picture width and height	The content on the left is the typical setting in wall mount 180° panoramic correction mode (the output picture width is equal to the input	
InWidth 3000		
InHeight 3000		
180° correction parameters		



Typical Parameter Setting		Diagram	Correction Effect Demonstration
Mount mode	Wall mount	<p>picture width, and the output picture height is half of the input picture height). The following diagrams are the demonstration diagrams when the value of a specific parameter is changed under the typical setting. The pictures on the right are the corresponding correction effect pictures.</p>	
OutWidth	3000		
OutHeight	1500		
HorOffset	0		
VerOffset	0		
OutRadius	1500		
Pan	180		
Tilt	180		
ZoomH	4095		
ZoomV	4095		
Trapezoid Coef	0	<p>TrapezoidCoef = 32 The trapezoid strength coefficient TrapezoidCoef is valid only during normal correction and 180° correction in wall mount mode. Only Hi3519 V101 supports this parameter.</p>	
FanStrength	0		
FanStrength = 500 The fan strength coefficient FanStrength is valid only in 180° correction mode and is used to adjust lines in the horizontal direction. Only Hi3519 V101 supports this parameter.			



Typical Parameter Setting	Diagram	Correction Effect Demonstration
FanStrength = -500		
Pan = 270		
Pan = 90		
Tilt = 270		



Typical Parameter Setting	Diagram	Correction Effect Demonstration
Tilt = 90		
ZoomH = 2048		
ZoomV = 2048		

Table 11-4 Parameter description in wall mount normal correction mode

Parameter	Description
Pan	Parameter indicating whether the field angle rotates leftward or rightward. If Pan is greater than 180 , the field angle rotates rightward. If Pan is less than 180 , the field angle rotates leftward. Rotation range: [-90°, +90°]



Parameter	Description
Tilt	Parameter indicating whether the field angle rotates upward or downward. If Tilt is greater than 180 , the field angle rotates downward. If Tilt is less than 180 , the field angle rotates upward. Rotation range: [-90°, +90°]
ZoomH	Horizontal and vertical field angles, 4095 at the maximum (126°. The horizontal and vertical field angles are the same in normal mode.)
ZoomV	Invalid parameter

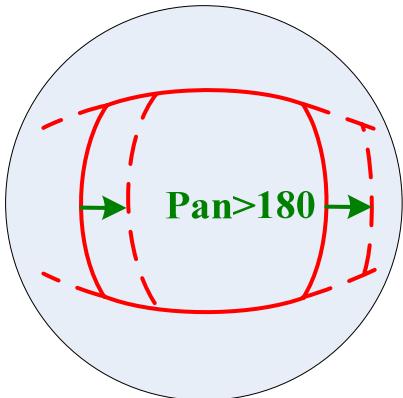
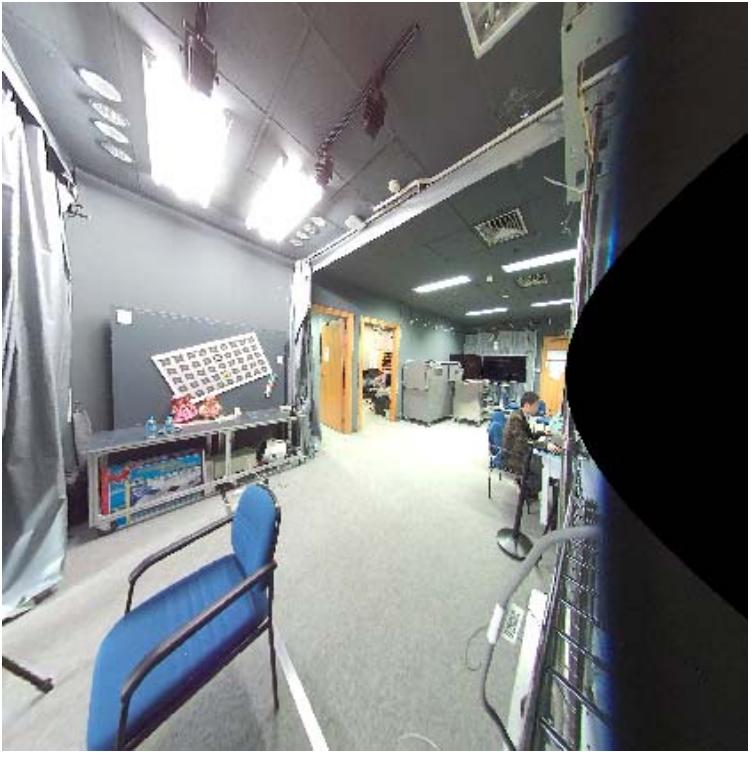
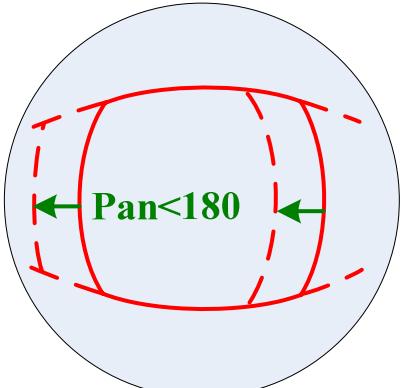
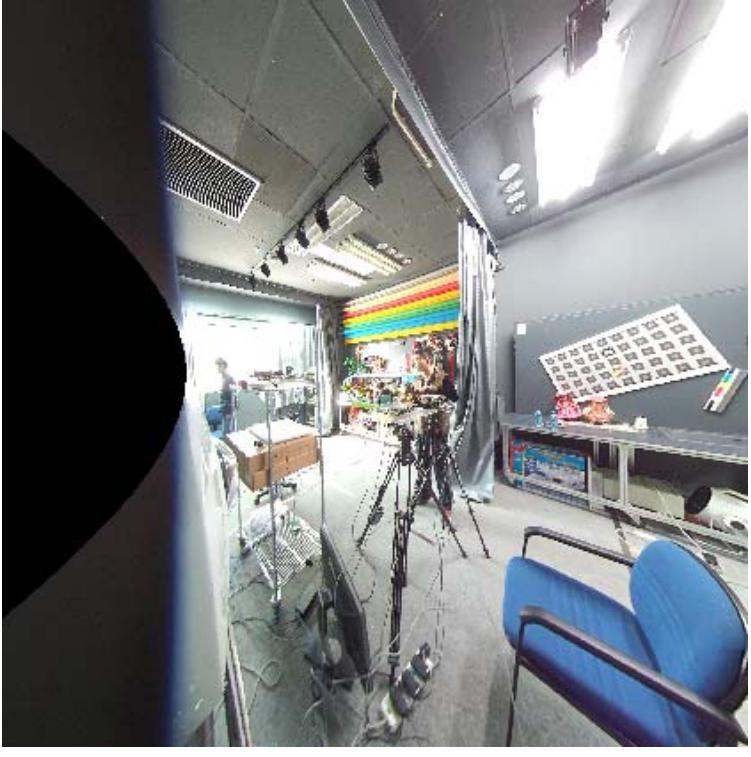
Table 11-5 Effect demonstration in wall mount normal correction mode

Typical Parameter Setting		Diagram	Correction Effect Demonstration
Input picture width and height		The content on the left is the typical setting in wall mount normal correction mode (the output picture width and height are equal to the input picture width and height). The following diagrams are the demonstration diagrams when the value of a specific parameter is changed under the typical setting. The pictures on the right are the corresponding correction effect pictures.	
InWidth	3000		
InHeight	3000		
Normal correction parameters			
Mount mode	Wall mount		
OutWidth	3000		
OutHeight	3000		
HorOffset	0		
VerOffset	0		
OutRadius	1500		
Pan	180		
Tilt	180		
ZoomH	4095		
Trapezoid Coef	0		

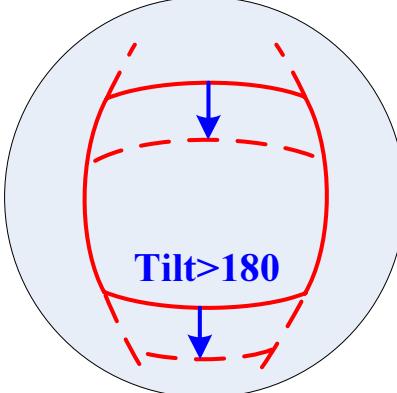
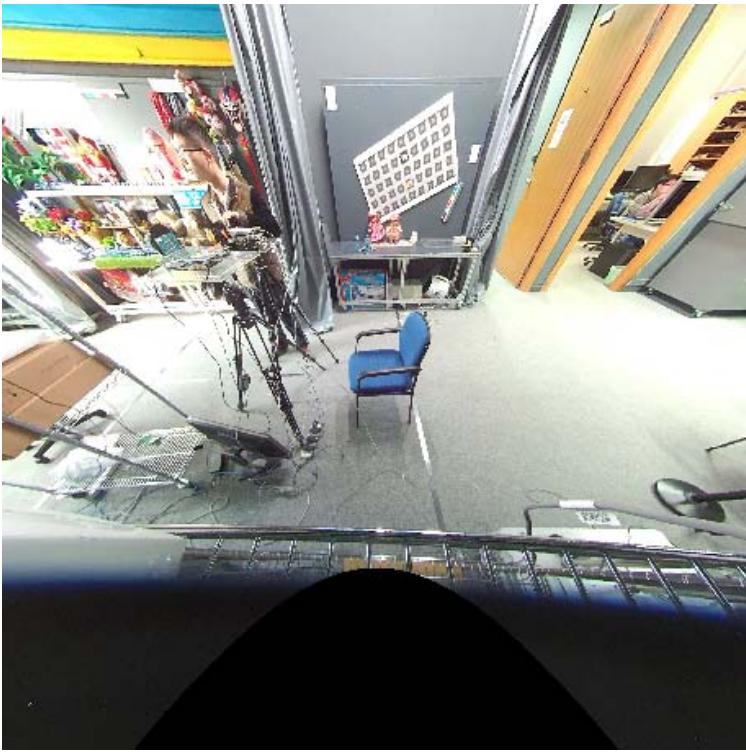
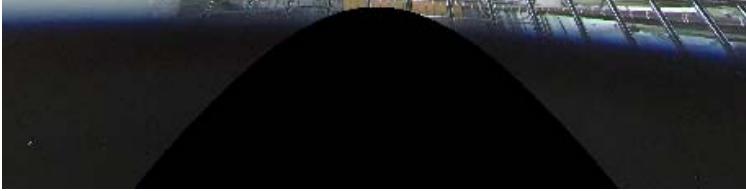
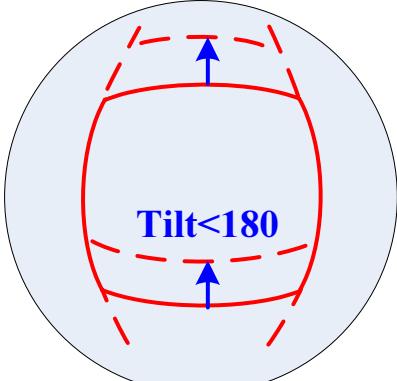
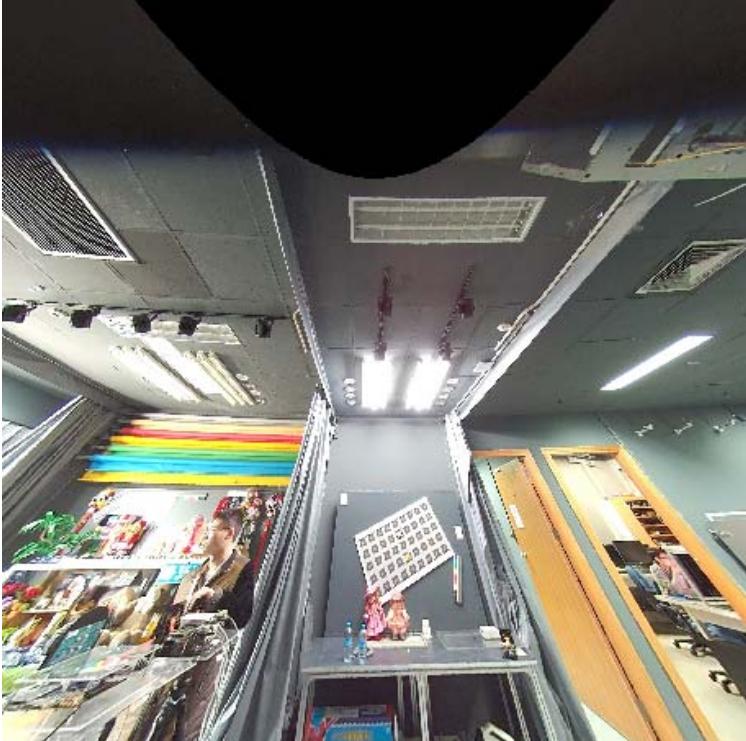


Typical Parameter Setting		Diagram	Correction Effect Demonstration
FanStrength	0		
TrapezoidCoef = 32	The trapezoid strength coefficient TrapezoidCoef is valid only during normal correction and 180° correction in wall mount mode.		

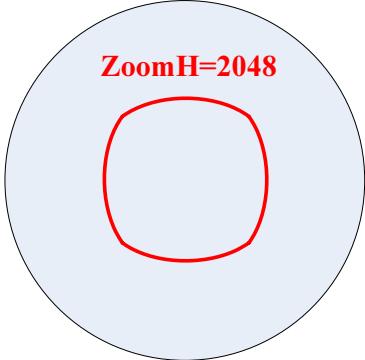


Typical Parameter Setting	Diagram	Correction Effect Demonstration
Pan = 270		
Pan = 90		



Typical Parameter Setting	Diagram	Correction Effect Demonstration
Tilt = 270		 
Tilt = 90		 



Typical Parameter Setting	Diagram	Correction Effect Demonstration
ZoomH = 2048 (The field angle when ZoomH is 2048 is 63°, which is only half of that when ZoomH is 4095 .)		

- Ceiling mount and floor mount

In ceiling mount and floor mount modes, the 360° panoramic correction mode and normal correction mode are recommended. [Table 11-6](#) to [Table 11-9](#) describe the effect demonstration and parameter description.

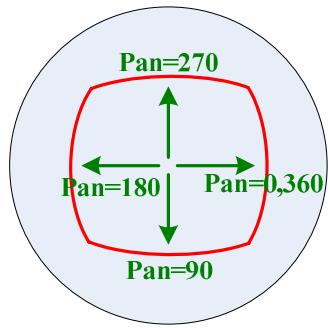
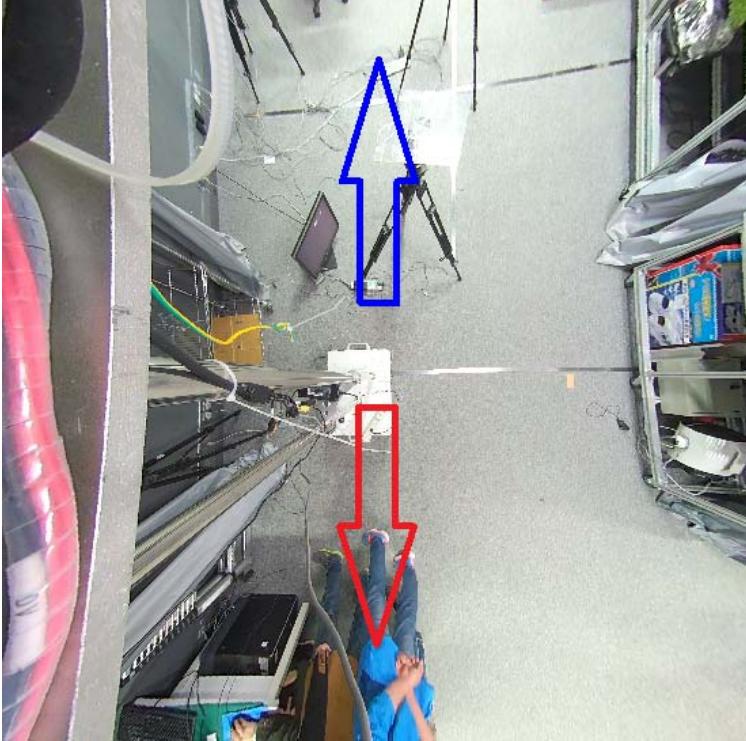
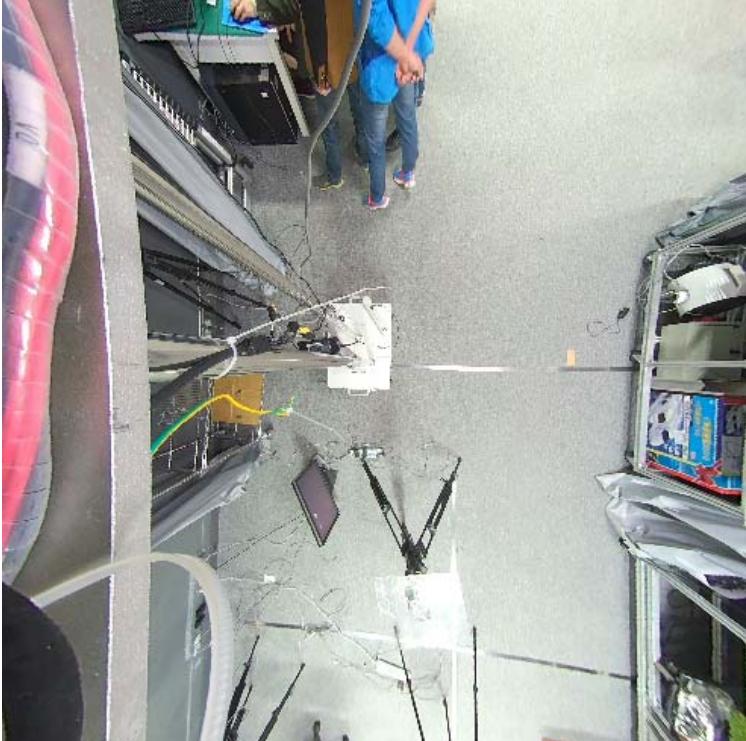
Table 11-6 Parameter description in ceiling/floor mount normal correction mode

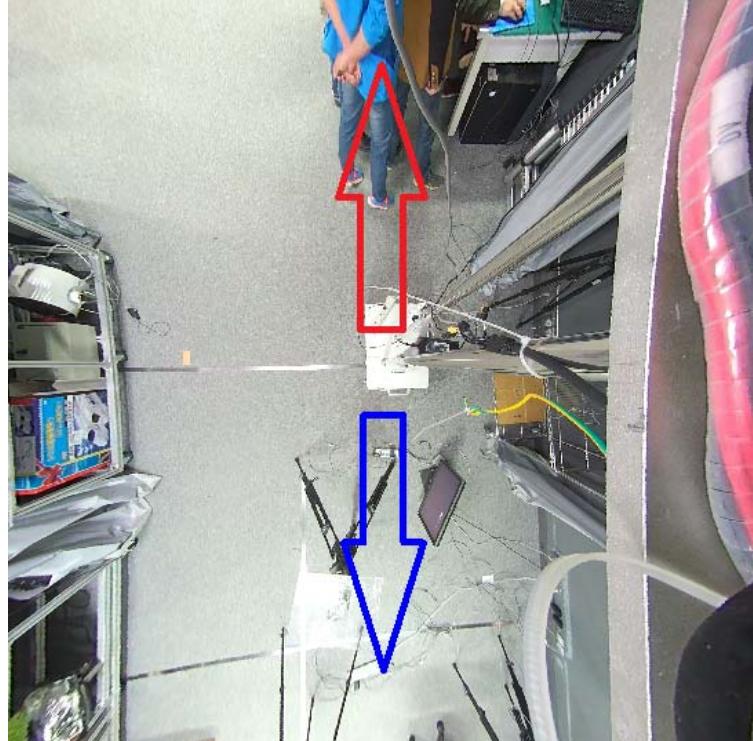
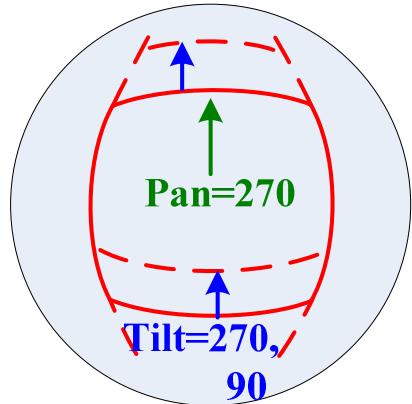
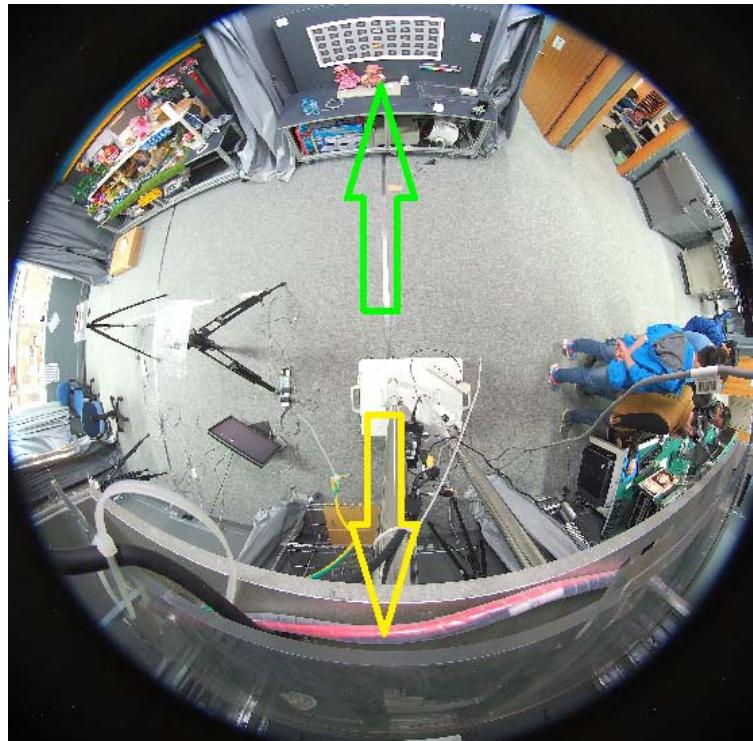
Parameter	Description
Pan	The direction in the Pan angle of the original picture is selected as the direction right above the corrected picture. Rotation range: [0, 360°]
Tilt	The absolute value of the difference between Tilt and 180 indicates the amplitude of the field angle rotates to the direction determined by Pan . Rotation range: [0, 90°]
ZoomH	Horizontal and vertical field angles, 4095 at the maximum (126°. The horizontal and vertical field angles are the same in normal mode.)
ZoomV	Invalid parameter



Table 11-7 Effect demonstration in ceiling/floor mount normal correction mode

Typical Parameter Setting		Diagram	Correction Effect Demonstration		
Input picture width and height		<p>The content on the left is the typical setting in ceiling mount normal correction mode (the output picture width and height are equal to the input picture width and height). The following diagrams are the demonstration diagrams when the value of a specific parameter is changed under the typical setting. The pictures on the right are the corresponding correction effect pictures.</p>			
InWidth	3000				
InHeight	3000				
Normal correction parameters					
Mount mode	Ceiling mount				
OutWidth	3000				
OutHeight	3000				
HorOffset	0				
VerOffset	0				
OutRadius	1500				
Pan	180				
Tilt	180				
ZoomH	4095				

Typical Parameter Setting	Diagram	Correction Effect Demonstration
<p>Pan is used to select the upward direction of the correction region in the corrected picture after normal correction.</p> <p>When Pan is 180, the left part of the correction region is in the upward direction of the corrected picture, as shown by the blue arrow in the right picture.</p>		
<p>Mount mode: floor mount</p> <p>(The effect of normal correction in floor mount mode is similar to that in ceiling mount mode except that the picture is flipped up/down on the basis of the picture in ceiling mount mode.)</p>		

Typical Parameter Setting	Diagram	Correction Effect Demonstration
When Pan is 0 or 360 , the right part of the correction region is in the upward direction of the corrected picture, as shown by the red arrow in the right picture.		
In normal correction ceiling mount mode, the absolute value of the difference between Tilt and 180 indicates the rotation angle of the field angle determined by ZoomH . The rotation direction is determined by Pan . Pan = 270 Tilt = 270 or 90 The green arrow in the original picture points to the upward direction in the corrected picture. The effect is the same when Tilt is 270 and 90 .		



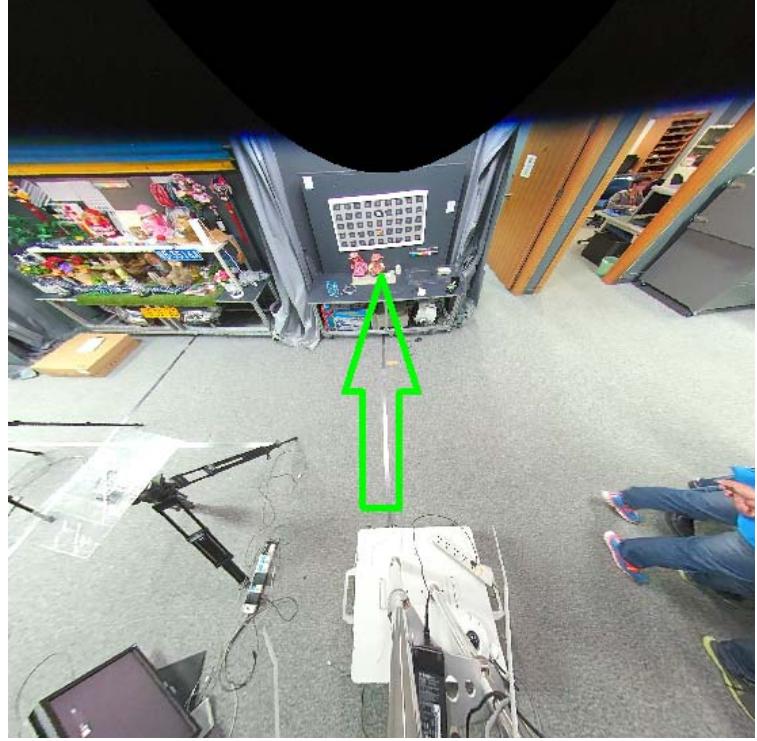
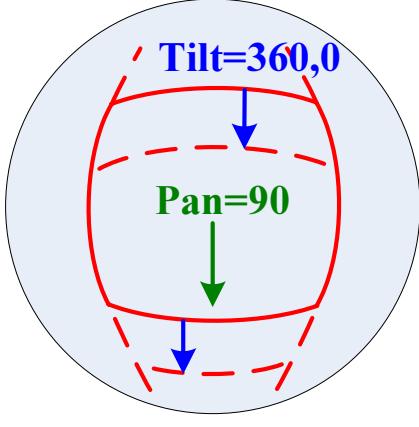
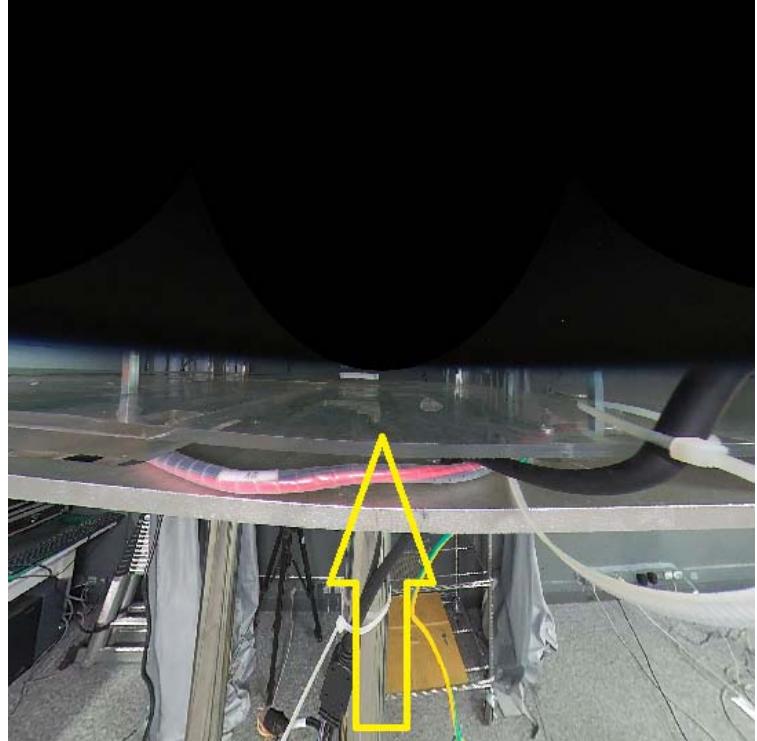
Typical Parameter Setting	Diagram	Correction Effect Demonstration
		
<p>Pan = 90 Tilt = 360 or 0 The yellow arrow in the original picture points to the upward direction in the corrected picture.</p> 		



Table 11-8 Parameter description in ceiling/floor mount 360° panoramic correction mode

Parameter	Description
Pan	Start position for the leftmost part of the corrected picture relative to the original picture, that is, start position of the correction region in the original picture
Tilt	Parameter indicating whether the correction region moves inward or outward relative to the original picture. If Tilt is greater than 180 , the correction region moves outward. If Tilt is less than 180 , the correction region moves inward.
ZoomH	Range (Amplitude) of the correction region. The value 4095 indicates that one circle is selected, and the value 2048 indicates that half a circle is selected.
ZoomV	Height of the correction region within the range determined by InRadius and OutRadius
InRadius	Radius of the visible inner ring
OutRadius	Radius of the visible outer ring. This parameter is generally set to the radius of the original picture for the fisheye.

Table 11-9 Effect demonstration in ceiling/floor mount 360° panoramic correction mode

Typical Parameter Setting	Diagram	Correction Effect Demonstration
Input picture width and height	The content on the left is the typical setting in ceiling mount 360° panoramic correction mode (the output picture width is equal to the input picture width, and the output picture height is 1/3 of the input picture height). The following diagrams are the demonstration diagrams when the value of a specific parameter is changed under the typical setting. The pictures on the right are the corresponding correction effect pictures.	
InWidth	3000	
InHeight	3000	
360° correction parameters		
Mount mode	Ceiling mount	
OutWidth	3000	
OutHeight	1000	
HorOffset	0	
VerOffset	0	



Typical Parameter Setting		Diagram	Correction Effect Demonstration
InRadius	0		
OutRadius	1500		
Pan	180		
Tilt	180		
ZoomH	4095		
ZoomV	4095		
Mount mode: floor mount (The effect of 360° panoramic correction in floor mount mode is similar to that in ceiling mount mode except that the picture is flipped up/down on the basis of the picture in ceiling mount mode.)			
ZoomV = 2048 OutRadius = 1500 InRadius = 100 ZoomV is used to adjust the height of the correction region within the range determined by InRadius and OutRadius . The height of the correction region can be calculated as follows: $h = (\text{OutRadius} - \text{InRadius}) \times \text{ZoomV}/4095$ The shadow region in the following figure is the selected correction region:			



Typical Parameter Setting	Diagram	Correction Effect Demonstration
<p>ZoomV = 4095</p> <p>OutRadius = 1300 (OutRadius is set to 1300 to demonstrate the effect. It should be set to the actual radius of the original picture, which is 1500. ZoomV needs to be used to select the correction region within the range determined by InRadius and OutRadius.)</p> <p>InRadius = 200</p> <p>(Compared with the corrected picture under the typical setting, some regions in the upper and lower parts of the corrected picture are cropped when InRadius is set to 200.)</p>		
<p>Pan = 270</p> <p>(This parameter is used to select the start position of the correction region.)</p>		
Pan = 360		



Typical Parameter Setting	Diagram	Correction Effect Demonstration
<p>ZoomV = 4095 OutRadius = 1500 InRadius = 100 Tilt = 90</p> <p>Tilt is used to move the correction region inward or outward. When Tilt is less than 180, the correction region is moved inward. When Tilt is greater than 180, the correction region is moved outward. The inside and outside boundaries of the correction region are determined by OutRadius' and InRadius' as follows:</p> $\text{OutRadius}' = \text{OutRadius}' + \text{offset}$ $\text{InRadius}' = \text{InRadius} + \text{offset}$ <p>where</p> $\text{Offset} = \text{OutRadius} \times (\text{Tilt} - 180) / 360$ $\text{OutRadius}' = \text{InRadius} + (\text{OutRadius} - \text{InRadius}) \times \text{ZoomV} / 4095$		



Typical Parameter Setting	Diagram	Correction Effect Demonstration
<p>ZoomV = 4095 OutRadius = 1500 InRadius = 100 Tilt = 270</p> <p>For the fisheye hardware, OutRadius specifies the boundary of the picture content. The parts that exceed OutRadius are considered to be invalid contents. To be specific:</p> <p>When OutRadius' is greater than OutRadius, black parts (shadow region in the following figure) appear in the corrected picture. When InRadius' is less than 0, black parts also appear.</p>		
<p>ZoomH = 2048</p> <p>(The correction region is only half of the original picture. To obtain better effect, you are advised to downscale the width of the output picture based on a certain ratio.)</p>		



Typical Parameter Setting	Diagram	Correction Effect Demonstration
ZoomH = 2048 OutWidth = 1500		

NOTE

- If the picture is flipped after fisheye correction in the floor mount 360° panoramic correction mode or normal correction mode, it is recommended that mirroring be performed on the picture in the VI or VPSS module.
- The calculation amount and bandwidth of fisheye processing differs greatly under different settings. The performance may be insufficient under atypical settings.

11.2.2 Function Description

The functions supported by the fisheye subsystem include fisheye correction, LDC, compression, output picture combination, and overlaying of the background color for the output picture.

- Fisheye correction

The fisheye subsystem can implement fisheye correction on a picture.



CAUTION

Three fisheye correction modes are supported: 360° panoramic correction, 180° panoramic correction, and normal correction. Three mount modes are supported in each correction mode: floor mount, ceiling mount, and wall mount.

- LDC

The fisheye subsystem can implement LDC on a picture.



CAUTION

When the fisheye subsystem is used to implement LDC, the size of the input picture must be the same as that of the output picture. LDC and scaling cannot be supported simultaneously.

- Scaling

The fisheye subsystem can implement simultaneous scaling and fisheye correction on a picture. The fisheye scaling effect is not good. Therefore, the typical parameter setting is



recommended, and you are advised not to scale the picture by using the fisheye subsystem.



CAUTION

The maximum sizes of the input and output pictures for the fisheye subsystem are 4608 x 3456, the minimum size of the input picture is 1920 x 1080, and the minimum size of the output picture is 960 x 360. When fisheye correction is being implemented, scaling is supported within the range allowed by the output picture.

- Compression

The fisheye subsystem supports only uncompressed linear input pictures and compressed/uncompressed linear output pictures. When the compressed output is used, if the number of correction regions is 1 and the output size of the correction region is capable of covering the output range, the background color does not need to be overlaid. The background color must be overlaid in other cases.

- Output picture combination

The fisheye subsystem can implement correction on multiple regions of a picture, and combine the corrected picture regions for output. When the uncompressed output is used, the horizontal (X) coordinates for each region must be 16-pixel-aligned and the vertical (Y) coordinates for each region must be 2-pixel-aligned; when the compressed output is used, the horizontal (X) coordinates for each region must be 256-pixel-aligned and the vertical (Y) coordinates for each region must be 2-pixel-aligned. Besides, the width of each region needs to be 256-byte-aligned, or equal to the difference value of the output picture width and horizontal (X) coordinate of the region.



CAUTION

Output picture combination indicates that at most four regions are corrected and combined into one output picture in the same extended video input (VI) or video processing subsystem (VPSS) channel. The outputs of multiple extended VI/VPSS channels after fisheye correction cannot be combined for output.

- Overlaying of the background color for the output picture

The fisheye subsystem can overlay the background color on the output picture for outputting the combined picture with seams. If the combined picture without a seam needs to be output, it is recommended that background color overlaying be disabled to improve performance.

[Table 11-10](#) and [Table 11-11](#) list the hardware specifications of the fisheye subsystems of Hi3519 V100 and Hi3519 V101, respectively.



Table 11-10 Hardware specifications of the fisheye subsystem of Hi3519 V100

Item	Description
Data format of the video input	<ul style="list-style-type: none">• Semi-planar420 format (single component)• Linear format• Uncompressed pictures
Input picture resolution	<ul style="list-style-type: none">• Minimum resolution: 1920 x 1080 for fisheye correction, and 640 x 480 for LDC• Maximum resolution: 4608 x 3456• 2-pixel-aligned width and height, 16-pixel-aligned stride
Data format of the video output	<ul style="list-style-type: none">• Semi-planar420 format (single component)• Linear format• Compressed or uncompressed pictures
Output picture resolution	<ul style="list-style-type: none">• Minimum resolution: 960 x 360 for fisheye correction, and 640 x 480 for LDC• Maximum resolution: 4608 x 3456• 2-pixel-aligned height; 2-pixel-aligned width for the uncompressed output; 256-pixel-aligned width or (Output width – Horizontal coordinate of the correction region) width for the compressed output; 16-pixel-aligned stride
Scaling capability	<ul style="list-style-type: none">• Simultaneous scaling and LDC not supported• Picture scaling within the range allowed by the output picture (960 x 360 to 4608 x 3456) during fisheye correction
Value range of PTZ parameters	<ul style="list-style-type: none">• Pan: [0, 360]• Tilt: [0, 360]• ZoomH: [1, 4095]• ZoomV: [1, 4095]

Table 11-11 Hardware specifications of the fisheye subsystem of Hi3519 V101

Item	Description
Data format of the video input	<ul style="list-style-type: none">• Semi-planar420 format (single component)• Linear format• Uncompressed pictures
Input picture resolution	<ul style="list-style-type: none">• Minimum resolution: 1920 x 1080 for fisheye correction, and 640 x 480 for LDC• Maximum resolution: 4608 x 3456• 2-pixel-aligned width and height, 16-pixel-aligned stride



Item	Description
Data format of the video output	<ul style="list-style-type: none">• Semi-planar420 format (single component)• Linear format• Compressed or uncompressed pictures
Output picture resolution	<ul style="list-style-type: none">• Minimum resolution: 960 x 360 for fisheye correction, and 640 x 480 for LDC• Maximum resolution: 4608 x 3456• 2-pixel-aligned height; 2-pixel-aligned width for the uncompressed output; 256-pixel-aligned width or (Output width – Horizontal coordinate of the correction region) width for the compressed output; 16-pixel-aligned stride
Scaling capability	<ul style="list-style-type: none">• Simultaneous scaling and LDC not supported• Picture scaling within the range allowed by the output picture (960 x 360 to 4608 x 3456) during fisheye correction
Value range of PTZ parameters	<ul style="list-style-type: none">• Pan: [0, 360]• Tilt: [0, 360]• ZoomH: [1, 4095]• ZoomV: [1, 4095]

11.3 Fisheye Tool Library

11.3.1 Fisheye Calibration Tool Library

The fisheye calibration tool library is provided to automatically determine the position features (offset and radius) of the fisheye lens based on the picture captured by the fisheye lens. Currently, this calibration tool library is only supported by the Hi3519 V101 and Hi3519 V100, and needs to work with the fisheye lens.

To guarantee that the tool can correctly determine the fisheye outline, ensure that the fisheye picture region and corners in the black borders are clear and identifiable during snapshot.

11.3.1.1 API Reference

APIs related to the fisheye calibration tool are defined as follows:

- **HI_FISHEYE_ComputeCalibrateResult:** Calculates the calibration result.
- **HI_FISHEYE_MarkCalibrateResult:** Marks out the calibration result on the output picture according to the input picture and calculated calibration result.

HI_FISHEYE_ComputeCalibrateResult

[Description]

Calculates the calibration result.



[Syntax]

```
HI_S32 HI_FISHEYE_ComputeCalibrateResult(VIDEO_FRAME_S *pstVFrameIn,  
CALIBRATE_OUTPUT_S* pOutCalibrate);
```

NOTE

For details about VIDEO_FRAME_S, see chapter 2 "System Control."

[Parameter]

Parameter	Description	Input/Output
pstVFrameIn	Frame information address of the original input picture	Input
pOutCalibrate	Calibration output	Output

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

- Header files: **fisheye_calibrate.h**, **hi_comm_video.h**
- Library file: **libhifisheyecalibrate.a**

[Note]

- Only the pictures in PIXEL_FORMAT_YUV_SEMIPLANAR_420 format can be calibrated.
- The **pstVFrameIn** and **pOutCalibrate** pointers cannot be null or invalid.

HI_FISHEYE_MarkCalibrateResult

[Description]

Marks out the calibration result on the output picture according to the input picture and calculated calibration result.

[Syntax]

```
HI_S32 HI_FISHEYE_MarkCalibrateResult(VIDEO_FRAME_S* pstPicIn,  
VIDEO_FRAME_S* pstPicOut, FISHEYE_CALIBRATE_RESULT_S* pCalibrateResult);
```

NOTE

For details about VIDEO_FRAME_S, see chapter 2 "System Control."

[Parameter]



Parameter	Description	Input/Output
pstVFramIn	Frame information address of the original picture	Input
pstPicOut	Frame information address of the target picture	Input
pCalibrateResult	Calibration result obtained by calling <code>HI_FISHEYE_ComputeCalibrateResul</code>	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure

[Requirement]

- Header files: **fisheye_calibrate.h**, **hi_comm_video.h**
- Library file: **libhifisheyecalibrate.a**

[Note]

- Only the pictures whose input and output are both in `PIXEL_FORMAT_YUV_SEMIPLANAR_420` format can be calibrated.
- The **pstVFramIn**, **pstPicOut**, and **pCalibrateResult** pointers cannot be null or invalid.

11.3.1.2 Data Structures

The data structures related to the fisheye calibration tool are defined as follows:

- **CALIBTATE_OUTPUT_S**: Defines the calibration output.
- **FISHEYE_CALIBTATE_RESULT_S**: Defines the calibration output result.

CALIBTATE_OUTPUT_S

[Description]

Defines the calibration output.

[Syntax]

```
typedef struct hCALIBTATE_OUTPUT_S
{
    FISHEYE_CALIBTATE_RESULT_S stCalibrateResult; /*the output of result*/
}CALIBTATE_OUTPUT_S;
```

[Member]

Member	Description
FISHEYE_CALIBTATE_RESULT_S	Calibration output result



[Note]

None

[See Also]

[FISHEYE_CALIBTATE_RESULT_S](#)

FISHEYE_CALIBTATE_RESULT_S

[Description]

Defines the calibration output result.

[Syntax]

```
typedef struct hiFISHEYE_CALIBTATE_RESULT_S
{
    HI_S32 s32OffsetV;          /* the horizontal offset between image center and
                                 physical center of len*/
    HI_S32 s32OffsetH;          /* the vertical offset between image center and
                                 physical center of len*/
    HI_S32 s32Radius_X;         /* the X coordinate of physical center of len*/
    HI_S32 s32Radius_Y;         /* the Y coordinate of physical center of len*/
    HI_U32 u32Radius;           /* the radius of len*/
}FISHEYE_CALIBTATE_RESULT_S;
```

[Member]

Member	Description
s32OffsetV	Horizontal offset of the lens center point relative to the sensor center point, in pixel
s32OffsetH	Vertical offset of the lens center point relative to the sensor center point, in pixel
s32Radius_X	Horizontal coordinate of the circle center of the calibration result
s32Radius_Y	Vertical coordinate of the circle center of the calibration result
u32Radius	Outer radius of the calibration result

[Note]

None

[See Also]

[CALIBTATE_OUTPUT_S](#)



11.4 API Reference

You need to call the following MPIs provided in the video input unit (VIU) and VPSS to enable or disable fisheye functions and set fisheye attributes:

- `HI_S32 HI_MPI_VI_SetFisheyeDevConfig`: Sets the lens multiplier factor (LMF) parameter of the fisheye lens corresponding to the VI device.
- `HI_S32 HI_MPI_VI_GetFisheyeDevConfig`: Obtains the LMF parameters of the fisheye lens corresponding to the VI device.
- `HI_S32 HI_MPI_VI_SetFisheyeAttr`: Sets the fisheye attribute of an extended VI channel.
- `HI_S32 HI_MPI_VI_GetFisheyeAttr`: Obtains the fisheye attribute of an extended VI channel.
- `HI_S32 HI_MPI_VPSS_SetFisheyeConfig`: Sets the LMF parameters of the fisheye lens corresponding to the VPSS group.
- `HI_S32 HI_MPI_VPSS_GetFisheyeConfig`: Obtains the LMF parameters of the fisheye lens corresponding to the VPSS group.
- `HI_S32 HI_MPI_VPSS_SetFisheyeAttr`: Sets the fisheye attribute of the extended channel in the VPSS group.
- `HI_S32 HI_MPI_VPSS_GetFisheyeAttr`: Obtains the fisheye attribute of the extended channel in the VPSS group.

For details, see the **Description**, **Note**, and **Example** fields of the MPIs in chapter 3 "VI" and chapter 5 "VPSS" of the *HiMPP IPC V3.0 Media Processing Software Development Reference*.

The fisheye subsystem provides the following MPIs:

- `HI_MPI_FISHEYE_BeginJob`: Starts a job.
- `HI_MPI_FISHEYE_SetConfig`: Sets the configuration of the fisheye lens.
- `HI_MPI_FISHEYE_AddCorrectionTask`: Adds a fisheye correction task to a started job.
- `HI_MPI_FISHEYE_EndJob`: Submits a job to the fisheye subsystem.
- `HI_MPI_FISHEYE_CancelJob`: Cancels a job.
- `fisheye_mod_init`: Initializes the module.

HI_MPI_FISHEYE_BeginJob

[Description]

Starts a job.

[Syntax]

```
HI_S32 HI_MPI_FISHEYE_BeginJob(FISHEYE_HANDLE *phHandle);
```

[Parameter]

Parameter	Description	Input/Output
phHandle	Parameter used to return the handle	Output



[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 11.6 "Error Codes."

[Requirement]

- Header files: **hi_comm_fisheye.h**, **mpi_fisheye.h**
- Library file: **libmpi.a**

[Note]

- Multiple jobs can be started at a time. However, the handle returned by **phHandle** can be used only when the code indicating success is returned after **HI_MPI_FISHEYE_BeginJob** is called.
- The **phHandle** pointer cannot be null or invalid.

HI_MPI_FISHEYE_SetConfig

[Description]

Sets the configuration of the fisheye lens.

[Syntax]

```
HI_S32 HI_MPI_FISHEYE_SetConfig(FISHEYE_HANDLE hHandle, const
FISHEYE_JOB_CONFIG_S *pstJobConfig);
```

[Parameter]

Parameter	Description	Input/Output
hHandle	Handle of a started job	Input
pstJobConfig	Pointer to the job configuration, including LMF coefficients	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 11.6 "Error Codes."

[Requirement]



- Header files: **hi_comm_fisheye.h**, **mpi_fisheye.h**, and **hi_comm_video.h**
- Library file: **libmpi.a**

[Note]

None

[Example]

See the example of [HI_MPI_FISHEYE_CancelJob](#).

HI_MPI_FISHEYE_AddCorrectionTask

[Description]

Adds a fisheye correction task to a started job.

[Syntax]

```
HI_S32 HI_MPI_FISHEYE_AddCorrectionTask(FISHEYE_HANDLE hHandle,  
FISHEYE_TASK_ATTR_S *pstTask, const FISHEYE_ATTR_S *pstFisheyeAttr);
```

[Parameter]

Parameter	Description	Input/Output
hHandle	Handle of a started job	Input
pstTask	Pointer to fisheye task attributes	Input
pstFisheyeAttr	Pointer to the attribute configuration of the fisheye correction region	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 11.6 "Error Codes."

[Requirement]

- Header files: **hi_comm_fisheye.h**, **mpi_fisheye.h**, and **hi_comm_video.h**
- Library file: **libmpi.a**

[Note]

- The fisheye job identified by **hHandle** must be a started job.
- **bEnable** in **pstFisheyeAttr** must be set to **1**.
- The task attributes must meet the capability of the fisheye subsystem.
- Each region does not support the **FISHEYE_NO_TRANSFORMATION** tasks.



- If an error code indicating failure is returned after this MPI is called and no task needs to be added, you can submit added tasks by calling [HI_MPI_FISHEYE_EndJob](#). If an error code indicating failure is returned after this MPI is called and there are tasks to be added, you must cancel the job identified by **hHandle** by calling [HI_MPI_FISHEYE_CancelJob](#). Otherwise, the job identified by **hHandle** cannot be reused.

[Example]

See the example of [HI_MPI_FISHEYE_CancelJob](#).

HI_MPI_FISHEYE_EndJob

[Description]

Submits a job to the fisheye subsystem.

[Syntax]

```
HI_S32 HI_MPI_FISHEYE_EndJob(FISHEYE\_HANDLE hHandle);
```

[Parameter]

Parameter	Description	Input/Output
hHandle	Handle of a started job	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 11.6 "Error Codes."

[Requirement]

- Header files: **hi_comm_fisheye.h**, **mpi_fisheye.h**
- Library file: **libmpi.a**

[Note]

- The job identified by **hHandle** must be a started job.
- If an error code indicating failure is returned after this MPI is called, you must cancel the job identified by **hHandle** by calling [HI_MPI_FISHEYE_CancelJob](#). Otherwise, the job identified by **hHandle** cannot be reused.

[Example]

See the example of [HI_MPI_FISHEYE_BeginJob](#).

HI_MPI_FISHEYE_CancelJob

[Description]



Cancels a job.

[Syntax]

```
HI_S32 HI_MPI_FISHEYE_CancelJob(FISHEYE_HANDLE hHandle);
```

[Parameter]

Parameter	Description	Input/Output
hHandle	Handle of a started job	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 11.6 "Error Codes."

[Requirement]

- Header files: **hi_comm_fisheye.h**, **mpi_fisheye.h**
- Library file: **libmpi.a**

[Note]

The job identified by **hHandle** must be a started job.

[Example]

```
HI_S32 s32Ret = HI_SUCCESS;
FISHEYE_HANDLE hHandle;
FISHEYE_TASK_ATTR_S stTask;
FISHEYE_ATTR_S      stFisheyeAttr;
FISHEYE_JOB_CONFIG_S stFisheyeJobConfig;
...
s32Ret = HI_MPI_FISHEYE_BeginJob(&hHandle);
if(s32Ret != HI_SUCCESS)
{
    goto HandleErr;
}
...s32Ret = HI_MPI_FISHEYE_SetConfig(hHandle,&stFisheyeJobConfig);
if(s32Ret != HI_SUCCESS)
{
    HI_MPI_FISHEYE_CancelJob(hHandle);
    goto HandleErr;
}
...
...
```



```
s32Ret = HI_MPI_FISHEYE_AddCorrectionTask(hHandle, &stTask,  
&stFisheyeAttr);  
if(s32Ret != HI_SUCCESS)  
{  
    HI_MPI_FISHEYE_CancelJob(hHandle);  
    goto HandleErr;  
}  
  
s32Ret = HI_MPI_FISHEYE_EndJob(hHandle);  
if(s32Ret != HI_SUCCESS)  
{  
    HI_MPI_FISHEYE_CancelJob(hHandle);  
    goto HandleErr;  
}
```

fisheye_mod_init

[Description]

Initializes the module.

[Syntax]

```
int fisheye_mod_init(void *pArgs);
```

[Parameter]

Parameter	Description	Input/Output
pArgs	Module parameter	Input

[Return Value]

Return Value	Description
0	Success
Other values	Failure. Its value is an error code. For details, see section 11.6 "Error Codes."

[Requirement]

Header file: **hi_module_param.h**

[Note]

- This MPI applies only to the Huawei LiteOS version, and is not contained in the Linux version.
- Hi3519 V100 does not support this MPI.

[Example]



See **fisheye_init.c** and **sdk_init.c**.

11.5 Data Structures

The data structures of the fisheye subsystem are as follows:

- **FISHEYE_HANDLE**: Defines the handle of a fisheye job.
- **FISHEYE_TASK_ATTR_S**: Defines attributes of a fisheye task.
- **FISHEYE_LMFCOEF_NUM**: Defines the number of LMF parameters for the fisheye lens.
- **FISHEYE_CONFIG_S**: Defines the configuration of the LMF parameters for the fisheye lens.
- **FISHEYE_JOB_CONFIG_S**: Defines the configuration of the LMF parameter for the fisheye lens to which a fisheye job corresponds.
- **FISHEYE_MOUNT_MODE_E**: Defines the mount mode in the fisheye attribute.
- **FISHEYE_VIEW_MODE_E**: Defines the correction mode in the fisheye attribute.
- **RECT_S**: Defines the attributes of the rectangular region.
- **FISHEYE_REGION_ATTR_S**: Defines the configuration of attributes for each fisheye correction region.
- **FISHEYE_MAX_REGION_NUM**: Defines the maximum number of correction regions supported by the fisheye subsystem.
- **FISHEYE_ATTR_S**: Defines the configuration related to the fisheye attribute.
- **SPREAD_ATTR_S**: Defines the configuration related to the broadening attribute.
- **FISHEYE_CYLIND_ATTR_S**: Defines the configuration related to the cylindrical projection attribute.
- **FISHEYE_MIN_IN_IMAGE_WIDTH**: Defines the minimum width of the input image for fisheye correction.
- **FISHEYE_MIN_IN_IMAGE_HEIGHT**: Defines the minimum height of the input image for fisheye correction.
- **FISHEYE_MIN_OUT_IMAGE_WIDTH**: Defines the minimum width of the output image for fisheye correction.
- **FISHEYE_MIN_OUT_IMAGE_HEIGHT**: Defines the minimum height of the output image for fisheye correction.
- **LDC_MIN_IMAGE_WIDTH**: Defines the minimum width of the input/output image for LDC.
- **LDC_MIN_IMAGE_HEIGHT**: Defines the minimum height of the input/output image for LDC.
- **FISHEYE_MAX_IMAGE_WIDTH**: Defines the maximum image width supported by the fisheye module.
- **FISHEYE_MAX_IMAGE_HEIGHT**: Defines the maximum image height supported by the fisheye module.
- **FISHEYE_MODULE_PARAMS_S**: Sets fisheye module parameters in the `sdk_init` function of Huawei LiteOS.



FISHEYE_HANDLE

[Description]

Defines the handle of a fisheye job.

[Syntax]

```
typedef HI_S32      FISHEYE_HANDLE;
```

[Note]

None

[See Also]

None

FISHEYE_TASK_ATTR_S

[Description]

Defines attributes of a fisheye task.

[Syntax]

```
typedef struct hiFISHEYE_TASK_ATTR_S
{
    VIDEO_FRAME_INFO_S      stImgIn;        /* input picture */
    VIDEO_FRAME_INFO_S      stImgOut;       /* output picture */
    HI_U32                  au32privateData[4]; /* task's private data */
    HI_U32                  reserved;       /* save current picture's state while
debug */
} FISHEYE_TASK_ATTR_S;
```

[Member]

Member	Description
stImgIn	Input picture attribute
stImgOut	Output picture attribute
au32privateData	Task-related private data. The fisheye subsystem does not use and modify the data.
reserved	Reserved

[Note]

The input picture attribute and output picture attribute should meet the attribute requirements.

[See Also]

VIDEO_FRAME_INFO_S



FISHEYE_LMFCOEF_NUM

[Description]

Defines the number of LMF parameters for the fisheye lens.

[Syntax]

```
#define FISHEYE_LMFCOEF_NUM 128
```

[Note]

None

[See Also]

[FISHEYE_CONFIG_S](#)

FISHEYE_CONFIG_S

[Description]

Defines the configuration of the LMF parameters for the fisheye lens.

[Syntax]

```
typedef struct hiFISHEYE_CONFIG_S
{
    HI_U16      au16LMFCoeff[FISHEYE_LMFCOEF_NUM]; /* LMF parameter of
fisheye lens */
} FISHEYE_CONFIG_S;
```

[Member]

Member	Description
au16LMFCoeff	LMF parameter for the fisheye lens

[Note]

- If the the [FISHEYE_ATTR_S](#) enables the function of using LMF parameters in the `HI_MPI_VI_SetFisheyeAttr` of the VI or `HI_MPI_VPSS_SetFisheyeAttr` of the VPSS, `HI_MPI_VI_SetFisheyeDevConfig` of the VI or `HI_MPI_VPSS_SetFisheyeConfig` of the VPSS must be called to set the LMF parameters. Otherwise, an error occurs during fisheye processing because the configured LMF parameters fail to be obtained.
- The LMF parameters need to be converted based on the recommended parameter setting of the lens vendor before configuration (the correctly configured LMF parameters should comply with the increasing rule of $au16LMFCoeff[i+1] \geq au16LMFCoeff[i]+5$). If the configured LMF parameters do not comply with the increasing rule of $au16LMFCoeff[i+1] \geq au16LMFCoeff[i]+5$, errors occur. If the LMF parameters are incorrectly configured, exceptions such as bus errors may occur. You are advised to disable the LMF function if there is no parameter provided by the lens vendor.

[See Also]

[FISHEYE_ATTR_S](#)



FISHEYE_JOB_CONFIG_S

[Description]

Defines the configuration of the LMF parameter for the fisheye lens to which a fisheye task corresponds.

[Syntax]

```
typedef struct hiFISHEYE_JOB_CONFIG_S
{
    HI_U32    u32LenMapPhyAddr; /*LMF coefficient Physic Addr*/
} FISHEYE_JOB_CONFIG_S;
```

[Member]

Member	Description
u32LenMapPhyAddr	Physical addresses to save the LMF coefficients for the fisheye lens.

[Note]

- In [HI_MPI_FISHEYE_AddCorrectionTask](#), if the function of using LMF parameters is enabled in [FISHEYE_ATTR_S](#), the LMF parameters must be configured by calling [HI_MPI_FISHEYE_SetConfig](#). Otherwise, an error occurs during fisheye processing because the configured LMF parameters fail to be obtained.
- The LMF parameters need to be configured by following the recommended parameter setting of the lens vendor (the correctly configured LMF parameters should be in increasing order of $au16LMFCoeff[i+1] \geq au16LMFCoeff[i] + 5$). If the configured LMF parameters are not in this increasing order, errors occur. If the LMF parameters are incorrectly configured, exceptions such as bus errors may occur. You are advised to disable the LMF function if there is no parameter provided by the lens vendor.
- **u32LenMapPhyAddr** can be the address of the MMZ memory only.

[See Also]

[FISHEYE_ATTR_S](#)

FISHEYE_MOUNT_MODE_E

[Description]

Defines the mount mode in the fisheye attribute.

[Syntax]

```
typedef enum hiFISHEYE_MOUNT_MODE_E
{
    FISHEYE_DESKTOP_MOUNT    = 0,          /* desktop mount mode */
    FISHEYE_CEILING_MOUNT    = 1,          /* ceiling mount mode */
    FISHEYE_WALL_MOUNT       = 2,          /* wall mount mode */
    FISHEYE_MOUNT_MODE_BUTT
} FISHEYE_MOUNT_MODE_E;
```



[Member]

Member	Description
FISHEYE_DESKTOP_MOUNT	Floor mount mode
FISHEYE_CEILING_MOUNT	Ceiling mount mode
FISHEYE_WALL_MOUNT	Wall mount mode

[Note]

None

[See Also]

None

FISHEYE_VIEW_MODE_E

[Description]

Defines the correction mode in the fisheye attribute.

[Syntax]

```
typedef enum hiFISHEYE_VIEW_MODE_E
{
    FISHEYE_VIEW_360_PANORAMA = 0,      /* 360 panorama mode of fisheye
correction */
    FISHEYE_VIEW_180_PANORAMA = 1,      /* 180 panorama mode of fisheye
correction */
    FISHEYE_VIEW_NORMAL         = 2,      /* normal mode of fisheye correction
*/
    FISHEYE_NO_TRANSFORMATION  = 3,      /* no fisheye correction */
    FISHEYE_VIEW_MODE_BUTT
} FISHEYE_VIEW_MODE_E;
```

[Member]

Member	Description
FISHEYE_VIEW_360_PANORAMA	360° panoramic correction mode
FISHEYE_VIEW_180_PANORAMA	180° panoramic correction mode
FISHEYE_VIEW_NORMAL	Normal correction mode
FISHEYE_NO_TRANSFORMATION	No correction. The source picture is output.

[Note]



The fisheye correction mode and the mount mode can be combined, and scaling is supported in each correction mode.

[See Also]

None

RECT_S

[Description]

Defines the attributes of the rectangular region.

[Syntax]

```
typedef struct hiRECT_S
{
    HI_S32 s32X;
    HI_S32 s32Y;
    HI_U32 u32Width;
    HI_U32 u32Height;
} RECT_S;
```

[Member]

Member	Description
s32X	Start horizontal coordinate of a region, Value range: [0, 4608-960].
s32Y	Start vertical coordinate of a region, Value range: [0, 3456-360].
u32Width	Region width, Value range: [960, 4608].
u32Height	Region height, Value range: [360, 3456].

[Note]

None

[See Also]

None

FISHEYE_REGION_ATTR_S

[Description]

Defines the configuration of attributes for each fisheye correction region.

[Syntax]

```
typedef struct hiFISHEYE_REGION_ATTR_S
{
    FISHEYE_VIEW_MODE_E    enViewMode;      /* fisheye view mode */
```



```
    HI_U32           u32InRadius;      /* inner radius of fisheye
correction region */
    HI_U32           u32OutRadius;     /* out radius of fisheye
correction region */
    HI_U32           u32Pan;          /* [0, 360] */
    HI_U32           u32Tilt;          /* [0, 360] */
    HI_U32           u32HorZoom;       /* [1, 4095] */
    HI_U32           u32VerZoom;       /* [1, 4095] */
    RECT_S           stOutRect;
}FISHEYE_REGION_ATTR_S;
```

[Member]

Member	Description
enViewMode	Correction mode for the correction region
u32InRadius	Inside radius of the source picture corresponding to the correction region, valid only in 360° panoramic correction mode. Value range: [0, u32OutRadius)
u32OutRadius	Outside radius of the source picture corresponding to the correction region in 360° panoramic correction mode, and visible radius of the correction region in other modes. Value range for Hi3519 V100: [1, 1.2 x max(1/2 width of the input picture, 1/2 height of the input picture)] Value range for Hi3519 V101: [1, 2304]
u32Pan	Value of the PTZ parameter Pan for the correction region Value range: [0, 360]
u32Tilt	Value of the PTZ parameter Tilt for the correction region Value range: [0, 360]
u32HorZoom	Value of the PTZ parameter ZoomH (horizontal zoom) for the correction region Value range: [1, 4095]
u32VerZoom	Value of the PTZ parameter ZoomV (vertical zoom) for the correction region Value range: [1, 4095]
stOutRect	Output position, width, and height of the correction region

[Note]

The fisheye subsystem can implement correction on multiple regions of a picture, and the attribute configuration of each region is independent.

[See Also]



None

FISHEYE_MAX_REGION_NUM

[Description]

Defines the maximum number of correction regions supported by the fisheye subsystem.

[Syntax]

```
#define FISHEYE_MAX_REGION_NUM      4
```

[Note]

None

[See Also]

[FISHEYE_ATTR_S](#)

FISHEYE_ATTR_S

[Description]

Defines the configuration related to the fisheye attribute.

[Syntax]

```
typedef struct hiFISHEYE_ATTR_S
{
    HI_BOOL          bEnable;           /* whether enable fisheye correction
or not */
    HI_BOOL          bLMF;              /* whether the LMF parameter of the
fisheye lens is from user config or from default linear config */
    HI_BOOL          bBgColor;         /* whether use background color or not */
    HI_U32           u32BgColor;        /* the background color RGB888 [0,
0xFFFFFFF] */
    HI_S32           s32HorOffset;       /* the horizontal offset between image
center and physical center of len */
    HI_S32           s32VerOffset;       /* the vertical offset between image
center and physical center of len */
    HI_U32           u32TrapezoidCoef;   /* strength coefficient of trapezoid
correction */

    HI_S32           s32FanStrength;     /* strength coefficient of fan
correction */

    FISHEYE_MOUNT_MODE_E   enMountMode;      /* fisheye mount mode */
    HI_U32           u32RegionNum;        /* fisheye correction region
number [1, FISHEYE_MAX_REGION_NUM] */
    FISHEYE_REGION_ATTR_S
    astFisheyeRegionAttr[FISHEYE_MAX_REGION_NUM];/* attribution of fisheye
correction region */
```



```
}FISHEYE_ATTR_S;
```

[Member]

Member	Description
bEnable	Parameter indicating whether to enable the fisheye correction function of an extended VI/VPSS channel 0: disabled 1: enabled
bLMF	Parameter indicating whether to use the LMF parameters of the fisheye lens configured by the user 0: not use 1: use
bBgColor	Parameter indicating whether to overlay the background color on the output picture 0: no 1: yes
u32BgColor	Background color in the RGB888 format Value range: [0, 0xFFFFFFFF]
s32HorOffset	Horizontal offset of the lens center point relative to the sensor center point, in pixel Value range: [-127, +127]
s32VerOffset	Vertical offset of the lens center point relative to the sensor center point, in pixel Value range: [-127, +127]
u32TrapezoidCoef	Strength coefficient for trapezoid correction. This member is used to correct the pitch angle in wall mount mode. Hi3519 V100 does not support this member. Value range: [0, 32]
s32FanStrength	Fan correction strength coefficient, valid only in 180° correction mode. Only Hi3519 V101 supports this member. Value range: [-760, +760]
enMountMode	Mount mode of the fisheye lens
u32RegionNum	Number of correction regions in a picture. At most four correction regions are supported.
astFisheyeRegionAttr	Attribute configuration for each correction region

[Note]

The fisheye subsystem can implement correction on multiple regions of a picture, and the correction mode and PTZ parameters of each region are independent from those of other



regions. The outputs of all correction regions are combined for output based on the configured output position, width, and height of each correction region.

[See Also]

None

SPREAD_ATTR_S

[Description]

Defines the configuration related to the broadening attribute.

[Syntax]

```
typedef struct hiSPREAD_ATTR_S
{
    HI_BOOL             bEnable;          /* whether enable fisheye
correction or not */
    HI_U32              u32SpreadCoef;     /* strength coefficient of
spread correction */
    SIZE_S               stDestSize;        /*dest size of spread*/
} SPREAD_ATTR_S;
```

[Member]

Member	Description
bEnable	Switch that determines whether to enable or disable the broadening function of the VI or VPSS channel
u32SpreadCoef	Broadening coefficient Value range: [0, 16]
stDestSize	Target picture width and height after broadening. The value range of u32Width is [640, 4608] and that of u32Height is [480, 3456].

[Note]

None

[See Also]

None

FISHEYE_CYLIND_ATTR_S

[Description]

Defines the configuration related to the cylindrical projection attribute.

[Syntax]

```
typedef struct hiFISHEYE_CYLIND_ATTR_S
{
```



```
    HI_S32 s32CenterXOffset;           /* Horizontal offset of the image
distortion center relative to image center.*/
    HI_S32 s32CenterYOffset;           /* Vertical offset of the image distortion
center relative to image center.*/
    HI_S32 s32Ratio;                  /* Distortion ratio.*/
} FISHEYE_CYLIND_ATTR_S;
```

[Member]

Member	Description
s32CenterXOffset	Horizontal offset of the lens center point relative to the sensor center point, in pixel Value range: [-127, +127]
s32CenterYOffset	Vertical offset of the lens center point relative to the sensor center point, in pixel Value range: [-127, +127]
s32Ratio	Degree of cylindrical projection correction Value range: [0, 500]

[Note]

None

[See Also]

None

FISHEYE_MIN_IN_IMAGE_WIDTH

[Description]

Defines the minimum width of the input image for fisheye correction.

[Syntax]

```
#define FISHEYE_MIN_IN_IMAGE_WIDTH      1920
```

[Note]

None

[See Also]

None

FISHEYE_MIN_IN_IMAGE_HEIGHT

[Description]

Defines the minimum height of the input image for fisheye correction.

[Syntax]



```
#define FISHEYE_MIN_IN_IMAGE_HEIGHT 1080
```

[Note]

None

[See Also]

None

FISHEYE_MIN_OUT_IMAGE_WIDTH

[Description]

Defines the minimum width of the output image for fisheye correction.

[Syntax]

```
#define FISHEYE_MIN_OUT_IMAGE_WIDTH 960
```

[Note]

None

[See Also]

None

FISHEYE_MIN_OUT_IMAGE_HEIGHT

[Description]

Defines the minimum height of the output image for fisheye correction.

[Syntax]

```
#define FISHEYE_MIN_OUT_IMAGE_HEIGHT 360
```

[Note]

None

[See Also]

None

LDC_MIN_IMAGE_WIDTH

[Description]

Defines the minimum width of the input/output image for LDC.

[Syntax]

```
#define LDC_MIN_IMAGE_WIDTH 640
```

[Note]

None

[See Also]



None

LDC_MIN_IMAGE_HEIGHT

[Description]

Defines the minimum height of the input/output image for LDC.

[Syntax]

```
#define LDC_MIN_IMAGE_HEIGHT      480
```

[Note]

None

[See Also]

None

FISHEYE_MAX_IMAGE_WIDTH

[Description]

Defines the maximum image width supported by the fisheye module.

[Syntax]

```
#define FISHEYE_MAX_IMAGE_WIDTH      4608
```

[Note]

None

[See Also]

None

FISHEYE_MAX_IMAGE_HEIGHT

[Description]

Defines the maximum image height supported by the fisheye module.

[Syntax]

```
#define FISHEYE_MAX_IMAGE_HEIGHT      3456
```

[Note]

None

[See Also]

None

FISHEYE_MODULE_PARAMS_S

[Description]

Sets fisheye module parameters in the sdk_init function of Huawei LiteOS.



[Syntax]

```
typedef struct hiFISHEYE_MODULE_PARAMS_S
{
    HI_U32 u32MaxFisheyeJob;
    HI_U32 u32MaxFisheyeTask;
    HI_U32 u32MaxFisheyeNode;
    HI_U32 u32WeightThreshold;
} FISHEYE_MODULE_PARAMS_S;
```

[Member]

Member	Description
u32MaxFisheyeJob	Maximum number of jobs for the fisheye subsystem Default value: 128 Value range: [20, 400]
u32MaxFisheyeTask	Maximum number of tasks for the fisheye subsystem Default value: 200 Value range: [20, 800]
u32MaxFisheyeNode	Maximum number of nodes for the fisheye subsystem Default value: 200 Value range: [20, 800]
u32WeightThreshold	Weight threshold of the fisheye subsystem Default value: 6 Value range: [1, 100]

[Note]

This data structure is described in **hi_module_param.h**. When Huawei LiteOS is used, this data structure can be used to set fisheye module parameters in the initialization function. This data structure applies only to the Huawei LiteOS version, and is not contained in the Linux version.

[See Also]

[fisheye_mod_init](#)

11.6 Error Codes

Table 11-12 describes the error codes of the fisheye subsystem.



Table 11-12 Error codes of the fisheye subsystem

Error Code	Macro Definition	Description
0xA033800D	HI_ERR_FISHEYE_NOBUF	The memory fails to be allocated.
0xA033800E	HI_ERR_FISHEYE_BUF_EMPTY	The jobs, tasks, or nodes of the fisheye subsystem are used up.
0xA0338006	HI_ERR_FISHEYE_NULL_PTR	The pointer of the input parameter is null.
0xA0338003	HI_ERR_FISHEYE_ILLEGAL_PARAM	The configuration of fisheye parameters is invalid.
0xA0338010	HI_ERR_FISHEYE_SYS_NOTREADY	The system is not initialized.
0xA0338008	HI_ERR_FISHEYE_NOT_SUPPORT	This operation is not supported.
0xA0338009	HI_ERR_FISHEYE_NOT_PERMITTED	This operation is not allowed.



Contents

12 Proc Debugging Information.....	12-1
12.1 Overview.....	12-1
12.2 SYS	12-2
12.3 VB	12-4
12.4 LOG	12-6
12.5 CHNL.....	12-8
12.6 VGS.....	12-10
12.7 H264E	12-14
12.8 JPEGGE	12-20
12.9 RC	12-23
12.10 Region	12-29
12.11 VENC	12-36
12.12 VI	12-42
12.13 VO.....	12-58
12.14 VPSS	12-67
12.15 VDA	12-83
12.16 AI.....	12-87
12.17 AO	12-101
12.18 AENC	12-108
12.19 ADEC	12-110
12.20 H265E	12-110
12.21 ACODEC.....	12-116



12 Proc Debugging Information

12.1 Overview

The debugging information is obtained from the proc file system of Linux. The information reflects the status of the current system and can be used to locate and analyze problems.

[Directory]

/proc/umap

[Checklist]

File	Description
sys	Records the current status of the SYS module.
vb	Records the current status of the video buffer (VB).
logmpp	Records the debugging level of each module (for internal debugging).
chnl	Records the status of the channel (CHNL) module.
vgs	Records the status of the dedicated scaling unit (DSU).
h265e	Records the encoding attributes, status, and history statistics of each channel during H.265 encoding.
h264e	Records the encoding attributes, status, and history statistics of each channel during H.264 encoding.
jpege	Records the encoding attributes, status, and history statistics of each channel during JPEG encoding.
rc	Records the stream control attributes, status, and history statistics of each encoding channel.
rgn	Records the region management information about the video overlay on-screen display (OSD).
venc	Records the information about the video encoder.
vdec	Records the information about the video decoder.
vi	Records the information about the video input unit (VIU).



File	Description
vo	Records the information about the video output unit (VOU).
vpss	Records the information about the video pre-processing module.
ive	Records the status of the operators of the intelligent video engine (IVE).
vda	Records the status and attributes of the channel whose video detection analysis (VDA) function is enabled.
ai	Records the information about audio input (AI) channels.
ao	Records the information about audio output (AO) channels.
aenc	Records the audio encoding (AENC) information.
adec	Records the audio decoding (ADEC) information.
mpeg4e	Records the encoding attributes, status, and history statistics of each channel during MPEG4 encoding.
acodec	Records the audio CODEC volume information.

[View Methods]

- You can run the **cat** command such as **cat /proc/umap/venc** on the console. You can also run file operation commands such as **cp /proc/umap/ ./ -rf** to copy all the proc files under umap to the current directory.
- You can read the preceding files as common read-only files through applications such as **fopen** and **fread**.

NOTE

Note the following when reading the parameter descriptions:

- For the parameter whose value is **0** or **1**, if the mapping between the values and the definitions is not specified, the value 1 indicates affirmative and the value 0 indicates negative.
- For the parameter whose value is **aaa**, **bbb**, or **ccc**, if the mapping between the values and the definitions is not specified, you can identify the parameter definitions based on **aaa**, **bbb**, or **ccc**.

12.2 SYS

[Debugging Information]

```
# cat /proc/umap/sys
```

```
[SYS] Version: [Hi3516A_MPP_V1.0.0.0 Debug], Build Time [Dec 20 2011, 10:40:10]
System State: 0 (0: initialized; 1: exiting; 2: exited)
-----MODULE PARAM-----
vi_vpss_online    sensor
          0      mn34220
-----SCALE COEFF INFO-----
RangeLevel   RangeValue   HorLum   HorChr   VerLum   VerChr
```



```
RANGE_0      (0,     1/4)    LEVEL_0    LEVEL_0    LEVEL_0    LEVEL_0
RANGE_1      [1/4,   1/3)    LEVEL_1    LEVEL_0    LEVEL_1    LEVEL_0
RANGE_2      [1/3,   1/2)    LEVEL_2    LEVEL_1    LEVEL_2    LEVEL_1
RANGE_3      [1/2,   3/4)    LEVEL_3    LEVEL_2    LEVEL_3    LEVEL_2
RANGE_4      [3/4,     1)    LEVEL_3    LEVEL_2    LEVEL_4    LEVEL_2
RANGE_5      [1,       1]    LEVEL_4    LEVEL_3    LEVEL_5    LEVEL_3
RANGE_6      (1,     MAX)   LEVEL_5    LEVEL_2    LEVEL_6    LEVEL_2
-----MEM TABLE-----
MOD          MODNAME  DEV CHN          MMZNAME
5            grp      1   0             ddr1
5            grp      3   0             ddr1
5            grp      5   0             ddr1
5            grp      7   0             ddr1
-----BIND RELATION TABLE-----
FirMod FirDev FirChn SecMod SecDev SecChn TirMod TirDev TirChn SendCnt
rstCnt
vpss     0       2       vo       0       0       null     0       0       0       0       0
vi       0       0       vpss     0       2       vo       0       0       0       448     0

```

[Analysis]

This section records the current status of the SYS module.

[Parameter Description]

Parameter	Description	
System State	initialized	Initialized status
	exiting	Exiting status
	exited	Exited status
MODULE PARAM	vi_vpss_online	Module parameter used to control the online and offline modes (0: offline; 1: online). The default mode in the online mode.
	sensor	Sensor being used by the system. This parameter is transferred when the script is loaded.
SCALE COEFF INFO	RangeLevel	Enumeration of the scaling ratio range
	RangeValue	Specific scaling ratio range
	HorLum	Scaling coefficient level of horizontal luminance
	HorChr	Scaling coefficient level of horizontal chrominance
	VerLum	Scaling coefficient level of vertical luminance
	VerChr	Scaling coefficient level of vertical chrominance
MEMTABLE	MOD	Module ID



Parameter	Description	
BIND RELATION TABLE	MODNAME	Module name
	DEV	Device ID corresponding to a module
	CHN	Channel ID corresponding to a module
	MMZNAME	Media memory zone (MMZ) name. Typically, the MMZ name is the DDR name. If the DDR name is NULL, the MMZ name is not displayed.
BIND RELATION TABLE	FirMod	Level-1 module name in the settings of the binding relationship. Data is transferred from level 1 to level 2.
	FirDev	Level-1 device name in the settings of the binding relationship. Data is transferred from level 1 to level 2.
	FirChn	Level-1 channel name in the settings of the binding relationship. Data is transferred from level 1 to level 2.
	SecMod	Level-2 module name in the settings of the binding relationship. Data is transferred from level 1 to level 2.
	SecDev	Level-2 device name in the settings of the binding relationship. Data is transferred from level 1 to level 2.
	SecChn	Level-2 channel name in the settings of the binding relationship. Data is transferred from level 1 to level 2.
	TirMod	Level-3 module name in the settings of the binding relationship. If there is level-3 binding relationship, data is transferred from level 2 to level 3. If there is no level-3 binding relationship, this parameter is null.
	TirDev	Level-3 device name in the settings of the binding relationship
	TirChn	Level-3 channel name in the settings of the binding relationship
	SendCnt	Amount of data transferred from level 1 to level 2. The data amount is in the unit of frame. The amount of data transferred from the VPSS to the VOU is always 0, because SYS does not count such data. You need to view the data amount from the proc information about each module.
	rstCnt	Number of times that level 1 resets level 2

12.3 VB

[Debugging Information]

```
# cat /proc/umap/vb
```



```
[VB] Version: [Hi3516A_MPP_V1.0.0.0 B010 Release], Build Time[Oct 29 2013,  
14:28:26]  
-----VB PUB CONFIG-----  
Max Count of Pools: 256  
-----VB SUPPLEMENT ATTR-----  
Supplement Config: 0  
Supplement Size: 0  
Vb Total Cnt: 40  
-----COMMON POOL CONFIG-----  
PoolId 0  
Size 3135488  
Count 8  
-----MODULE COMMON POOL CONFIG of VB_UID <4> -----  
PoolId 0 1  
Size 3159552 522240  
Count 20 0  
-----  
PoolId PhysAddr VirtAddr IsComm Owner BlkSz BlkCnt Free  
MinFree  
0 0x8518c000 0xc9000000 1 -1 3135488 8 5(5) 4  
BLK VIU VOU VGS VENC VDEC VDA H264E JPEG E MPEGE H264D JPEG D MPEGD  
VPSS GRP MPI PCIV AI AENC RC VFMW USER H265E FISHEYE  
5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0  
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0  
6 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1  
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  
Sum 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2  
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

[Analysis]

This section records the current status of VB.

[Parameter Description]

Parameter	Description	
VB PUB CONFIG	Max Count of Pools	Maximum number of buffer pools
VB SUPPLEMENT ATTR	Supplement Config	Configuration of the supplementary information about video frames
	Supplement Size	Memory space occupied by the supplementary information about video frames
	Vb Total Cnt	Total number of VBs in the public and private VB pools
COMMON	PoolId	Handle of a public buffer pool



Parameter		Description
POOL CONFIG	Size	Size of the block in a buffer pool
	Count	Number of blocks in a buffer pool
MODULE COMMON POOL CONFIG of VB_UID	PoolId	Handle of a public buffer pool
	Size	Size of the block in a buffer pool
	Count	Number of blocks in a buffer pool
NULL (it indicates the DDR that is not named.)	PoolId	Handle of a public or private buffer pool
	PhysAddr	Start physical address for a public or private buffer pool
	VirtAddr	Start virtual address for a public or private buffer pool
	IsComm	Whether the pool is a public buffer pool Value: {0, 1}
	Owner	Buffer pool owner -2: private pool -1: public pool ≥ 0 : module VB
	BlkSz	Size of the buffer in a buffer pool
	BlkCnt	Number of buffers in a buffer pool
	Free	Number of free buffers in a buffer pool
	MinFree	Minimum number of free buffers since the program runs. If the minimum number is 0, some frames may be lost because the buffers are insufficient.
	BLK	Handle of the buffer in a buffer pool
VIU/VOU/VGS/VERN/NC/VDEC/VDA/H264E/JPEG/E/MPEG/E/H264D/JPEG/D/MPEG/D/VPSS/GΡ/ΜΡΙ/PCIV/AΙ/AΕΝC/RC/VFMW/USER/H265E/FISHEYΕ	Module name	The value indicates the number of times that a buffer in the buffer pool is occupied by the current module. 0: not occupied Other values: number of times that a buffer is occupied

12.4 LOG

[Debugging Information]



```
# cat /proc/umap/logmpp
-----LOG BUFFER
STATE-----
MaxLen  ReadPos  WritePos  ButtPos
64 (KB)      0        0    65536

-----CURRENT LOG
LEVEL-----
vb      : 3
sys     : 3
region  : 3
chnl    : 3
vpss    : 3
venc    : 3
h264e   : 3
jpege   : 3
vou     : 3
viu     : 3
rc      : 3
aio     : 3
ai      : 3
ao      : 3
aenc    : 3
adec    : 3
isp     : 3
ive     : 3
tde     : 3
vgs     : 3
h265e   : 3
fisheye : 3
```

[Analysis]

This section records the debugging level of each module (for internal debugging).

The **cat /proc/umap/logmpp** command is used to obtain the logmpp-level information.

[Parameter Description]

Parameter	Description	
LOG BUFFER STATE	MaxLen	Log buffer size (in KB)
	ReadPos	Read pointer position
	WritePos	Write pointer position
	ButtPos	Buffer bottom position
CURRENT	vb/sys/region/ch	Module name. The digit at the end of the parameter



Parameter	Description
LOG LEVEL nl/vpss/venc/h2 64e/jpege/vou/v iu/rc/aio/ai/ao/a enc/adec/isp/ive /tde/vgs/h265e/f ishey	indicates the log display level.

12.5 CHNL

[Debugging Information]

```
# cat /proc/umap/chnl

[CHNL] Version: [Hi3516A_MPP_V1.0.0.0 Debug], Build Time [Sep 2 2011,
15:36:02]

Total Chnl Scheduler Count: 1
-----SCHEDULER 0 INFO-----
SchdlId VpuNum
    0      1
-----VPU INFO-----
VpuId  Name   State   IntCnt  TimeCnt  VpuCnt  ErrCnt  InqCnt  StartOk
StartNo Config Reset
0     VEUD_0  RUN     5252     312     4940     0       7735     4941     0
233          0
-----CHNL STATE-----
ChnlId Prio  Type   TskNum  State   InqCnt  StartOk StartNo  IntPro
0      0     H264E    6      IDLE    304     155      0       4940
-----CHNL PERF-----
ChnlId Type   StartCost IntCost IntCostL HWCost  HWCycle HWCostL HWCostAcc
0      H264E  2829     4354     6352    22234     0      25635     3
-----CHNL CURRENT RUN STATE-----
VpuId  VpuName ChnlId Type
    0     VEUD_0   27     H264E
```

[Analysis]

This section records the current status of the CHNL module.

[Parameter Description]

Parameter	Description
SCHEDULER 0	SchdlId



Parameter		Description
INFO	VpuNum	Number of video processing units (VPUs) managed by the scheduler
VPU INFO	VpuId	VPU ID
	Name	VPU name
	State	Running status of a VPU
	IntCnt	Number of times that the VPU is interrupted by the software and hardware
	TmCnt	Number of times that the VPU is interrupted by the software
	VpuCnt	Number of times that the VPU is interrupted by the hardware
	ErrCnt	Number of times that the VPU reports incorrect interrupt information
	InqCnt	Number of times that the VPU query tasks
	StartOk	Number of times that the VPU starts successfully
	StartNo	Number of times that the VPU fails to start
CHNL STATE	Config	Number of times that the VPU is successfully configured
	Reset	Number of times that the VPU crashes
	ChnlId	ID of the channel that is registered in the current scheduler
	Prio	Channel priority
	Type	Channel type
	TskNum	Number of tasks in the current channel
	State	Status of the current task
	InqCnt	Number of times that the current task is queried
	StartOk	Number of times that the current task is started
CHNL PERF	StartNo	Number of times that the current task fails to start
	IntPro	Number of times that the interrupt processing function processes the current task
	ChnlId	ID of the channel that is registered in the current scheduler
	Type	Channel type
	StartCost	Time spent in starting the current task (in μ s)



Parameter	Description
CHNL CURRENT RUN STATE	IntCost Time spent by the interrupt processing function for processing the current task (in μ s)
	IntCostL Maximum time spent by the interrupt processing function for processing the current task (in μ s)
	HWCost Time that the chip spent in processing the current task (in μ s)
	HWCycle Number of cycles for the current task
	HWCostL Maximum time that the chip spent in processing a task slice in the currently scheduled channel (in μ s)
	HWCostAcc Number of times that the chip spent more than 30 ms in processing a task slice in the currently scheduled channel
CHNL CURRENT RUN STATE	VpuId VPU ID
	VpuName VPU name
	ChnlId ID of the channel that is registered in the current scheduler
	Type Channel type

12.6 VGS

[Debugging Information]

```
# cat /proc/umap/vgs
[VGS] Version: [Hi3516A_MPP_V1.0.0.0 B000 Debug], Build Time[Aug 10 2013,
15:47:22]
-----MODULE PARAM-----
max_job_num    max_task_num    max_node_num
          128           200           200
-----RECENT JOB INFO-----
SeqNo ModName JobHdl TaskNum   State  InSize OutSize CostTime HwTime
  0    VOU      52       1     Proced 2073600  414720  53773  39854
  1    VOU      51       1     Proced 2073600  414720  53550  39847
  2    VOU      50       1     Proced 2073600  414720  53695  39846
-----MAX WASTE TIME JOB INFO-----
SeqNo ModName JobHdl TaskNum   State  InSize OutSize CostTime HwTime
  0    VOU      23       1     Proced 2073600  414720  57442  39936
-----VGS JOB STATUS-----
Success      Fail      Cancel  AllJobNum   FreeNum  BeginNum  BusyNum
ProcingNum
```



```
53      0      0      128      128      0      0
0

-----VGS TASK STATUS-----
Success    Fail    Cancel AllTaskNum   FreeNum   BusyNum
53        0       0     200         200        0

-----VGS NODE STATUS-----
AllNodeNum  BusyNum  MinFree  MaxInJob  SubmitFail  IntFail
200        0       196      2          0          0

-----VGS INT STATUS-----
IntNum     IntTm   HalProcTm
53        0       0

-----VGS MEM REQ STATUS -----
ReqOk     FreeOk  ReqFail  FreeFail
0         0       0       0
```

[Analysis]

This section records the information about the VGS, including the number of recently completed jobs, the most time-consuming job, history statistics, and interrupt information.

[Parameter Description]

Parameter	Description	
MODULE PARAM	max_job_num	Maximum number of jobs. The default value is 128. This parameter value needs to be set during driver loading only when you want to change the number of jobs supported by the VGS.
	max_task_num	Maximum number of tasks. The default value is 200. You need to specify the number during driver loading only when you want to adjust the tasks supported by the VGS.
	max_node_num	Maximum number of nodes. The default value is 200. You need to specify the number during driver loading only when you want to adjust the nodes supported by the VGS.
RECENT JOB INFO	SeqNo	Print sequence Value range: [0, 7]
	ModName	Name of the module that submits a job
	JobHdl	Handle ID of a job, for debugging Value range: [0, 127]
	TaskNum	Number of tasks in a job Value range: [0, 200]
	State	Status of a job Value range: {ProcErr, Proced, Procing} ProcErr: The job fails to be handled.



Parameter	Description
	Proced: The job is handled successfully. Procing: The job is being handled by the hardware.
InSize	Total size of the input pictures involved in all the tasks of a job The size is expressed by pixel. Each time a task is added to the job, the value of InSize increases by adding the size of the input pictures of the task. The scaling performance of the VGS depends on the total sizes of the input pictures and output pictures (InSize/OutSize). In general, the larger the InSize and OutSize of a job, the longer (CostTime) the job is handled.
OutSize	Total size of the output pictures involved in all the tasks of a job The size is expressed by pixel. Each time a task is added to the job, the value of OutSize increases by adding the size of the output pictures of the task.
CostTime	Period spent in submitting and handling a job The period includes the processing time (in μ s) of the software, hardware, and interrupt service routine (ISR) relevant to the job. If the VGS performance is insufficient (for example, too many scaling tasks are submitted or the performance limit is exceeded), the period exceeds the expected value. For example, in the PAL preview scenario, if a job is handled more than 40 ms, the previewed pictures are paused intermittently. In this case, you can check whether the VGS performance limit is exceeded.
HwTime	Time period during which the job is processed in hardware Unit: μ s HwTime indicates the hardware processing time, and is usually less than CostTime .
MAX WASTE TIME JOB INFO	The items are the same as those of RECENT JOB INFO. Information of the most time-consuming job among 500 jobs The members are the same as those of RECENT JOB INFO. For details, see the preceding description. When a job consumes more time or the number of jobs is greater than 500, the value of this item is updated. The value reflects the recent performance of the VGS and indicates whether some jobs are not handled in a timely manner by the VGS.
VGS JOB STATUS	Success Number of jobs that are handled successfully The number is incremented by 1 when a job is successfully handled by the hardware.



Parameter		Description
	Fail	Number of jobs that fail to be handled The number is incremented by 1 when the VGS fails to submit a job to the driver layer. If the number increases, you can view the log to learn the failure causes.
	Cancel	Number of jobs that are proactively canceled by users The number is incremented by 1 each time users call the cancelJob interface. If a task fails to be added to a job, users call the cancelJob interface. If the number increases, you can view the log to learn the failure causes.
	AllJobNum	Number of VGS job nodes. This number is consistent with the value of max_job_num and is typically set to 128.
	FreeNum	Number of free job nodes
	BeginNum	Number of jobs that are created but not submitted (EndJob interface)
	BusyNum	Number of jobs that are submitted (EndJob interface) but not transferred to the hardware
	ProcingNum	Number of jobs that are being handled by the hardware
VGS TASK STATUS	Success	Number of tasks that are handled successfully A job consists of one or more scaling tasks. If a job is handled successfully, its tasks are all handled successfully. Therefore, this number increases faster than the Success item of the job. When a job is successfully handled by the hardware, the number increases by adding the tasks of the job.
	Fail	Number of tasks that fail to be handled If a job fails to be handled, its tasks all fail to be handled. When a job fails to be handled, the number increases by adding the number of failed tasks. If the number increases, you can view the log to learn the failure causes.
	Cancel	Number of tasks that are proactively canceled by users When users call the cancelJob interface to cancel a job, all the tasks of the job are canceled and the number increases. If the number increases, you can view the log to learn the failure causes.
	AllTaskNum	Number of VGS task nodes. This number is typically set to 200
	FreeNum	Number of free task nodes
	BusyNum	Number of tasks that are added to a job
	VGS	AllNodeNum Number of task nodes



Parameter		Description
NODE STATUS	BusyNum	Number of occupied nodes
	MinFree	Minimum number of idle nodes
	MaxInJob	Maximum number of nodes in a job
	SubmitFail	Number of times that a task fails to be submitted due to insufficient nodes
	IntFail	Number of times that a task fails to be submitted in the interrupt handling function due to insufficient nodes
VGS INT STATUS	IntNum	Number of VGS interrupts This number is incremented by 1 when a job is handled by the hardware.
	IntTm	Time (in μ s) of processing a VGS interrupt The time includes the time of processing a VGS interrupt and calling a callback interface.
	HalProcTm	Time (in μ s) of submitting a job to the driver layer, for internal debugging
VGS MEM REQ STATUS	ReqOk	Times of applying for VBs successfully The number is incremented by 1 each time a VB is applied successfully.
	FreeOk	Number of times that VBs are released successfully The number is incremented by 1 each time a VB is released successfully. The number of ReqOk must be consistent with the value of FreeOk. This indicates that the memory is not leaked.
	ReqFail	Number of times that VBs fail to be applied The number is 0 in general and is incremented by 1 each time a VB fails to be applied.
	FreeFail	Number of times that VBs fail to be released The number is 0 in general and is incremented by 1 each time a VB fails to be released.

12.7 H264E

NOTE

This section is not supported by Hi3519 V100

[Debugging Information]

```
# cat /proc/umap/h264e
[H264E] Version: [Hi3516A_MPP_V1.0.0.0 Debug], Build Time [Sep 6 2011,
```



```
09:51:15]
-----MODULE PARAM-----
OnePack      H264eVBSource      H264eRcnEqualRef      H264eMiniBufMode
          0              1                  0                  0
-----CHN ATTR-----
ID      MaxWidth   MaxHeight   Width    Height   profile   C2GEN   BufSize
ByFrame  MaxStrCnt
0       720        576        720      576      hp        Y        829440      Y
200
-----PICTURE INFO-----
ID      EncdStart   EncdSucceed   Lost     Disc     Skip     Pskip    Recode   RlsStr
UnrdStr
0       29           29           0         0         0         0         0         29         0
-----STREAM BUFFER-----
ID      Base        RdTail       RdHead       WrTail       WrHead       DataLen
BufFree
0       0xd8300000  0x6de80      0x6de80      0x6de80      0x6de80      0
829376
-----RefParam INFO-----
ID      EnPred     Base        Enhance   bVirtualIEnable  VirtualIInterval
VirtualIQpDelta GetVbFail
0       Y           1           0           N                   30           0
0
-----ROI INFO-----
ID      Index      bAbsQp      Qp        Width     Height   StartX   StartY   BgSrcFr
BgTarFr
0       0           N           -10       256       256       0           0           30           15
-----Syntax INFO1-----
ID      SlcspltEn   Slcmode      Slcsize      IntraRefresh   enIslice
RefreshLine  QpOfIDR
0       Y           ByteNum     10000          N                   N           17
51
-----Inter & Intra prediction INFO-----
ID      Profile    HWsize     VWsize    P16x16    P16x8     P8x16    P8x8     MvExt    I16x16
Inxn    Ipcm
0       hp          3           2           Y           Y           Y           Y           Y           Y           Y
-----Syntax INFO2-----
ID      Profile    EntrpyI   EntrpyP   Itrans    Ptrans    QMatrix   POC      DblkIdc   Alpha
Beta
0       hp          cavlc     cabac     all       all       N           2           0           5
5
[Analysis]
```



During multi-channel H.264 encoding, the upper-layer software may not fetch streams in a timely manner. As a result, the stream buffer is insufficient, which causes frame loss. The best way of checking whether frames are lost is to view the **Disc** and **UnrdStr** parameters. **Disc** indicates the number of discarded frames due to stream buffer insufficiency. **UnrdStr** indicates the number of remaining frames in the stream buffer. A larger value indicates streams are fetched in a less timely manner.

[Parameter Description]

Parameter		Description
MODULE PARAM	OnePack	Mode in which streams are obtained 0: Streams are obtained in multiple-packet mode. 1: Streams are obtained in single-packet mode.
	H264eVBSource	Mode in which the VB is obtained for the reference frame and reconstruction frame 1: The private VBs are used. 2: The VBs are allocated by the user.
	H264eRcnEqual Ref	Whether the reference frame shares the buffer for storing the luminance data with the reconstruction frame
	H264eMiniBufMode	Configuration mode of the H.264 encoding stream buffer 0: normal mode 1: memory reduction mode
CHN ATTR	ID	Channel ID
	MaxWidth	Maximum width of the encoding channel (in pixel)
	MaxHeight	Maximum height of the encoding channel (in pixel)
	Width	Width (in pixel)
	Height	Height (in pixel)
	profile	Encoding channel profile Base: baseline Mp: main profile Hp: high profile
	C2GEn	Color-to-gray enable
	BufSize	Stream buffer size (in byte)
	ByFrame	Whether to obtain streams by frame
	MaxStrCnt	Maximum number of frames in the stream buffer Default value: 200
PICTURE INFO	ID	Channel ID
	EncdStart	Number of received pictures



Parameter	Description	
	This value is incremented by 1 after the pictures to be encoded are received.	
EncdSucceed	Number of frames that are encoded successfully	
Lost	Number of frames that are discarded during encoding. The discarded frames include: <ul style="list-style-type: none">• Frames that are discarded due to encoder exceptions or jumbo frames. That is, the number of frames is the value of Disc.• Frames that are discarded for controlling the frame rate. The frame rate is controlled by the rate controller (RC). That is, the number is the value of Skip.• Frames that are discarded when the configured transient bit rate exceeds the threshold	
Disc	Number of frames discarded for the following reasons: <ul style="list-style-type: none">• Jumbo frames occur.• An exception occurs in the encoder. For example, the encoder crashes.• The frame rate is controlled or the transient bit rate is above the threshold.	
Skip	Number of frames that are discarded for controlling the frame rate or discarded when the transient bit rate is above the threshold during encoding.	
Pskip	Number of frames that are encoded as Pskip frames	
Recode	Number of frames that are re-encoded by the encoder	
UnrdStr	Number of frames that are not obtained by users	
RlsStr	Number of frames that are obtained and released by users	
STREAM BUFFER	ID	Channel ID
	base	Base address for a stream buffer
	RdTail	Read tail pointer
	RdHead	Read head pointer
	WrTail	Write tail pointer
	WrHead	Write head pointer
	datalen	Data length
	buffree	Free buffer size
RefParam INFO	ID	Channel ID
	EnPred	Whether some frames at the base layer are referenced by other frames at the base layer



Parameter	Description	
	Base	Base layer period
	Enhance	Enhance layer period
	bVirtualIEnable	Whether to insert virtual I frames
	u32VirtualIInterval	Interval of virtual I frames
	s32VirtualIQpDelta	QpDelta of the virtual I frame relative to the common P frame
	GetVbFail	Number of times that the VB of reconstruction frames fails to be obtained
ROI INFO	ID	Channel ID
	Index	Region of interest (ROI) index
	bAbsQp	Whether the ROI uses the absolute quantizer parameter (QP) mode
	Qp	QP value configured by the ROI
	Width	ROI width (in pixel)
	Height	ROI height (in pixel)
	StartX	Start horizontal coordinate of the ROI (in pixel)
	StartY	Start vertical coordinate of the ROI (in pixel)
	BgSrcFr	Source frame rate of a non-ROI
	BgTarFr	Target frame rate of a non-ROI
Syntax INFO	ID	Channel ID
	SlcsplEn	Slice split enable
	Slcmode	Slice split mode <ul style="list-style-type: none">• ByteNum: split by byte• MbLine: split by macroblock
	Slcsize	Size of each slice When the slice is split by byte, it indicates the number of bytes of each slice. When the slice is split by macroblock, it indicates the number of macroblock lines of each slice.
	IntraRefresh	Enable for refreshing the I macroblock in the P frame
	enIslice	Enable for converting the first refreshed I macroblock into Islice 0: disabled 1: enabled



Parameter		Description
	RefreshLine	Number of refreshed rows in the I macroblock for each frame
	QpOfIDR	QP value of the IDR frame when IDR frame is requested
Inter & Intra prediction INFO	ID	Channel ID
	Profile	Encoding channel profile type <ul style="list-style-type: none">• Base: baseline• MP: main profile• HP: high profile
	HWsize	Size of the horizontal search window
	VWsize	Size of the vertical search window
	P16x16	Inter16x16 prediction mode enable
	P16x8	Inter16x8 prediction mode enable
	P8x16	Inter8x16 prediction mode enable
	P8x8	Inter8x8 prediction mode enable
	MvExt	Whether to extend the boundaries when search reaches the boundaries of the pictures
	I16x16	Intra16x16 prediction mode enable
Syntax INFO	Inxn	Intra8x8 and intra4x4 prediction mode enable
	Ipcm	Ipcm prediction mode enable
	ID	Channel ID
	Profile	Encoding channel profile type <ul style="list-style-type: none">• Base: baseline• MP: main profile• HP: high profile
	EntrpyI	Entropy encoding mode of the I frame, context-based adaptive binary arithmetic coding (CABAC) or context-based adaptive variable-length coding (CAVLC)
	EntrpyP	Entropy encoding mode of the P frame, CABAC or CAVLC
	Itrans	Transform mode used by I frame <ul style="list-style-type: none">• All: The trans8x8 mode and trans4x4 mode are enabled simultaneously.• 4x4: The trans4x4 mode is enabled.• 8x8: The trans8x8 mode is enabled.



Parameter	Description
Ptrans	Transform mode used by P frame <ul style="list-style-type: none">• All: The trans8x8 mode and trans4x4 mode are enabled simultaneously.• 4x4: The trans4x4 mode is enabled.• 8x8: The trans8x8 mode is enabled.
	QMatrix
	POC
	DblkIdc
	Alpha
	Beta

12.8 JPEGE

NOTE

This section is not supported by Hi3519 V100

[Debugging Information]

```
[JPEGE] Version: [Hi3516A_MPP_V1.0.0.0 Debug], Build Time[Sep 6 2011,  
09:51:15]
```

```
-----MODULE PARAM-----
```

OnePack	JpegeMiniBufMode
0	0

```
-----ATTRIBUTE1-----
```

ID	bMjpeg	PicType	MaxWidth	MaxHeight	Width	Height	BufSize
ByFrm	MCU	Qfactor	C2Gen	DcfEn			
12	Y	YUV422	32	32	32	32	2048
0	90	Y	N				Y

```
-----ATTRIBUTE2-----
```

ID	ThumbPicW	ThumbPicH	ThumbPicN
12	160	120	1

```
-----STATUS1-----
```

ID	BufLen	FreeLen	StrmCnt	MaxStrm
12	829340	829376	0	100000

```
-----STATUS2-----
```

ID	PicRec	PicCoded	PicDropped	PicDisc	NoStmCnt	RcFail	PicRecode
UnrdStr							
12	189	8	181	0	0	0	0



```
0
-----STREAM BUFFER-----
ID      Base      RdTail  RdHead  WrTail  WrHead  BufLen  DataLen
BufFree
12      0xc3400000    12800   12800   12800   12800   829440    0     829376
```

[Analysis]

This section records the encoding attributes, status, and history statistics of each channel during JPEG encoding. A maximum of 64 encoding channels are supported. The information is used to locate problems such as system blocking and frame loss.

[Parameter Description]

Parameter		Description
MODULE PARAM	OnePack	Mode in which streams are obtained 0: Streams are obtained in multiple-packet mode. 1: Streams are obtained in single-packet mode.
	JpegeMiniBufMode	Configuration mode of the JPEG encoding stream buffer 0: normal mode 1: memory reduction mode
ATTRIBUTE1	ID	Channel ID
	bMjpeg	MJPEG encoding N: JPEG snapshot Y: MJPEG encoding
	PicType	Picture type: YUV422, YUV420 or YUV400
	MaxWidth	Maximum width of the encoding channel (in pixel)
	MaxHeight	Maximum height of the encoding channel (in pixel)
	Width	Picture width
	Height	Picture height
	BufSize	Stream buffer size (in byte)
	MCU	Number of minimum coded units (MCUs) in each embedded CPU subsystem (ECS)
	ByFrm	Whether to obtain streams by frame
ATTRIBUTE2	Qfactor	Channel Qfactor
	C2GEn	Color-to-gray enable
	DcfEn	Whether the JPEG picture has a thumbnail
ATTRIBUTE2	ID	Channel ID
	ThumbPicW	Thumbnail width



Parameter		Description
	ThumbPicH	Thumbnail height
	ThumbPicN	Number of thumbnails
Status1	ID	Channel ID
	BufLen	Total length of the stream buffer
	FreeLen	Length of the free stream buffer
	StrmCnt	Number of frames in the current stream buffer
	MaxStrm	Maximum number of frames in the stream buffer Default value: 100,000
Status2	ID	Channel ID
	PicRec	Number of pictures that are transmitted for encoding
	PicCoded	Number of frames that are successfully encoded
	PicDropped	Total number of discarded pictures before encoding, including the discarded pictures of Nostmcnt , and RcFail
	PicDisc	Number of frames discarded for the following reasons: <ul style="list-style-type: none">• Jumbo frames occur.• The encoder is abnormal, for example, the system is crashed.• The encoder detects stream buffer insufficiency during encoding.
	Nostmcnt	Number of times that frames are discarded because the number of frames in the stream buffer exceeds upper limit
	RcFail	Number of times that frames are discarded when the frame rate is controlled or the transient bit rate is above the threshold
	PicRecode	Number of jumbo frames that are encoded again
	UnrdStr	Number of frames that are not obtained by users
STREAM BUFFER	ID	Channel ID
	base	Base address for a stream buffer
	RdTail	Read tail pointer
	RdHead	Read head pointer
	WrTail	Write tail pointer
	WrHead	Write head pointer



Parameter		Description
	Buflen	Stream buffer length
	datalen	Data length
	buffree	Free buffer size

[Note]

The **ATTRIBUTE2** attributes are displayed only when the system supports the thumbnail.

12.9 RC

NOTE

This section is not supported by Hi3519 V100

[Debugging Information]

[RC] Version: [Hi3516A_MPP_V1.0.0.0 B010 Debug], Build Time[Aug 4 2016, 16:11:35]

```
----- BASE PARAMS -----
ID      Gop   StatTm  ViFr   TrgFr  ProType  RcMode   Br(kbps)  FluLev  IQp    PQp
0       30     2       30     30/0    265      CBR       2048      0        N/A    N/A

-----RUN COMM PARAM 1-----
ChnId  RowQpDelta                      ThrdI(12)
0           1  [3  3  5  5  8  8  8  15 20 20 25 25]

-----RUN COMM PARAM 2-----
ChnId  FirstFrmStartQP                  ThrdP(12)
0           -1  [3  3  5  5  8  8  8  15 20 20 225 225]

-----RUN COMM PARAM 3-----
ChnId  bLost    LostThr    LostFrmStr  EncGap   RCPriority  EnIDR
SprFrmMod  SprIFrm   SprPFrm   SprBFrm
0         N        83886080  NORMAL    0        BITRATE     Y
None        500000    500000    500000

-----RUN CBR PARAM1 -----
ID      MinIprop  MaxIprop  MaxQp   MinQp  MaxIQp  MinIQp  IPQpDelta  QLevel
MaxReEncTimes
0       1          20        51      10      45      15      3          3
2
```



```
-----RUN INFO1-----
ID  InsBr(kbps)  InsFr   WatL   CfgBt(kb)  RealBt(kb)  IPRatio   TarPercent
StartQp  MinQp    MaxQp
0      2030      30      1908     46          49          14
N/A      22        10      51

-----RC performance  INFO-----
ID  StaOfstaTim  TotaOfstaTim  StaOfEndTim  TotaOfEndTim  TotalTime
0   523265034    127267       523270697    8143         135410
```

The following is the proc information in VBR mode:

```
[RC] Version: [Hi3516A_MPP_V1.0.0.0 B010 Debug], Build Time[Aug 4 2016,
16:11:35]
```

```
----- BASE PARAMS -----
ID  Gop  StatTm  ViFr   TrgFr  ProType  RcMode  Br(kbps)  FluLev  IQp  PQp
2   240     8      30      30/0    265      VBR      2048      N/A      N/A  N/A
```

```
-----RUN COMM PARAM 1-----
ChnId  RowQpDelta
2           1  [3  3  5  5  8  8  8  15  20  20  25  25]
```

```
-----RUN COMM PARAM 2-----
ChnId  FirstFrmStartQP
2           -1  [3  3  5  5  8  8  8  15  20  20  225  225]
```

```
-----RUN COMM PARAM 3-----
ChnId  bLost  LostThr  LostFrmStr  EncGap  RCPriority  EnIDR
SprFrmMod  SprIFrm  SprPFrm  SprBFrm
2       N      83886080  NORMAL     0        BITRATE      Y
None      500000    500000    500000
```

```
-----RUN VBR PARAM -----
ID  IPQpDelta  ChgPs  MinIprop  MaxIprop  MaxStillQP  MinPercent
MaxReEncTimes  MinStillPSNR  MaxQp  MinQp  MaxIQp  MinIQp
2       1       90      1       20          N/A          N/A
N/A      N/A      51       15          45          44
```

```
-----RUN INFO1-----
ID  InsBr(kbps)  InsFr   WatL   CfgBt(kb)  RealBt(kb)  IPRatio
TarPercent  StartQp  MinQp    MaxQp
2      1737      30      5085     59          64          9
N/A      22        16      51
```



```
-----RC performance INFO-----
ID StaOfstaTim TotaOfstaTim StaOfEndTim TotaOfEndTim TotalTime
2 523270787    136763      523281424   9018        145781
```

The following is the proc information in AVBR mode:

```
[RC] Version: [Hi3516A_MPP_V1.0.0.0 B010 Debug], Build Time [Aug 4 2016,
16:11:35]
```

```
----- BASE PARAMS -----
ID Gop StatTm ViFr TrgFr ProType RcMode Br(kbps) FluLev IQp PQp
1 240     8     30   30/0    265    AVBR     2048    N/A    N/A N/A
-----RUN COMM PARAM 1-----
ChnId RowQpDelta ThrdI(12)
1           1 [3 3 5 5 8 8 8 15 20 20 25 25]
```

```
-----RUN COMM PARAM 2-----
ChnId FirstFrmStartQP ThrdP(12)
1           -1 [3 3 5 5 8 8 8 15 20 20 225 225]
```

```
-----RUN COMM PARAM 3-----
ChnId bLost LostThr LostFrmStr EncGap RCPriority EnIDR
SprFrmMod SprIFrm SprPFrm SprBFrm
1          N 83886080 NORMAL 0 BITRATE Y
None       500000 500000 500000
```

```
-----RUN VBR PARAM -----
ID IPQpDelta ChgPs MinIprop MaxIprop MaxStillQP MinPercent
MaxReEncTimes MinStillPSNR MaxQp MinQp MaxIQp MinIQp
1           1 90 1 100 35 25
0           0 51 15 45 44
```

```
-----RUN INFO1-----
ID InsBr(kbps) InsFr WatL CfgBt(kb) RealBt(kb) IPRatio
TarPercent StartQp MinQp MaxQp
1          1772 30 5085 58 49 10
98         23 16 51
```

```
-----RC performance INFO-----
ID StaOfstaTim TotaOfstaTim StaOfEndTim TotaOfEndTim TotalTime
1 523265331    161155      523276084   13167        174322
```

The following is proc information in FixQp mode:

```
[RC] Version: [Hi3516A_MPP_V1.0.0.0 B010 Debug], Build Time [Aug 4 2016,
```



16:11:35]

```
----- BASE PARAMS -----
ID      Gop     StatTm   ViFr    TrgFr  ProType   RcMode   Br(kbps)  FluLev  IQp    POp
0       30      1        30      30/0    265      FIXQP     N/A      N/A      20      24

-----RUN COMM PARAM 1-----
ChnId  RowQpDelta          ThrdI(12)
0           [ 255 255 255 255 255 255 255 255 255 255 255 255 255 255]

-----RUN COMM PARAM 2-----
ChnId FirstFrmStartQP      ThrdP(12)
0           -1   [ 255 255 255 255 255 255 255 255 255 255 255 255 255 255]

-----RUN COMM PARAM 3-----
ChnId  bLost    LostThr  LostFrmStr  EncGap  RCPriority  EnIDR   SprFrmMod
SprIFrm  SprPFrm  SprBFrm
0       N        83886080  NORMAL      0       BITRATE      Y       None
500000  500000  500000

-----RUN INFO1-----
ID      InsBr(kbps)  InsFr     WatL     CfgBt(kb)  RealBt(kb)  IPRatio
TarPercent StartQp    MinQp    MaxQp
0       976        30        0        0          1          13
N/A      24         24        24

-----RC performance  INFO-----
ID      StaOfstaTim  TotaOfstaTim  StaOfEndTim  TotaOfEndTim  TotalTime
0       8771981698    31627        8771987337    1194        32821
```

[Analysis]

This section records the bit rate control information during encoding.

[Parameter Description]

Parameter		Description
BASE PARAMS	ID	VENC channel ID
	Gop	Encoding group of pictures (GOP)
	StatTm	Bit rate statistics (in second)
	ViFr	Frame rate for transmitting pictures by the VIU
	TrgFr	Target frame rate for encoding
	ProType	Encoding type



Parameter	Description
RUN COMM PARAM 1	RcMode Bit rate control mode (CBR, VBR, AVBR, or FixQp)
	Br(kbps) In CBR mode, it indicates that average bit rate. In VBR mode, it indicates that the maximum bit rate. The unit of the bit rate is kbit/s.
	FluLev Fluctuation level, valid only for the CBR mode
	IQp I frame QP, valid only for the FixQp mode
	PQp P frame QP, valid only for the FixQp mode
	MinQp Minimum QP, valid only for VBR mode
	MinIQp Minimum I frame Qp, valid for the CBR and VBR modes
	MaxQp Maximum QP, valid only for the VBR mode
RUN COMM PARAM 1	ChnId VENC channel ID
	RowQpDelta Fluctuation amplitude of the start QP value of each macroblock row relative to the start QP value of the frame during macroblock-level bit rate control Value range: [0, 10]
	ThrdI(12) Mad threshold for controlling the macroblock-level bit rate of I frames Value range: [0, 255]
RUN COMM PARAM 2	ChnId VENC channel ID
	FirstFrmStartQP Start QP value of the first frame. The setting of this parameter is valid if it is configured before the first frame is encoded. The value -1 is displayed if this parameter is not configured.
	ThrdP(12) Mad threshold for controlling the macroblock-level bit rate of P frames Value range: [0, 255]
RUN COMM PARAM 3	ChnId VENC channel ID
	bLost Whether to discard frames when the transient bit rate exceeds the threshold
	LostThr Frame discarding threshold
	LostFrmStr Frame discarding policy when the instantaneous bit rate exceeds the threshold
	EncGap Interval of encoding frames. For details, see the description of VENC_PARAM_LOSTFRM_S in chapter 6 "VENC."
	RCPriority RC priority BITRATE: The target bit rate takes priority.



Parameter	Description	
	EnIDR	FRAMEBITS: The jumbo frame threshold takes priority.
	SprFrmMod	IDR enable switch
	SprIFrm	Jumbo frame policy <ul style="list-style-type: none">• None: The jumbo frame is not checked.• Lost: The jumbo frame is discarded.• ReEncode: The jumbo frame is re-encoded.
	SprPfrm	Threshold of the jumbo I frame
	SprBfrm	Threshold of the jumbo P frame
	RUN CBR PARAM1	Threshold of the jumbo B frame
RUN VBR PARAM		For details, see the description of VENC_PARAM_H264_CBR_S in chapter 6 "VENC."
RUN INFO1	ID	VENC channel ID
	InsBr(kbps)	Instant bit rate (in kbit/s)
	InsFr	Instant frame rate
	WatL	Bit rate threshold (for internal debugging)
	CfgBt(kb)	Target size of the current frame (in kbit)
	RealBt(kb)	Actual size of the streams in the previous frame (in kbit)
	IPRatio	Ratio of the I frame size to P frame size
	TarPercent	Current target bit rate of AVBR (percentage)
	StartQp	Start QP
	MinQp	Minimum QP
	MaxQp	Maximum QP
RC PERFORMANCE INFO	ID	VENC channel ID
	StaOfstaTim	Time spent by the RC for calculating the configuration information about a frame such as QP before encoding starts
	TotaOfstaTim	Time spent by the RC for calculating the configuration information about n frames to be encoded such as QP before encoding starts
	StaOfEndTim	Time spent by the RC for updating the information



Parameter	Description
	about a frame such as the number of bytes after the frame is encoded
TotaOfEndTim	Time spent by the RC for updating the information about n frames such as the number of bytes after the n frames are encoded
TotalTime	The total time is calculated as follows: TotalTime = TotaOfstaTim + TotaOfEndTim

12.10 Region

[Debugging Information]

```
[RGN] Version: [Hi3516A_MPP_V1.0.0.0 B010 Debug], Build Time [Sep 15 2014,  
15:10:15]
```

```
-----REGION STATUS OF OVERLAY-----
```

Hdl	Type	Used	PiFmt	W	H	BgColor	Phy	Virt	Stride
-----	------	------	-------	---	---	---------	-----	------	--------

```
-----REGION CALL TDE STATUS OF OVERLAY-----
```

Hdl	CallCnt	JobSuc	JobFail	TaskSuc	TaskFail	EndSuc	EndFail
-----	---------	--------	---------	---------	----------	--------	---------

```
-----REGION CHN STATUS OF OVERLAY-----
```

Hdl_Type	Mod	Dev	Chn	bSh	X	Y	AphF	AphB	Layer	bAQP	QP	bQpDis	bInv	InvW	InvH	Luma	ChnM
----------	-----	-----	-----	-----	---	---	------	------	-------	------	----	--------	------	------	------	------	------

```
-----REGION STATUS OF COVER-----
```

Hdl	Type	Used
-----	------	------

```
-----REGION CHN STATUS OF RECT COVER-----
```

Hdl	Type	Mod	Dev	Chn	bShow	X	Y	W	H	Field	Color	Layer
-----	------	-----	-----	-----	-------	---	---	---	---	-------	-------	-------

```
-----REGION CHN STATUS OF QUAD_RANGLE COVER-----
```

Hdl	Type	Mod	Dev	Chn	bShow	X0	Y0	X1	Y1	X2	Y2	X3	Y3
Solid	Thick	Shape	Field	Color	Layer								

```
-----REGION STATUS OF COVEREX-----
```

Hdl	Type	Used
-----	------	------

0	2	0
1	2	0
2	2	0
3	2	0
4	2	0



```
5 2 0
6 2 0
7 2 0
```

-----REGION CHN STATUS OF RECT COVEREX-----

Hdl	Type	Mod	Dev	Chn	bShow	X	Y	W	H	Field	Color	Layer
-----	------	-----	-----	-----	-------	---	---	---	---	-------	-------	-------

-----REGION CHN STATUS OF QUAD_RANGLE COVEREX-----

Hdl	Type	Mod	Dev	Chn	bShow	X0	Y0	X1	Y1	X2	Y2	X3	Y3
-----	------	-----	-----	-----	-------	----	----	----	----	----	----	----	----

Solid	Thick	Shape	Field	Color	Layer
-------	-------	-------	-------	-------	-------

0	2	7	0	0	Y	0	0	100	100	100	300	0	200	Y
2	1	frame		fffff	0									
1	2	7	0	0	1	100	0	200	100	200	300	100	200	1

-----REGION STATUS OF OVERLAYEX-----

Hdl	Type	Used	PiFmt	W	H	BgColor	Phy	Virt	Stride
-----	------	------	-------	---	---	---------	-----	------	--------

-----REGION CALL TDE STATUS OF OVERLAYEX-----

Hdl	CallCnt	JobSuc	JobFail	TaskSuc	TaskFail	EndSuc
	EndFail					

-----REGION CHN STATUS OF OVERLAYEX-----

Hdl	Type	Mod	Dev	Chn	Field	bSh	X	Y	AphF	AphBLayer
-----	------	-----	-----	-----	-------	-----	---	---	------	-----------

[Analysis]

This section records the information about the current region.

[Parameter Description]

Parameter	Description	
REGION STATUS OF OVERLAY	Hdl	Overlay handle ID
	Type	Overlay type. Its value is 0.
	Used	Number of times that resources are being used
	PiFmt	Overlay pixel format. For details, see PIXEL_FORMAT_E.
	W	Overlay width (in pixel)
	H	Overlay height (in pixel)
	BgColor	Overlay background color
	Phy	Physical address for the memory occupied by the overlay region
	Virt	Virtual address for the memory occupied by the overlay region
	Stride	Overlay memory stride (in byte)



Parameter	Description	
REGION CALL TDE STATUS OF OVERLAY	Hdl	Overlay handle ID
	CallCnt	Number of times that the overlay region copies data by using the TDE
	JobSuc	Number of times that TDE jobs are successfully submitted
	JobFail	Number of times that TDE jobs fail to be submitted
	TaskSuc	Number of times that tasks are successfully added to TDE jobs
	TaskFail	Number of times that tasks fail to be added to TDE jobs
	EndSuc	Number of times that TDE jobs are successfully completed
	EndFail	Number of times that TDE jobs fail to be completed
REGION CHN STATUS OF OVERLAY	Hdl	Overlay handle ID
	Type	Overlay type. Its value is 0.
	Mod	Module ID
	Dev	Device ID
	bsh	Whether to show the overlay region in the channel N: hide Y: show
	Chn	Channel ID
	X	Horizontal coordinate of the start position of the overlay region in the channel
	Y	Vertical coordinate of the start position of the overlay region in the channel
	AphF	Foreground alpha displayed in the channel
	AphB	Background alpha displayed in the channel
	Layer	Layer displayed in the channel
	bAQp	Whether the value is an absolute QP value N: relative QP value Y: absolute QP value
	QP	QP parameters for the encoding module
	bQpDis	Whether OSD QP protection is disabled
	bInv	Whether the inverse color function is enabled
	InvW	Width of the basic unit for color inversion



Parameter	Description	
	InvH	Height of the basic unit for color inversion
	Luma	Luminance threshold during color inversion
	ChnM	Color inversion mode: 0: trigger color inversion when the video background luminance is less than the luminance threshold 1: trigger color inversion when the video background luminance is greater than the luminance threshold
REGION STATUS OF COVER	Hdl	Cover handle ID
	Type	Cover type. Its value is 1.
	Used	Number of times that resources are being used
REGION CHN STATUS OF RECT COVER	Hdl	Cover handle ID
	Type	Cover type. Its value is 1.
	Mod	Module ID
	Dev	Device ID
	Chn	Channel ID
	bShow	Whether to show the Cover region in the channel N: hide Y: show
	X	Horizontal coordinate of the start position of the cover region in the channel
	Y	Vertical coordinate of the start position of the cover region in the channel
	W	Cover width (in pixel)
	H	Cover height (in pixel)
	Field	Attribute indicating that a region is overlaid with a frame or field frame: A region is overlaid with a frame. top: A region is overlaid with a top field. botm: A region is overlaid with a bottom field.
	Color	Cover color
	Layer	Layer displayed in the channel
REGION CHN STATUS OF QUAD_RANGLE COVER	Hdl	Cover handle ID
	Type	Cover type. Its value is 1.
	Mod	Module ID
	Dev	Device ID



Parameter	Description	
	Chn	Channel ID
	bShow	Whether to show the Cover region in the channel N: hide Y: show
	X0	Horizontal coordinate of coordinate 0
	Y0	Vertical coordinate of coordinate 0
	X1	Horizontal coordinate of coordinate 1
	Y1	Vertical coordinate of coordinate 1
	X2	Horizontal coordinate of coordinate 2
	Y2	Vertical coordinate of coordinate 2
	X3	Horizontal coordinate of coordinate 3
	Y3	Vertical coordinate of coordinate 3
	Solid	Cover border type N: dashed Y: solid
	Thick	Cover border thickness, valid only for dashed borders
	Shape	Cover shape 0: rectangle 1: quadrilateral
	Field	Attribute indicating that a region is overlaid with a frame or field frame: A region is overlaid with a frame. top: A region is overlaid with a top field. botm: A region is overlaid with a bottom field.
	Color	Cover color
	Layer	Layer displayed in the channel
REGION STATUS OF COVEREX	Hdl	CoverEx handle ID
	Type	CoverEx type. Its value is 2.
	Used	Number of times that resources are being used
REGION CHN STATUS OF RECT COVEREX	Hdl	CoverEx handle ID
	Type	CoverEx type. Its value is 2.
	Mod	Module ID. The VIU ID is 16.
	Dev	Device ID



Parameter	Description	
	Chn	Channel ID
	bShow	Whether to show the CoverEx region in the channel N: hide Y: show
	X	Horizontal coordinate of the start position of the CoverEx region in the channel
	Y	Vertical coordinate of the start position of the CoverEx region in the channel
	W	CoverEx width (in pixel)
	H	CoverEx height (in pixel)
	Field	Attribute indicating that a region is overlaid with a frame or field frame: A region is overlaid with a frame. top: A region is overlaid with a top field. botm: A region is overlaid with a bottom field.
	Color	CoverEx color
	Layer	Layer displayed in the channel
REGION CHN STATUS OF QUAD_RANGLE COVEREX	Hdl	CoverEx handle ID
	Type	CoverEx type. Its value is 1.
	Mod	Module ID
	Dev	Device ID
	Chn	Channel ID
	bShow	Whether to show the CoverEx region in the channel N: hide Y: show
	X0	Horizontal coordinate of coordinate 0
	Y0	Vertical coordinate of coordinate 0
	X1	Horizontal coordinate of coordinate 1
	Y1	Vertical coordinate of coordinate 1
	X2	Horizontal coordinate of coordinate 2
	Y2	Vertical coordinate of coordinate 2
	X3	Horizontal coordinate of coordinate 3
	Y3	Vertical coordinate of coordinate 3
	Solid	CoverEx border type



Parameter	Description	
	N: dashed Y: solid	
Thick	CoverEx border thickness, valid only for dashed borders	
Shape	CoverEx shape 0: rectangle 1: quadrilateral	
Field	Attribute indicating that a region is overlaid with a frame or field frame: A region is overlaid with a frame. top: A region is overlaid with a top field. botm: A region is overlaid with a bottom field.	
Color	CoverEx color	
Layer	Layer displayed in the channel	
REGION STATUS OF OVERLAYEX	Hdl Type Used PiFmt W H BgColor Phy Virt Stride	OverlayEx handle ID OverlayEx type. Its value is 3. Number of times that resources are being used OverlayEx pixel format. For details, see PIXEL_FORMAT_E. OverlayEx width (in pixel) OverlayEx height (in pixel) OverlayEx background color Physical address for the memory occupied by the OverlayEx region Virtual address for the memory occupied by the OverlayEx region OverlayEx memory stride (in byte)
REGION CALL TDE STATUS OF OVERLAYEX	Hdl CallCnt JobSuc JobFail TaskSuc TaskFail	OverlayEx handle ID Number of times that the OverlayEx region copies data by using the TDE Number of times that TDE jobs are successfully submitted Number of times that TDE jobs fail to be submitted Number of times that tasks are successfully added to TDE jobs Number of times that tasks fail to be added to TDE



Parameter	Description	
	jobs	
	EndSuc	Number of times that TDE jobs are successfully completed
	EndFail	Number of times that TDE jobs fail to be completed
REGION CHN STATUS OF OVERLAYEX	Hdl	OverlayEx handle ID
	Type	OverlayEx type. Its value is 0.
	Mod	Module ID. The VIU ID is 16.
	Dev	Device ID
	Chn	Channel ID
	Field	Attribute indicating that a region is overlaid with a frame or field frame: A region is overlaid with a frame. top: A region is overlaid with a top field. botm: A region is overlaid with a bottom field.
	bSh	Whether to show the Vloverlay region in the channel N: hide Y: show
	X	Horizontal coordinate of the start position of the Vloverlay region in the channel
	Y	Vertical coordinate of the start position of the Vloverlay region in the channel
	AphF	Foreground alpha displayed in the channel
	AphB	Background alpha displayed in the channel
	Layer	Layer displayed in the channel

12.11 VENC

NOTE

This section is not supported by Hi3519 V100

[Debugging Information]

```
# cat /proc/umap/venc
[VENC] Version: [Hi3516A_MPP_V1.0.0.0 Debug], Build Time [Sep 6 2011,
09:51:16]
-----MODULE PARAM-----
VencBufferCache
```



```
0
-----VENC CHN ATTR1-----
ID  Width  Height  Type  ByFrame  Timeout  Sequence
0    720     576      96          Y           1           2
LeftBytes  LeftFrm  CurPacks  prio
0        0        0        1

-----VENC CHN ATTR 2-----
VeStr  OsdStr  SrcFr  TarFr  Timeref  PixFmt  PicAddr
Y       N       -1      -1        24      YUV420  0x89170000

----- VENC CHN RECEIVE STAT-----
ID      Start      StartEx     RecvLeft     EncLeft
0        1          0            0            0

----- VENC VPSS QUERY-----
ID.  Query     QueryOk   QueryFR   Invld   Full   VbFail   QueryFail
InfoErr      Stop
0        0          0            0            0            0            0            0            0            0            0

----- VENC SEND1-----
ID      VpssSnd   VInfErr   OthrSnd   OInfErr   Send   Stop
Full     CropErr   DirectSnd  SizeErr
0        0          0            12          0            12            0

----- VENC SEND2-----
ID      SendVgs   StartOk   StartFail   IntOk   IntFail   SrcAdd
SrcSub   DestAdd   DestSub
0        12         12          0            12            0            12
12        12         0

----- VENC PIC QUEUE STATE-----
ID.  Free     Busy     Vgs
0        2         4          0

----- VENC CHNL INFO-----
ID      Inq      InqOk     Start     StartOk   Config   VencInt
ChaResLost   OverLoad
0        48         9          9          9            9            8
0        0

----- VENC CROP INFO-----
ID      CropEn   StartX   StartY   Width   Height
```



0 1 0 0 1280 720

-----VENC STREAM STATE-----

ID	FreeCnt	BusyCnt	UserCnt	UserGet	UserRls	GetTimes	Interval	FrameRate
0	4	0	0	875	875	850	15	25

[Analysis]

This section records the attributes and status of the current VENC channel.

[Parameter Description]

Parameter	Description
MODULE PARAM	VencBufferCache Whether the encoding stream buffer uses the cache mode 0: no 1: yes Note: If each frame of encoding streams needs to be copied for multiple times by calling the memcpy function, you are advised to enable the cache mode to improve the efficiency. It is recommended that the cache mode be disabled in other cases.
VENC CHN ATTR1	ID VENC channel ID
	Width VENC channel width
	Height VENC channel height
	Type VENC channel type
	ByFrame Mode of obtaining streams 0: by packet 1: by frame
	Timeout Timeout period for obtaining streams -1: block mode 0: non-block mode (0, +∞): timeout period
	Sequence Sequence number When the streams are obtained by frame, it represents the frame sequence number. When the streams are obtained by packet, it represents the packet sequence number.
	LeftBytes Remaining bytes in a stream buffer
	LeftFrm Number of remaining stream frames in a stream buffer
	CurPacks Number of stream packets for the current frame (invalid currently)



Parameter		Description
	prio	Channel priority
VENC CHN ATTR 2	VeStr	Whether to start encoding
	OsdStart	OSD enable
	SrcFr	Source frame rate (input frame rate) used by the VENC for controlling the frame rate
	TarFr	Target frame rate used by the VENC for controlling the frame rate
	Timeref	Timeref of the latest frame in the busy queue
	PixFmt	Format of the frame that is being encoded The value is YUV422, YUV420, or YUV400.
	PicAddr	Address for the frame that is being encoded
VENC CHN RECEIVE STAT	ID	VENC channel ID
	Start	Whether the VENC channel starts to receive pictures
	StartEx	Whether the VENC channel starts to receive pictures by frame
	RecvLeft	Number of frames that are to be received by the VENC. This parameter is used with HI_MPI_VENC_StartRecvPicEx().
	EncLeft	Number of frames that are to be encoded by the VENC. This parameter is used with HI_MPI_VENC_StartRecvPicEx().
VENC VPSS QUERY	ID	VENC channel ID
	Query	Total number of times that the VENC is queried by the VPSS
	QueryOk	Number of times that the VENC is successfully queried by the VPSS
	QueryFR	Number of times the VENC discards frames due to frame rate control, which is queried by the VPSS
	Invld	Number of times that the source picture of the VENC is empty, which is queried by the VPSS
	Full	Number of times that the picture queue of the VENC is full, which is queried by the VPSS
	VbFail	Number of times that the VENC fails to request VBs, which is queried by the VPSS
	QueryFail	Number of times that the VPSS fails to query the VENC
	InfoErr	Number of times that the VPSS fails to query the VENC due to incorrect information



Parameter		Description
	Stop	Number of times that the VPSS fails to query the VENC because the VENC stops receiving pictures
VENC SEND1	ID	VENC channel ID
	VpssSnd	Number of times that the VPSS transmits pictures
	VInfErr	Number of times that the VPSS fails to transmit pictures due to incorrect picture information
	OthrSnd	Number of times that other modules transmit pictures
	OInfErr	Number of times that other modules fail to transmit pictures due to incorrect picture information
	Send	Number of times that external modules transmit pictures properly
	Stop	Number of times that the VENC stops receiving pictures when external modules are transmitting pictures
	Full	Number of times that the picture queue is full when external modules are transmitting pictures
	CropErr	Number of lost frames caused by errors of the crop parameter
	DrectSnd	Number of times that the external modules transmit pictures to the picture busy queue directly (pictures are transmitted to the VGS)
VENC SEND2	SizeErr	Number of times that the VENC refuses to receive pictures. This item increases when the width or height of the picture transmitted by the front-end is less than the width or height of the VENC channel.
	ID	VENC channel ID
	SendVgs	Number of times that pictures are transmitted to the VGS
	StartOk	Number of times that the VGS is started successfully
	StartFail	Number of times that the VGS fails to be started
	IntOk	Number of interrupts indicating that the VGS tasks are performed successfully
	IntFail	Number of interrupts indicating that VGS tasks fail to be performed
	SrcAdd	Number of times that the count of using the source picture VBs increases
	SrcSub	Number of times that the count of using the source picture VBs decreases
	DestAdd	Number of times that the count of using the target



Parameter		Description
		picture VBs increases
	DestSub	Number of times that the count of using the target picture VBs decreases
VENC PIC QUEUE STATE	ID	VENC channel ID
	Free	Number of the nodes in the VENC free queue (number of frames allowed)
	Busy	Number of the nodes in the VENC busy queue (allowed number of used frames)
	Vgs	Number of frames being processed by the VENC
VENC CHNL INFO	ID	VENC channel ID
	Inq	Number of times that VENC channel is queried
	InqOk	Number of times that VENC channel is queried successfully
	Start	Number of times that the VENC channel starts encoding
	StartOk	Number of times that the VENC channel starts encoding successfully
	Config	Number of times that the VENC channel configures encoding hardware
	VencInt	Number of times that the VENC channel receives interrupts
	ChaResLost	Number of times that frames are discarded due to resolution switching of the VENC channel
	OverLoad	Number of times that the channels fail to start encoding due to the insufficient performance
VENC CROP INFO	ID	VENC channel ID
	CropEn	Whether to enable the cropping function of the VENC channel
	StartX	Start horizontal coordinate of the picture to be cropped
	StartY	Start vertical coordinate of the picture to be cropped
	Width	Width of the cropped picture
	Height	Height of the cropped picture
VENC STREAM STATE	ID	VENC channel ID
	FreeCnt	Number of free nodes in a stream buffer
	BusyCnt	Number of busy nodes in a stream buffer
	UserCnt	Number of user nodes in a stream buffer This number is incremented by 1 each time users obtain



Parameter	Description
	a frame successfully. This number decreased by 1 each time users release a frame successfully.
UserGet	Number of stream packets from which streams are obtained successfully
UserRls	Number of stream packets whose streams are released successfully
GetTimes	Number of times that streams are obtained
Interval	Interval of obtaining two frames from the encoder (in μ s)
FrameRate	Number of frames obtained from the encoder in one second, that is, frame rate of the encoder

12.12 VI

[Debugging Information]

```
# cat /proc/umap/vi
[VIU] Version: [Hi3516A_MPP_V1.0.0.0 B010 Debug], Build Time: [Jun 19 2014,
14:31:29]
VI-VPSS is offline.

-----MODULE PARAM-----
detect_err_frame drop_err_frame stop_int_level
          0          0          0

-----VI DEV ATTR-----
Dev  IntfM  WkM  ComMsk0  ComMsk1  ScanM AD0 AD1 AD2 AD3  Seq  DPPath DType
DRev CapX CapY CapW CapH
0    MIPI  1Mux  fff0000      0     P  -1  -1  -1  -1  N/A   ISP   RGB   N
0    20   1920  1080

-----VI HIGH DEV ATTR-----
Dev  InputM  WkM  ComMsk0  ComMsk1 ScanM AD0 AD1 AD2 AD3  Seq CombM CompM ClkM
Fix  FldP  DPPath  DType  DRev  CapX CapY CapW CapH

-----VI PHYCHN ATTR-----
PhyChn CapX CapY  CapW  CapH  DstW  DstH CapSel Mirror Flip IntEn PixFom SrcRat
DstRat  Comp
0     0     0   1920   1080   1920   1080 both      N     N      Y   SP420     30     30
```



Y

-----VI PHYCHN STATUS 1-----

PhyChn	Dev	IntCnt	VbFail	LosInt	TopLos	BotLos	BufCnt	IntT	SendT
Field	Stride								
0	0	23020	0	2	0	2	797	397	frm
		1920							

-----VI PHYCHN STATUS 2-----

PhyChn	MaxIntT	IntGapT	MaxGapT	LIntCnt	ThrCnt	AutoDis	CasAutD	TmgErr
ccErrN	IntRat							
0	856	40019	41173	0	1	0	0	2
								24

-----VI OTHER ATTR-----

LDC	Mode	Ratio	COffX	COffY	Enable
--	All	0	0	0	N

Flash	Mode	StartTime	DuraTime	InterVal	CapIdx	Enable	FlashedNum
--	Once	0	0	0	0	N	0

CSC	Type	HueVal	ContrVal	LumaVal	StatuVal	TVMode
--	USER	50	50	50	50	N

Idc[0]	Idc[1]	Idc[2]
0	0	0

Odc[0]	Odc[1]	Odc[2]
16	128	128

Coef[0]	Coef[1]	Coef[2]	Coef[3]	Coef[4]	Coef[5]	Coef[6]	Coef[7]
Coef[8]							
183	614	62	-101	-338	439	439	-399
-40							

DCI	En	BlackGain	ContrGain	LightGain
--	N	0	0	0

DIS En

-- N

-----VI WDR ATTR-----

Mode	BufNum	DstW	DstH	PoolId	VcNum	DesNum	State	bCompress
NONE	0	0	0	-1	0	0	NONE	N



```
-----VI WDR DES STATUS-----
Idx IntGap IntCnt CcErrCnt

-----VI WDR SRC STATUS-----
Idx IntGap IntCnt CcErrCnt

-----VI WDR COMBINE STATUS-----
IntGap IntCnt CcErrCnt
0 0 0

-----VI EXTCHN ATTR-----
ExtChn BindChn CropEn CropX CropY CropW CropH DstW DstH PixFom SrcRat
DstRat Depth Comp
1 0 Y 340 340 820 700 1920 1080 SP420 30
30 0 Y

-----VI FISHEYE ATTR-----
ViChn Enable MntMode RgnNum BgEnable BgColor LMF HOffset VOffset TCoef
4 1 Desktop 1 0 0xfc 0 0 0 0 0

-----VI FISHEYE Region Attr-----
ExtChn RgnIndex ViewMode InRadius OutRadius Pan Tilt HorZoom VerZoom
4 0 360 2 960 180 180 4095 4095
OutX OutY OutW OutH
0 0 1920 1080

-----USER PIC INFO-----
UPicID Width Height Stride Field PixForm PoolID PhyAddr bUpdate
0 1920 1080 1920 frm sp420 1 8a534000 N

-----VI DIS INFO1-----
ViChn bEnable Acc Mode Fixlevel Roef BufNum MovSub NoMov Dof
CropRatio DISFrmRate
0 Y High NORMAL 7 80 8 0 0 8DOF_HARD 80
60

-----VI DIS INFO2-----
ViChn TimeLag AngleType ViewAngle bScale OutW OutH DelayNum
RetCenter GyroWeight bStillCrop
0 16666 DIAGONAL 565 N 1536 864 0 7
0 N

-----VI DISDebug INFO-----
ViChn PyFrmlost AfFrmlost Pyflag Srcflag Mvflag PyTime AfTime MvTime
MaxPyTime MaxAfTime MaxMvTime
```



```
0      0      0 00000000 00000001 00000000 7639 4290 123
10331 7079 363
```

-----VI CHN STATUS-----

ViChn	bEnUsrP	FrmTime	FrmRate	SendCnt	SwLost	Rotate
Depth						
0	N	40017	25	23018	0	NONE
						0

-----VI CHN CALL VGS STATUS 1-----

ViChn	UsrBgnNok	UsrCancel	UsrEndOk	UsrCbOk	CovBgnNok
CovCancel	CovEndOk	CovCbOk			

-----VI CHN CALL VGS STATUS 2-----

ViChn	OsdBgnNOK	OsdCancel	OsdEndOk	OsdCbOk	ScaleNOK
SclCancel	SclEndOk	SclCbOk			

-----VI CHN CALL VGS STATUS 3-----

ViChn	RotateNOK	RotCancel	RotEndOk	RotCbOk	LDCNOK
LDCCancel	LDCEndOk	LDCCbOk			

[Analysis]

This section records the attributes and status of the current VI device and channel.

[Parameter Description]

Parameter	Description
MODULE PARAM	<p>detect_err_frame</p> <p>Policy of discarding detected error frames in real time when the signal is unstable</p> <p>> 0: When the number of consecutive error frames detected is greater than the parameter value, the system considers that an incorrect timing is configured, and the subsequent frames are not discarded.</p> <p>0: default value. The detected error frames are discarded in real time.</p> <p>< 0: Error frame detection is disabled.</p>
	<p>drop_err_frame</p> <p>When the system detects that the current frame is an error frame, the system considers the subsequent frames as error frames and discards them.</p> <p>0: default value. The function of discarding consecutive frames is disabled and only the current error frame is discarded.</p> <p>> 0: Consecutive drop_err_frame frames (including the current frame) are discarded when the system detects a picture error no matter</p>



Parameter	Description	
		whether the subsequent frames are correct.
	stop_int_level	When the interval between two interrupts is less than the parameter value, interrupt response is automatically disabled, ensuring that applications are responded. The default value is 0, and the unit is μ s.
VI DEV ATTR (for details, see the description of VI_DEV_ATTR_S)	Dev	VI device ID
	IntfM	Input mode
	WkM	Working mode
	ComMsk0	Component 0 mask
	ComMsk1	Component 1 mask
	ScanM	Scan mode (interlaced input or progressive input) Value range: {I, P}
	AD0	AD ID
	AD1	AD ID
	AD2	AD ID
	AD3	AD ID
	Seq	Data sequence Value range: {VUVU, UVUV, UYVY, VYUY, YUYV, YYVU}
	DPath	ISP enable. The default value is ISP.
	DType	Input data type. The default value is RGB.
	DRev	Whether the ADC and sensor reversely connect to the VI interface. The default value is N (indicating no reverse).
	CapX	Horizontal coordinate of the crop start position of a VI device
	CapY	Vertical coordinate of the crop start position of a VI device
	CapW	Width of the cropped picture of a VI device
	CapH	Height of the cropped picture of a VI device
VI HIGH DEV ATTR Advanced VI device attributes (for details, see	Dev	VI device ID
	InputM	Input mode
	WkM	Working mode
	ComMsk0	Component 0 mask



Parameter	Description	
VI_DEV_ATTR_S)	ComMsk1	Component 1 mask
	ScanM	Scan mode (interlaced input or progressive input) Value range: {I, P}
	AD0	AD ID
	AD1	AD ID
	AD2	AD ID
	AD3	AD ID
	Seq	Data sequence. Value range: {VUVU, UVUV, UYVY, VYUY, YUYV, YVYU}
	CombM	Component composite mode or separation mode Value range: {COMP, SEPAR}
	CompM	Single-component or dual-component mode Value range: {SING, DOUB}
	ClkM	Clock mode Value range: {UP, DOWN}
	Fix	Configuration of the most significant bit (MSB) of the timing reference code Value range: {0, 1}
	FldP	Field indicator mode of the timing reference code Value range: {STD, NSTD}
	DPath	ISP enable. The default value is ISP.
	DTypE	Input data type. The default value is RGB.
VI PHYCHN ATTR Primary attributes	DRev	Whether the ADC and sensor reversely connect to the VI interface. The default value is N (indicating no reverse).
	CapX	Horizontal coordinate of the crop start position of a VI device
	CapY	Vertical coordinate of the crop start position of a VI device
	CapW	Width of the cropped picture of a VI device
	CapH	Height of the cropped picture of a VI device
VI PHYCHN ATTR Primary attributes	PhyChn	ID of a physical VI channel
	CapX	Horizontal coordinate of the start position of the capture region



Parameter	Description	
of the physical VI channel (for details, see <code>VI_CHN_ATTR_S</code>)	CapY	Vertical coordinate of the start position of the capture region
	CapW	Width of the capture region
	CapH	Height of the capture region
	DstW	Target picture width (obtained by scaling the captured picture)
	DstH	Target picture height (obtained by scaling the captured picture)
	CapSel	Frame/Field select Value range: {top, bottom, both}
	Mirror	Mirror enable Value range: {Y, N}
	Flip	Flip enable Value range: {Y, N}
	IntEn	Whether to respond to interrupts Value range: {Y, N}
	PixFom	Pixel format Value range: {SP420, SP422, YUV400}
SrcRat	Source frame rate for controlling the capture frame rate of a physical VI channel The value is configured based on the value of <code>VI_CHN_ATTR_S</code> .	
	DstRa	Target frame rate for controlling the capture frame rate of a physical VI channel The value is configured based on the value of <code>VI_CHN_ATTR_S</code> .
	Comp	Whether to compress data Value range: {Y, N}
VI PHYCHN STATUS Basic status 1 of physical VI channel	Dev	ID of a VI device bound to a physical VI channel
	PhyChn	ID of a physical VI channel
	IntCnt	Interrupt count The value is incremented by 1 each time the frame is interrupted. If a VI channel does not properly work, the value may not increase.
	VBFail	Number of times that VBs fail to be obtained



Parameter	Description
	Before capturing a frame, the VI channel obtains the VB with the channel picture size. If the VB fails to be obtained, the value is incremented by 1. Typically, the value should be 0. Otherwise, the configuration of the public VB is incorrect.
LosInt	Number of times that interrupts are lost The VI channel needs to prepare for capturing pictures at the intervals of the initial one or two interrupts. It is normal that the value is 1 or 2 .
TopLos	Number of lost top field interrupts
BotLos	Number of lost bottom field interrupts
BufCnt	Number of picture VBs occupied by the VIU
IntT	Time of processing interrupts (in μ s)
SendT	Time of transmitting pictures during interrupt processing (in μ s)
Field	Frame/Field flag of a picture frm: progressive picture intl: interlaced picture
Stride	Picture stride
VI PHYCHN STATUS 2 Basic status 2 of physical VI channel	PhyChn ID of a physical VI channel MaxIntT Maximum time of processing interrupts IntGapT Interval between two adjacent interrupts MaxGapT Maximum interval between two adjacent interrupts LIntCnt Number of times that the interrupt processing time exceeds int_time/1000 ms. int_time is a module parameter, and its default value is 10000. ThrC0nt Number of times that frames or fields are lost. The parameter value is 1 in most cases when the program just starts. It is normal that this parameter value does not increase subsequently. AutoDis Number of times that interrupt mask is automatically disabled due to too many interrupts CasAutD Number of frames discarded due to interrupt delay in VI-VO cascade mode TmgErr Number of interrupts generated due to exceptions such as the buffer overflow ccErrN Number of interrupts indicating that data capture is not complete. If data is captured at a non-full



Parameter	Description	
	frame rate or VBFail increases, ccErrN also increases. If data is captured at a full frame rate and VBFail does not increase but ccErrN increases, check the input timing or input line connection.	
IntRat	Capture frame rate determined by VI interrupts	
LDC attribute	Mode	LDC mode (all or crop)
	Ratio	Correction coefficient
	COffX	Horizontal coordinate of the correction center
	COffY	Vertical coordinate of the correction center
	Enable	Whether to enable correction
Flash attribute	Mode	Blinking mode (blinking once or frequent blinking)
	StartTime	Blinking start time (in clock cycle)
	DuraTime	Blinking duration (in clock cycle)
	InterVal	Blinking interval in frequent blinking mode (in frame)
	CapIdx	Flag indicating the frame captured when the camera flash is on
	Enable	Whether to enable blinking
	FlashedNum	Number of frames that are captured when the camera flash is on
CSC attribute	Type	Color space conversion (CSC) type (601 or 709)
	HueVal	Chrominance strength Value range: [0, 100]
	ContrVal	Contrast strength Value range: [0, 100]
	LumaVal	Luminance strength Value range: [0, 100]
	StatuVal	Saturation strength Value range: [0, 100]
	TVMode	Value: {Y, N} N: The constant coefficient matrix for CSC is [0, 255]. Y: The constant coefficient matrix for CSC is [16, 235].



Parameter	Description	
	Idc[0]	Offset of the input DC component R Value range: [-256, +255]
	Idc[1]	Offset of the input DC component G Value range: [-256, +255]
	Idc[2]	Offset of the input DC component B Value range: [-256, +255]
	Odc[0]	Offset of the output DC component Y Value range: [-256, +255]
	Odc[1]	Offset of the output DC component U Value range: [-256, +255]
	Odc[2]	Offset of the output DC component V Value range: [-256, +255]
	Coef[0]	CSC matrix coefficient Value range: [-16384, +16383]
	Coef[1]	CSC matrix coefficient Value range: [-16384, +16383]
	Coef[2]	CSC matrix coefficient Value range: [-16384, +16383]
	Coef[3]	CSC matrix coefficient Value range: [-16384, +16383]
	Coef[4]	CSC matrix coefficient Value range: [-16384, +16383]
	Coef[5]	CSC matrix coefficient Value range: [-16384, +16383]
	Coef[6]	CSC matrix coefficient Value range: [-16384, +16383]
	Coef[7]	CSC matrix coefficient Value range: [-16384, +16383]
	Coef[8]	CSC matrix coefficient Value range: [-16384, +16383]
DCI attribute	En	Whether to enable dynamic contrast improvement (DCI)
	BlackGain	Dark gain Value range: [0, 63]
	ContrGain	Contrast gain



Parameter	Description	
	LightGain	Value range: [0, 63] Bright gain Value range: [0, 63]
DIS attribute	En	Whether to enable digital image stabilization (DIS)
VI WDR ATTR	Mode	WDR mode
	BufNum	Number of allocated buffers
	DstW	Picture width
	DstH	Picture height
	PoolId	ID of the memory pool from which the memory is allocated
	VcNum	Number of VCs
	DesNum	Number of DESs. The value is 1 at the full frame rate and VcNum at other frame rates
	State	Combination status
VI WDR DES STATUS (WDR write frame status)	bCompress	Whether to compress data
	Idx	DES ID
	IntGap	DES interrupt interval
	IntCnt	Number of DES interrupts
VI WDR SRC STATUS (WDR read frame status)	CcErrCnt	Number of errors indicating that the DES write operation is incomplete
	Idx	SRC ID
	IntGap	SRC interrupt interval
	IntCnt	Number of SRC interrupts
VI WDR COMBINE STATUS	CcErrCnt	Number of errors indicating that the SRC read operation is incomplete
	IntGap	Interval of combining interrupts
	IntCnt	Number of combined interrupts
VI EXTCHN ATTR	CcErrCnt	Number of errors indicating that interrupt combination is incomplete
	ExtChn	ID of an extended VI channel
	BindChn	ID of the primary channel to which an extended channel is bound
	CropEn	Cropping enable of an extended VI channel



Parameter	Description
VI FISHEYE ATTR	CropX Horizontal start coordinates of the region to be cropped in an extended VI channel
	CropY Vertical start coordinates of the region to be cropped in an extended VI channel
	CropW Width of the region to be cropped in an extended VI channel
	CropH Height of the region to be cropped in an extended VI channel
	DstW Width of the target picture
	DstH Height of the target picture
	PixFom Pixel format of the extended channel picture
	SrcRat Source frame rate
	DstRat Target frame rate
	Depth Buffer queue depth
	Comp Whether to compress data
VI FISHEYE ATTR	ViChn ID of the channel used by the fisheye
	Enable Fisheye function enable 0: disabled 1: enabled
	MntMode Mount mode of the fisheye lens
	RgnNum Number of correction regions
	BgEnable Background color enable 0: disabled 1: enabled
	BgColor Background color Value range: [0, 0xFFFF]
	LMF Enable for using the LMF parameters of the fisheye lens configured by the user 0: not used 1: used
	HOffset Horizontal offset of the picture center point relative to the physical center point
	VOffset Vertical offset of the picture center point relative to the physical center point
	TCoef Strength coefficient for trapezoid correction



Parameter	Description	
VI FISHEYE Region Attr	ExtChn	ID of the channel used by the fisheye
	RgnIndex	ID of the correction region Value range: [0, 3]
	ViewMode	Correction mode
	InRadius	Inside radius of the source picture corresponding to the correction region, valid only in 360° panoramic correction mode
	OutRadius	Outside radius of the source picture corresponding to the correction region
	Pan	Value of the PTZ parameter Pan for the correction region
	Tilt	Value of the PTZ parameter Tilt for the correction region
	HorZoom	Value of the PTZ parameter horizontal zoom for the correction region
	VerZoom	Value of the PTZ parameter vertical zoom for the correction region
	OutX	Horizontal coordinate (X) for the output position of the correction region
USER PIC INFO	OutY	Vertical coordinate (Y) for the output position of the correction region
	OutW	Output width of the correction region
	OutH	Output height of the correction region
	UPicID	ID of the user picture used by a channel
	Width	Width of the channel that uses a user picture
	Height	Height of the channel that uses a user picture
	Stride	Stride of the channel that uses a user picture
	Field	Frame/Field select The value is fixed at frm .
	PixForm	Pixel format Value range: {sp420, sp422}
VI DIS INFO1	PoolID	ID of the pool corresponding to a user picture
	PhyAddr	Physical address of the memory for the pool corresponding to a user picture
	bUpdate	Whether to update the user picture
VI DIS INFO1	ViChn	ID of the VI channel used by the DIS. Only the



Parameter	Description	
Information related to DIS configuration and attributes (supported only by Hi3519 V100)	bEnable	physical channel is used.
	Acc	Flag bit of DIS enable
	Mode	DIS accuracy
	Fixlevel	DIS scenario mode
	Roef	DIS strength level
	BufNum	Parameter used to correct the rolling shutter
	MovSub	Number of buffers used by the DIS to store pictures
	NoMov	Level used to determine whether the object is moving
	Dof	Level used to determine whether the camera is static
	CropRatio	Multi-DOF DIS algorithm
	DISFrmRate	Cropping ratio of the DIS output picture
VI DIS INFO2 Information related to DIS configuration and attributes (supported only by Hi3519 V100)	ViChn	ID of the channel used by the DIS. Only the physical channel is used.
	TimeLag	Gyroscope parameter used to match the picture with the gyroscope data
	AngleType	Type of the lens view angle
	ViewAngle	Lens view angle
	bScale	Boolean switch that controls the scaling function
	OutW	Output image width
	OutH	Output image height
	DelayNum	u32DelayFrmNum , which indicates the number of delayed output frames Value range: [0, 1]
	RetCenter	U32RetCenterStrength , which indicates the strength of returning the picture back to the center point
	GyroWeight	Weight of dependence on the gyroscope data in mixed mode
VI DISDebug INFO	bStillCrop	Enable switch that disables the DIS function but ensures that the picture is still output based on the cropping ratio
	ViChn	Channel ID



Parameter	Description	
DIS debugging information (supported only by Hi3519 V100)	PyFrmlost	Number of frame loss times during pyramid generation
	AffFrmlost	Number of frame loss times during affine processing
	Pyflag	Flag bit of the pyramid buffer
	Srcflag	Flag bit of the source frame buffer
	Mvflag	Flag bit of the motion data buffer
	PyTime	Pyramid processing period
	AfTime	Affine processing period
	MvTime	Period for generating the motion data
	MaxPyTime	Maximum pyramid processing period
	MaxAfTime	Maximum affine processing period
	MaxMvTime	Maximum period for generating the motion data
VI CHN STATUS	ViChn	VI channel ID
	bEnUsrP	Whether to enable the function of inserting a user picture
	FrmTime	Video frame interval (in millisecond)
	FrmRate	Capture frame rate of a VI channel
	SendCnt	Number of times that a VI channel transmits captured pictures
	SwLost	Number of times that a VI channel discards frames when transmitting pictures
	Rotate	Rotation angle (90°, 180°, or 270°)
	Depth	Buffer queue depth
VI CHN CALL VGS STATUS 1 Information when the VI channel calls the VGS (Usr corresponds to user pictures and Cvr corresponds to COVEREX_RGN)	ViChn	VI channel ID
	UsrBgnNOK	Number of times that the VGS begin job fails to be called
	UsrCancel	Number of times that the cancel job is called
	UsrEndOk	Number of times that VGS jobs are successfully submitted
	UsrCbOk	Number of times that the VGS is successfully called back
	CovBgnNOK	Number of times that the VGS begin job fails to be called



Parameter	Description	
	CovCancel	Number of times that the cancel job is called
	CovEndOk	Number of times that VGS jobs are successfully submitted
	CovCbOk	Number of times that the VGS is successfully called back
VI CHN CALL VGS STATUS 2 Information when the VI channel calls the VGS (Osd corresponds to OVERLAYEX_RG N and Scl indicates scaling)	ViChn	VI channel ID
	OsdBgnNOK	Number of times that the VGS begin job fails to be called
	OsdCancel	Number of times that the cancel job is called
	OsdEndOk	Number of times that VGS jobs are successfully submitted
	OsdCbOk	Number of times that the VGS is successfully called back
	ScaleNOK	Number of times that the VGS begin job fails to be called
	SclCancel	Number of times that the cancel job is called
	SclEndOk	Number of times that VGS jobs are successfully submitted
	SclCbOk	Number of times that the VGS is successfully called back
VI CHN CALL VGS STATUS 3 Information when the VI channel calls the VGS (Rotate indicates rotation and LDC indicates distortion correction)	ViChn	VI channel ID
	RotateNOK	Number of times that the VGS begin job fails to be called
	RotCancel	Number of times that the cancel job is called
	RotEndOk	Number of times that VGS jobs are successfully submitted
	RotCbOk	Number of times that the VGS is successfully called back
	LDCNOK	Number of times that the VGS begin job fails to be called
	LDCCancel	Number of times that the cancel job is called
	LDCEndOk	Number of times that VGS jobs are successfully submitted
	LDCCbOk	Number of times that the VGS is successfully called back



12.13 VO

[Debugging Information]

```
# cat /proc/umap/vo
```

```
[VOU] Version: [Hi3516A_MPP_V1.0.0.0 B010 Debug], Build Time[Nov 27 2014,  
16:47:20]
```

-----DEV CONFIG-----

DevId	DevEn	Mux1	Mux2	Mux3	InfSync	BkClr	DevFrt
0	Y	BT1120			1080P@60	ff0000	60

-----MODULE PARAM-----

detectCycle	transparentTransmit	lowPowerMode
0	N	Y

-----DEV VDAC STATUS-----

DevId	VDAC
0	N

-----VIDEO LAYER STATUS-----

LayerId	VideoEn	ClustMode	PixFmt	ImgW	ImgH	DispW	DispH	DispFrt
DoubFrm	Toleration	Priority						
0	Y	N	SP420	1920	1080	1920	1080	60
10000000		0						N

-----VIDEO LAYER STATUS 2-----

layerId	VideoEn	EnChNum	Matrix	Luma	Cont	Hue	Satu
0	Y	4	0	50	50	50	50

-----VIDEO LAYER STATUS 3-----

layerId	DevId	SetBeg	SetEnd	PartitionMode	bSDVgsBypass	u32BufLen
0	0	N	N	Single	N	3

-----CHN BASE INFO -----

LayerId	ChnId	ChnEn	Prio	DefLk	ChnX	ChnY	ChnW	ChnH	DispX	DispY	bSnap	Field
0	0	Y	0	N	0	0	960	540	-1	-1	N	both
0	1	Y	0	N	960	0	960	540	-1	-1	N	both
0	2	Y	0	N	0	540	960	540	-1	-1	N	both
0	3	Y	0	N	960	540	960	540	-1	-1	N	both

-----CHN PLAY INFO 1-----

LayerId	ChnId	Batch	Show	Pause	Step	Revrs	Refsh	Thrshd	ChnFrt	ChnGap
---------	-------	-------	------	-------	------	-------	-------	--------	--------	--------



0	0	N	Y	N	N	N	N	3	60	16666
0	1	N	Y	N	N	N	N	3	60	16666
0	2	N	Y	N	N	N	N	3	60	16666
0	3	N	Y	N	N	N	N	3	60	16666

-----CHN PLAY INFO 2-----

LayerId	ChnId	DisplayPts			PrePts			CurrPts		
		ScalePts		SetPts	RecvCurPts					
		0	0	1352807005	0	1352807005	1352807005			
0	0	-1	0	0	1352807005	1352807005	1352807005	0	1352807005	
0	1	-1	0	0	1352807005	1352807005	1352807005	0	1352807005	
0	2	-1	0	0	1352807005	1352807005	1352807005	0	1352807005	
0	3	-1	0	0	1352807005	1352807005	1352807005	0	1352807005	

-----BySingle CHN STATUS 1-----

LayerId	ChnId	Job	Task	LCnt	SCnt	ChRpt	DRpt	CBusy		DBusy	
								ShouD	Dspd	b2Scl	ChnAddr
0	0	0	158	0	158	0	161	0	1	0	1
1	N 87942e00	8c06d000									
0	1	0	158	0	158	0	161	0	1	0	1
1	N 8b204800	8c06d000									
0	2	0	158	0	158	0	161	0	1	0	1
1	N 8b2cf000	8c06d000									
0	3	0	158	0	158	0	161	0	1	0	1
1	N 8b399800	8c06d000									

-----BySingle CHN STATUS 2-----

LayerId	ChnId	bBorder	BorderWidth	Color	ChnFreeNum	ChnBusyNum	0	1	DisplayFreeNum		DisplayBusyNum	
									DisplayFreeNum	DisplayBusyNum	0	1
1	0	0	Y	6	ff	7	0	13	0	1	0	13
0	1	0	Y	6	ff	7	0	13	0	1	0	13
1	0	2	Y	6	ff	7	0	13	0	1	0	13
0	3	0	Y	6	ff	7	0	13	0	1	0	13

-----CHN OTHER INFO-----

LayerId	ChnId	bZoom	ZmTyp	ZoomX	ZoomY	ZoomW	ZoomH	SrcW	SrcH
0	0	N	0	0	0	0	0	1920	1080



0	1	N	0	0	0	0	0	720	576
0	2	N	0	0	0	0	0	720	576
0	3	N	0	0	0	0	0	720	576

-----LAYER CSC PARAM-----

LAYERID	Matrix	Luma	Cont	Hue	Satu
0	5	50	50	50	50

[Analysis]

This section records the usage and attributes of the VOU, including the status of the device, video layer, and channel. By checking the debugging information, you can dynamically obtain the current status of the VOU, which facilitates debugging or testing.

[Parameter Description]

Parameter		Description
DEV CONFIG	DevId	VO device ID Value range: [0, VO_MAX_DEV_NUM)
	DevEn	Whether to enable the VOU N: disabled Y: enabled
	Mux1/Mux2/Mux3	Interface type Value range: Hi3516A: CVBS, BT656, BT1120 Hi3518E V200: BT656, LCD_6BIT, LCD_8BIT Hi3519 V100: CVBS, BT656, BT1120, LCD_6BIT, LCD_8BIT, LCD_16BIT
	InfSync	Interface timing Value range: [0, VO_OUTPUT_BUTT)
	BkClr	Background color of a device, in RGB888 format
	DevFrt	Frame rate of a device, that is, refresh rate. The frame rate is related to the timing.
MODULE PARAM	detectCycle	Module parameter, indicating the number of load detection cycles. The value must be greater than or equal to 0. The value 0 indicates that the load is not detected. If detectCycle is greater than 0 but less than 10, the number of load detection cycles is set to the default value 30 .
	transparentTransmit	Module parameter, indicating whether YC transparent transmission is enabled when ko is loaded. The value range is [0, 1]. The default value is 0. The displayed proc



Parameter		Description
		information is described as follows: N: disabled Y: enabled
	lowPowerMode	Module parameter, indicating whether the low-power consumption mode is enabled when the ko is loaded. The value range is [0, 1]. The default value is 1. This parameter applies to the startup screen. To use the startup image, this parameter should be set to 0. Otherwise, the startup screen disappears after ko is loaded. The displayed proc information is described as follows: N: disabled. The VO clock is enabled after ko is loaded. Y: enabled. The VO clock is disabled after ko is loaded.
DEV VDAC STATUS	DevId	VO device ID Value range: [0, VO_MAX_DEV_NUM)
	VDAC	Whether to enable the DAC for the device Y: enabled N: disabled
VIDEO LAYER STATUS	LayerId	Video layer ID Value range: [0, VO_MAX_LAYER_NUM)
	VideoEn	Video layer enable N: disabled Y: enabled
	ClustMode	Whether the video layer memory is used in cluster mode N: non-cluster mode Y: cluster mode
	PixFmt	Pixel format of an input picture Hi3516A: SP422 and SP420 Hi3518E V200/Hi3519 V100: SP422, SP420, and YUV400
	ImgW	Width of a video layer canvas
	ImgH	Height of a video layer canvas
	DispW	Width of a displayed area
	DispH	Height of a displayed area
	DispFrt	Display frame rate of video layer
	DoubFrm	Whether the frame rate of a video layer is doubled N: not doubled Y: doubled



Parameter		Description
	Toleration	Play control tolerance of a video layer
	Priority	Display priority of an HD video layer
VIDEO LAYER STATUS 2	LayerId	Video layer ID Value range: [0, VO_MAX_LAYER_NUM].
	VideoEn	Video layer enable N: disabled Y: enabled
	EnChNum	Number of enabled channels at a video layer Value range: [0, VO_MAX_CHN_NUM]
	Matrix	Color space conversion (CSC) matrix select Value range: Hi3516A: [0, 2] Hi3518E V200/Hi3519V100: [0, 4]
	Luma	Luminance The value ranges from 0 to 100.
	Cont	Contrast The value ranges from 0 to 100.
	Hue	Hue The value ranges from 0 to 100.
	Satu	Saturation The value ranges from 0 to 100.
VIDEO LAYER STATUS 3	LayerId	Video layer ID Value range: [0, VO_MAX_LAYER_NUM)
	DevId	ID of the device bound to a video layer Value range: [0, VO_MAX_DEV_NUM)
	SetBeg	Whether the batch processing start status is enabled N: no Y: yes
	SetEnd	Whether the batch processing end status is enabled N: no Y: yes
	PartitionMode	Partition mode (single-partition or multi-partition)
	bSDVgsBypass	Bypass mode flag N: no bypass Y: bypass



Parameter		Description
	u32BufLen	Display buffer length
CHN BASE INFO	LayerId	Video layer ID Value range: [0, VO_MAX_LAYER_NUM).
	ChnId	VO channel ID Value range: [0, VO_MAX_CHN_NUM)
	ChnEn	Channel enable N: disabled Y: enabled
	Prio	Channel priority Value range: [0, VO_MAX_CHN_NUM)
	DeFlk	Whether to perform anti-flicker. On the interlaced device, when a single-field picture or a frame is zoomed out, the anti-flicker function needs to be enabled. In other cases, anti-flicker function is disabled. On the progressive output device, the anti-flicker function is disabled. N: disabled Y: enabled
	ChnX	Start horizontal coordinate of a channel
	ChnY	Start vertical coordinate of a channel
	ChnW	Channel width
	ChnH	Channel height
	DispX	Horizontal coordinate of the channel display position on the PIP layer of the HD device. This parameter is invalid for the channel on the PIP layer of the non-HD device.
CHN PLAY INFO1	DispY	Vertical coordinate of the channel display position on the PIP layer of the HD device. This parameter is invalid for the channel on the PIP layer of the non-HD device.
	bSnap	Whether to enable channel snapshot N: no Y: yes
	Field	Frame/Field information. This item is used to set the frame/field flag of a picture. Value range: [0, VO_FIELD_BUTT)
	LayerId	Video layer ID Value range: [0, VO_MAX_LAYER_NUM).
	ChnId	VO channel ID Value range: [0, VO_MAX_CHN_NUM)



Parameter	Description
Batch	Whether the channel processes data in batches N: no Y: yes
Show	Whether to show a channel N: hide Y: show
Pause	Whether to pause a channel N: disabled Y: enabled
Step	Whether to step N: disabled Y: enabled
Revs	Whether to enable reverse playing N: disabled Y: enabled
Refsh	Channel refresh enable N: disabled Y: enabled
Thrshd	Channel display threshold For the HD device, the threshold is the maximum number of frames to be displayed in the channel. For the SD device, the threshold is the maximum number of frames to be received in the channel buffer queue.
ChnFrt	Frame rate of a channel. This value reflects the playing and control mode of a channel.
ChnGap	Frame gap of channel (in μ s). The gap is in inverse proportion to the frame rate of the channel (in μ s).
CHN PLAY INFO2	LayerId Value range: [0, VO_MAX_LAYER_NUM).
	ChnId Value range: [0, VO_MAX_CHN_NUM)
	DisplayPts PTS of the frame being displayed (in μ s)
	PrePts PTS of the picture of the node at the end of the channel busy queue (in μ s)
	CurrPts PTS of the pictures that are queried by the VPSS (in μ s)
	ScalePts Target PTS, that is, PTS of the next frame (in μ s)
	SetPts Size of HD picture switching and PTS OF the last old



Parameter		Description
		frame (in μ s)
	RecvCurPts	Original PTS of the latest received picture
BySingle CHN STATUS 1	LayerId	Video layer ID Value range: [0, VO_MAX_LAYER_NUM].
	ChnId	VO channel ID Value range: [0, VO_MAX_CHN_NUM]
	Job	Number of tasks for the VGS. This value indicates the number of combination tasks that are being handled by the VGS.
	Task	Number of tasks in the VGS.
	LCnt	Number of times that the channel discards the frames sent by other modules.
	SCnt	Number of times that the channel receives frames sent by other modules. If the value is 0, the VO channel does not receive pictures; if the value does not increase, the VO channel cannot receive pictures.
	ChRpt	Number of times that the channel displays repeated pictures. If the channel is paused, this value continuously increases.
	DRpt	Number of times that spliced pictures are displayed repeatedly
	CBusy	Number of nodes in the channel busy queue
	DBusy	Number of displayed busy nodes in the queue
	ShouD	Number of times that the current picture should be displayed
	Dsped	Number of times that the current picture has been displayed. If the channel is paused, this value increases continuously.
	b2Scl	Whether the displayed picture in the channel is processed by using level-2 scaling Y: yes N: no
BySingle CHN STATUS 2	ChnAddr	Physical address for the current picture in the channel
	DispAddr	Physical address for the picture that is displayed at the video layer
	LayerId	Video layer ID Value range: [0, VO_MAX_LAYER_NUM)
	ChnId	VO channel ID



Parameter	Description
	Value range: [0, VO_MAX_CHN_NUM)
bBorder	Border enable Y: yes N: no
BorderWidth	Border width Value range: [2, 8]
Color	Border color
ChnFreeNum	Number of idle nodes in the channel queue
ChnBusyNum	Number of used nodes in the channel queue
DisplayFreeNum	Number of displayed idle nodes in the queue
DisplayBusyNum	Number of displayed nodes being used in the queue
CHN OTHER INFO	LayerId Value range: [0, VO_MAX_LAYER_NUM).
	ChnId Value range: [0, VO_MAX_CHN_NUM)
	bZoom Whether to perform partial enlargement N: no Y: yes
	ZmTyp Partial enlargement type, enlargement by region or ratio Value range: [0, VOU_ZOOM_IN_BUTT)
	ZoomX Start horizontal coordinate of a partially enlarged region. If the partially enlarged region is specified by ratio, the value is the coordinate of the source picture after processing by ratio and alignment.
	ZoomY Start vertical coordinate of a partially enlarged region. If the partially enlarged region is specified by ratio, the value is the coordinate of the source picture after processing by ratio and alignment.
	ZoomW Width of a partially enlarged region. If the partially enlarged region is specified by ratio, the value is the coordinate of the source picture after processing by ratio and alignment.
	ZoomH Height of a partially enlarged region. If the partially enlarged region is specified by ratio, the value is the coordinate of the source picture after processing by ratio and alignment.
	SrcW Width of the source channel picture



Parameter		Description
	SrcH	Height of the source channel picture
LAYER CSC PARAM	LAYERID	ID of the graphics layer bound to a device Value range: [0, VOU_GRAPHICS_LAYER_NUM)
	Matrix	CSC matrix select at the graphics layer Value range: [5, 6]
	Luma	Luminance Value range: [0, 100]
	Cont	Contrast Value range: [0, 100]
	Hue	Hue Value range: [0, 100]
	Satu	Saturation Value range: [0, 100]

12.14 VPSS

[Debugging Information]

```
# cat /proc/umap/vpss
```

```
[VPSS] Version: [Hi3516A_MPP_V1.0.0.0 B010 Debug], Build Time[May 21 2015,  
10:14:29]
```

```
VI-VPSS is online.
```

```
-----MODULE PARAM-----
```

```
rfr_frame_comp bOneBufferforLowDelay
```

```
Y Y
```

```
-----VPSS GRP ATTR-----
```

GrpID	MaxW	MaxH	PixFmt	DieMode	DciEn	NrEn	HistEn
0	720	576	SP420	node	N	Y	N

```
-----VPSS GRP FRAME CONTROL INFO-----
```

```
GrpID SrcFRate DstFRate
```

```
-----VPSS GRP PARAM-----
```

GrpID	bUsing	Cont	GlobStr	IeStr	YSfStr	YTfStr	CSfStr	CTfStr
-------	--------	------	---------	-------	--------	--------	--------	--------

```
MotionLimen
```

0	N	32	128	-1	-1	-1	-1	-1
---	---	----	-----	----	----	----	----	----



```
-----VPSS GRP PARAM V2-----
GrpID bUsing CSf CTf IEPost IEStr YMth YSf_MA YSf_SA YTf DeSend
      0     Y     8     0     0    21     64     32     32    12     0

-----VPSS GRP 3DNR PARAM -----
GrpID bUsing YPKStr YSFStr YTFSRt TFStrMax TFStrMov YSmthStr
      0     N     0    190     125     12     0
      110
YSmthRat YSFStrDlt YSFStrDl YTFSRtDlt YTFSRtDl YSFBRt Rat CSFStr
      16     0     0     25     20     64     40
CTFstr YTFSRtDlt
      15     0

-----VPSS CHN PARAM-----
GrpID ChnID ChnSp
      0     0     0

-----VPSS CHN ATTR-----
GrpID PhyChnID Enable SpEn FrmWkEn MirrorEn FlipEn SrcFRate
      0     0     Y     N     N     N     N     -1
DstFRate OverlayMask CoverMask ChnNr
      -1     0     ff     Y

-----VPSS PHY CHN MODE-----
GrpID PhyChnID WorkMode Width Height Double Pixfmt CompressMode
      0     0     USER   1920   1080     N     SP420     0
      0     1     USER   1920   1080     N     SP420     0
      0     2     USER   1920   1080     N     SP420     0
      0     3     USER   1920   1080     N     SP420     0

-----VPSS EXT-CHN ATTR-----
GrpID ExtChnID Enable SrcChn Width Height Srcfrc Dstfrc Pixfmt

-----VPSS EXT-CHN CROP ATTR-----
GrpID ExtChnID CropEn CoorType CoorX CoorY Width Height
      0     4     Y     RIT     0     0     500     500

-----VPSS GRP CROP INFO-----
GrpID CropEn CoorType CoorX CoorY Width Height OriW OriH
      0     Y     RIT     0     0     0     0     720     576
TrimX TrimY TrimWid TrimHgt
      0     0     720     576

-----VPSS CHN CROP INFO-----
GrpID ChnID CropEn CoorType CoorX CoorY Width Height TrimX
```



TrimY TrimWid TrimHgt

-----VPSS GRP PIC QUEUE-----

GrpID	FreeLen	BusyLen	Delay	Backup
0	7	0	0	Y

-----VPSS GRP WORK STATUS-----

GrpID	RecvPic	ViLost	VdecLost	NewDo	OldDo	NewUndo
0	2842	0	0	2842	0	0
OldUndo	NoHist	StartFl	bStart	CostTm	MaxCostTm	
7091	0	0	Y	148822	166473	

-----VPSS CHN WORK STATUS-----

GrpID	ChnID	WorkMode	Depth	SendOk	bConfident	bDouble
0	0	AUTO	0	2840	Y	N

-----VPSS CHN OUTPUT RESOLUTION-----

GrpID	ChnID	Enable	Width	Height	PixFmt	CompressMode	FrameRate
0	0	Y	640	360	SP420	0	60

-----VPSS NR PARAM-----

GrpID	REF	Compress	CurNRInt	CurNRVer
-------	-----	----------	----------	----------

-----VPSS CHN ROTATE INFO-----

GrpID	ChnID	Rotate
-------	-------	--------

-----VPSS CHN LDC ATTR-----

GrpID	ChnID	Enable	ViewType	Xoffset	Yoffset	Ratio
-------	-------	--------	----------	---------	---------	-------

-----VPSS CHN FISHEYE ATTR-----

GrpID	ChnID	Enable	MntMode	RgnNum	BgEnable	BgColor	LMF	HOffset	VOffset
0	4	Y	Desktop	1	N	0xfc	N	0	0

TCoef

0

-----VPSS CHN FISHEYE REGION ATTR-----

GrpID	ChnID	RgnIndex	ViewMode	InRadius	OutRadius	Pan	Tilt	HorZoom
0	4	0	360	2	960	180	180	4095
VerZoom	OutX	OutY	OutW	OutH				
4095	0	0	1920	1080				

-----VPSS CHN LOWDELAY ATTR-----

GrpID	ChnID	Enable	LineCnt	OneBufEnable	OneBufAddr
-------	-------	--------	---------	--------------	------------

-----TIMER WORK STATUS-----

CntPerSec	MaxCntPerSec	CostTm	MostCostTm	CostTmPerSec	MCostTmPerSec
101	101	7	116	2440	2536

-----DRV WORK STATUS-----



StartSuc0	StartSuc1	LinkInt	NodeInt	StartErr0
2840	0	2840	0	0
NodeIdErr0	StartErr1	NodeIdErr1	BusErr	
0	0	0	0	

-----DRV ONLINE INT STATUS-----

FrmCntPerSec	FrmStartCnt	FrmErrCnt	FrmFlowCnt	Chn0LowDelayCnt
30	21429	0	0	0
Chn1LowDelayCnt	Chn2LowDelayCnt			
0	0			

-----DRV NODE QUEUE-----

FreeNum	WaitNum	Busy00	Busy01	Sel0	Busy10	Busy11	Sel1
128	0	0	0	0	0	0	0

-----INT WORK STATUS-----

CntPerSec	MaxCntPerSec	CostTm	MostCostTm	CostTmPerSec	MCostTmPerSec
50	54	8	65	684	82

[Analysis]

This section records the current attributes and status of the VPSS.

[Parameter Description]

Parameter	Description	
VPSS MODULE PARAM	rfr_frame_com p	Whether the NR reference frame is compressed Y: compressed. The required bandwidth is reduced. N: non-compressed. The bandwidth is required. Note: For the Hi3516A, the compressed mode requires one more frame buffer than the non-compressed mode.
	bOneBufferfor LowDelay	Single-buffer mode enable in online low-delay mode N: disabled Y: enabled
VPSS GRP ATTR (including the configured maximum processing capability and enable status of each functional module.)	GrpID	Group ID Value range: [0, VPSS_MAX_GRP_NUM)
	MaxW	Maximum process width Value range: [VPSS_MIN_IMAGE_WIDTH, VPSS_MAX_IMAGE_WIDTH]
	MaxH	Maximum process height Value range: [VPSS_MIN_IMAGE_HEIGHT, VPSS_MAX_IMAGE_HEIGHT]



Parameter	Description	
	PixFmt	Pixel format. The formats semi-planar420, Border width2, and YUV400 single component are supported.
	DieMode	Deinterlace mode 0: adaptation 1: do not perform de-interlace (DEI) forcibly 2: perform DEI forcibly
	DciEn	Dynamic contrast improvement (DCI) enable N: disabled Y: enabled
	NrEn	Noise reduction enable N: disabled Y : enabled
	HistEn	Histogram output N: disabled Y: enabled
VPSS GRP FRAME CONTROL INFO	GrpID	Group ID Value range: [0, VPSS_MAX_GRP_NUM)
	SrcFRate	Source frame rate
	DstFRate	Target frame rate
VPSS GRP PARAM	GrpID	Group ID Value range: [0, VPSS_MAX_GRP_NUM - 1]
	bUsing	Whether the parameter is being used N: not used Y: used
	Cont	Contrast (not used currently)
	GlobStr	Overall 3DNR strength Value range: [0, 1408]
	IeStr	Strength of picture texture NR Value range: [-1, +100]
	YSfStr	Strength of luminance spatial-domain NR Value range: [-1, +9999]
	YTfStr	Strength of luminance time domain NR Value range: [-1, +15]
	CSfStr	Strength of chrominance spatial-domain NR Value range: [-1, +255]



Parameter	Description	
	CTfStr	Strength of chrominance time domain NR Value range: [-1, +32]
	MotionLimn	Motion threshold Value range: [-1, +511]
VPSS GRP PARAM V2	GrpID	Group ID Value range: [0, VPSS_MAX_GRP_NUM]
	bUsing	Whether the parameter is being used N: not used Y: used
	CSf	Strength of chrominance spatial-domain filtering Value range: [0, 255]
	CTf	Strength of chrominance time-domain filtering Value range: [0, 32]
	IEpst	Flag indicating whether texture/edge enhancement is performed before or after other processing Value range: [0, 1]
	IEstr	Texture/Edge enhancement strength Value range: [0, 63]
	YMth	Threshold of luminance motion detection Value range: [0, 511]
	YSf_MA	Strength of luminance spatial-domain filtering for the motion area Value range: [0, 255]
	YSf_SA	Strength of luminance spatial-domain filtering for the static area Value range: [0, 64]
	YTf	Strength of luminance time-domain filtering Value range: [0, 15]
VPSS 3DNR PARAM VPSS	DeSend	Degranulation strength Value range: [0, 8]
	GrpID	Group ID Value range: [0, VPSS_MAX_GRP_NUM - 1]
	bUsing	Whether the parameter is being used N: not used Y: used
	YPKStr	Texture enhancement



Parameter	Description	
	YSFStr	Value range: [0, 63]
	YTFStr	Strength of main spatial-domain filtering Value range: [0, 200]
	TFStrMax	Strength of main time-domain filtering Value range: [0, 128]
	TFStrMov	Upper limit for the strength of main time-domain filtering Value range: [0, 15]
	YSmthStr	Strength of temporal filtering in the motion region Value range: [0, 31]
	YSmthRat	Strength of smooth filtering Value range: [0, 200]
	YSFStrDlt	Relative strength of smooth filtering Value range: [0, 32]
	YSFStrDl	Strength 1 of auxiliary spatial-domain filtering Value range: [-128, +127]
	YTFStrDlt	Strength 2 of auxiliary spatial filtering Value range: [0, 255]
	YTFStrDl	Strength 1 of auxiliary time-domain filtering Value range: [-64, +63]
	YTFStrDl	Strength 2 of auxiliary time-domain filtering Value range: [0, 31]
	YSFBriRat	Relative strength of spatial-domain filtering for bright regions Value range: [0, 64]
	CSFStr	Strength of color difference spatial-domain filtering Value range: [0, 80]
	CTFstr	Strength of color difference time-domain filtering Value range: [0, 32]
	YTFMdWin	MD window for luminance temporal filtering Value range: [0, 1] Default value: 1 The value 0 indicates a small window and the value 1 indicates a large window.
VPSS CHN PARAM	GrpID	Group ID Value range: [0, VPSS_MAX_GRP_NUM - 1]



Parameter	Description	
	ChnID	Channel ID Value range: [0, 3]
	ChnSp	Channel sharpening (SP) strength
VPSS CHN ATTR	GrpID	Group ID Value range: [0, VPSS_MAX_GRP_NUM - 1]
	PhyChnID	Physical channel ID Value range: [0, 3]
	Enable	Channel enable N: disabled Y: enabled
	SpEn	SP enable N: disabled Y: enabled
	FrmWkEn	Border enable N: disabled Y: enabled
	MirrorEn	Mirror enable N: disabled Y: enabled
	FlipEn	Flip enable N: disabled Y: enabled
	SrcFRate	Source frame rate for channel frame rate control
	DstFRate	Target frame rate for channel frame rate control
	OverlayMask	Enable for VPSS channel video overlay regions. The lower eight bits are valid, and each bit is used to enable or disable a region. The region with a lower priority is overlaid with the region with a higher priority by default.
	CoverMask	Enable for the VPSS channel video cover regions. The lower eight bits are valid, and each bit is used to enable or disable a region. The region with a higher priority covers the region with a lower priority by default.
	ChnNr	Channel denoising enable Y: enabled N: disabled



Parameter	Description	
VPSS PHY CHN MODE	GrpID	Group ID Value range: [0, VPSS_MAX_GRP_NUM)
	PhyChnID	Physical channel ID Value range: [0, 3]
	WorkMode	Channel working mode
	Width	Width of the target picture to be scaled in the channel
	Height	Height of the target picture to be scaled in the channel
	Double	Whether the channel is using the field/frame conversion function
	Pixfmt	Target pixel format of the channel
VPSS EXT-CHN ATTR	CompressMode	Compression mode for the output picture of the channel 0: uncompressed 1: compressed
	GrpID	Group ID Value range: [0, VPSS_MAX_GRP_NUM - 1]
	ExtChnID	ID of an extended VPSS channel Value range: [4, 11]
	Enable	Extended channel enable N: disabled Y: enabled
	SrcChn	Source channel of an extended channel Value range: [0, 3]
	Width	Target output width of an extended channel
	Height	Target output height of an extended channel
	Srcfrc	Source frame rate for controlling the frame rate of an extended channel
	Dstfrc	Target frame rate for controlling the frame rate of an extended channel
VPSS EXT-CHN CROP ATTR	Pixfmt	Output pixel format of an extended channel
	GrpID	Group ID Value range: [0, VPSS_MAX_GRP_NUM-1]
	ExtChnID	ID of an extended VPSS channel Value range: [4, 11]



Parameter	Description	
	CropEn	Cropping enable N: disabled Y: enabled
	CoorType	Coordinate type RIT: relative coordinates ABS: absolute coordinates
	CoorX	Horizontal start coordinates When the coordinates are relative coordinates, the value range is [0, 999]. When the coordinates are absolute coordinates, the value range is [0, VPSS_MAX_IMAGE_WIDTH].
	CoorY	Vertical start coordinates When the coordinates are relative coordinates, the value range is [0, 999]. When the coordinates are absolute coordinates, the value range is [0, VPSS_MAX_IMAGE_HEIGHT].
	Width	Width of the rectangular region to be cropped, which cannot exceed the maximum picture width
	Height	Height of the rectangular region to be cropped, which cannot exceed the maximum picture height
VPSS GRP CROP INFO	GrpID	Group ID Value range: [0, VPSS_MAX_GRP_NUM-1]
	CropEn	Crop enable N: disabled Y: enabled
	CoorType	Coordinate type RIT: relative coordinate ABS: absolute coordinate
	CoorX	Horizontal start coordinate When the coordinate type is relative coordinate, the valid value range is [0, 999]. When the coordinate type is absolute coordinate, the valid value range is [0, VPSS_MAX_IMAGE_WIDTH].
	CoorY	Vertical start coordinate When the coordinate type is relative coordinate, the valid value range is [0, 999]. When the coordinate type is absolute coordinate, the valid value range is [0,



Parameter	Description
VPSS CHN CROP INFO	VPSS_MAX_IMAGE_HEIGHT].
	Width Width of CROP RECT. It cannot exceed the maximum picture width.
	Height Height of CROP RECT. It cannot exceed the maximum picture height.
	OriW Width of the original picture
	OriH Height of the original picture
	TrimX Horizontal coordinate of the actual picture start point
	TrimY Vertical coordinate of the actual picture start point
	TrimWid Actual picture width
	TrimHgt Actual picture height
VPSS CHN CROP INFO	GrpID Group ID Value range: [0, VPSS_MAX_GRP_NUM-1]
	ChnID Channel ID Value range: [1, 3]
	CropEn Crop enable N: disabled Y: enabled
	CoorType Coordinate type RIT: relative coordinate ABS: absolute coordinate
	CoorX Horizontal start coordinate When the coordinate type is relative coordinate, the value range is [0, 999]. When the coordinate type is absolute coordinate, the value range is [0, VPSS_MAX_IMAGE_WIDTH].
	CoorY Vertical start coordinate When the coordinate type is relative coordinate, the valid value range is [0, 999]. When the coordinate type is absolute coordinate, the value range is [0, VPSS_MAX_IMAGE_HEIGHT].
	Width Width of the cropped rectangle, which cannot exceed the maximum picture width
	Height Height of the cropped rectangle, which cannot exceed the maximum picture height



Parameter	Description	
	TrimX	Horizontal coordinate of the actual picture start point
	TrimY	Vertical coordinate of the actual picture start point
	TrimWid	Actual picture width
	TrimHgt	Actual picture height
VPSS GRP PIC QUEUE	GrpID	Group ID Value range: [0, VPSS_MAX_GRP_NUM-1]
	FreeLen	Number of nodes that can be used
	BusyLen	Number of nodes that can be used
	Delay	Delay queue length
	Backup	Whether to enable backup frames for a VPSS channel
VPSS GRP WORK STATUS	GrpID	Group ID Value range: [0, VPSS_MAX_GRP_NUM-1]
	RecvPic	Number of pictures the VPSS receives
	ViLost	Number of discarded VI pictures due to full queue
	VdecLost	Number of discarded VDEC pictures due to full queue
	NewDo	Number of NewDo times
	OldDo	Number of OldDo times
	NewUnDo	Number of NewUnDo times
	OldUnDo	Number of OldUnDo times
	NoHist	Number of times that the histogram buffer cannot be applied
	StartFl	Number of times that the start task fails
	bStart	Whether to receive pictures
	CostTm	Current time for completing tasks
	MaxCostTm	Historical maximum time for completing tasks
VPSS CHN WORK STATUS	GrpID	Group ID Value range: [0, VPSS_MAX_GRP_NUM-1]
	ChnID	Channel ID Value range: [0, 3]
	WorkMode	Channel working mode



Parameter	Description	
	Depth	User queue depth
	SendOk	Number of times that the VPSS transmits pictures successfully
	bConfident	Whether the backend picture processing requirement matches the channel capability
	bDouble	Whether field/frame conversion is being used in the channel
VPSS CHN OUTPUT RESOLUTION	GrpID	Group ID Value range: [0, VPSS_MAX_GRP_NUM)
	ChnID	Channel ID Value range: [0, 11]
	Enable	Channel enable N: disabled Y: enabled
	Width	Target picture width (in pixel)
	Height	Target picture height (in pixel)
	Pixfmt	Pixel format of the target picture
	CompressMode	Compression mode 0: uncompressed 1: compressed
	FrameRate	Real-time output frame rate of the channel
	GrpID	Group ID Value range: [0, VPSS_MAX_GRP_NUM-1]
VPSS NR PARAM	REF	NR reference frame source
	Compress	Whether to compress reconstruction frames 0: uncompressed 1: compressed
	CurNRInt	Type of the 3NDR interface being used currently S: standard interface B: advanced interface
	CurNRVer	Version of the 3NDR interface being used currently V1: VPSS_NR_V1 V2: VPSS_NR_V2 V3: VPSS_NR_V3
	VPSS CHN	GrpID



Parameter	Description	
ROTATE INFO		Value range: [0, VPSS_MAX_GRP_NUM-1]
	ChnID	Channel ID Value range: [0, 3]
	Rotate	Rotation angle enumeration
VPSS CHN LDC ATTR	GrpID	Group ID Value range: [0, VPSS_MAX_GRP_NUM-1]
	ChnID	Channel ID Value range: [0, 3]
	Enable	LDC enable N: disabled Y: enabled
	ViewType	LDC mode
	Xoffset	Horizontal offset coordinate of the distortion center
	Yoffset	Vertical offset coordinate of the distortion center
	Ratio	Correction degree
VPSS CHN FISHEYE ATTR	GrpID	Group ID Value range: [0, VPSS_MAX_GRP_NUM)
	ChnID	Channel ID Value range: [4, 11]
	Enable	Fisheye function enable N: disabled Y: enabled
	MntMode	Mount mode of fisheye correction
	RgnNum	Number of correction regions
	BgEnable	Background color enable N: disabled Y: enabled
	BgColor	Background color Value range: [0x0, 0xFFFF]
	LMF	Enable for using the LMF coefficient of the fisheye lens configured by the user N: disabled Y: enabled
	HOffset	Horizontal offset of the picture center point relative to the physical center point



Parameter	Description	
	VOffset	Vertical offset of the picture center point relative to the physical center point
	TCoef	Strength coefficient for trapezoid correction
VPSS CHN FISHEYE REGION ATTR	GrpID	Group ID Value range: [0, VPSS_MAX_GRP_NUM)
	ChnID	Channel ID Value range: [4, 11]
	RgnIndex	ID of the fisheye correction region Value range: [0, 3]
	ViewMode	Correction mode for the correction region
	InRadius	Inside radius of the source picture corresponding to the correction region, valid only in 360° panoramic correction mode
	OutRadius	Outside radius of the source picture corresponding to the correction region
	Pan	Value of the PTZ parameter Pan for the correction region
	Tilt	Value of the PTZ parameter Tilt for the correction region
	HorZoom	Value of the PTZ parameter horizontal zoom for the correction region
	VerZoom	Value of the PTZ parameter vertical zoom for the correction region
	OutX	Horizontal coordinate (X) for the output start position of the correction region
	OutY	Vertical coordinate (Y) for the output start position of the correction region
VPSS CHN LOWDELAY ATTR	OutW	Output width of the correction region
	OutH	Output height of the correction region
	GrpID	Group ID Value range: [0, VPSS_MAX_GRP_NUM - 1]
	ChnID	Channel ID Value range: [0, 2]
Enable	Enable	Channel short delay enable N: disabled Y: enabled
	LineCnt	Line ID for short delay



Parameter	Description
	Value range: [16, VPSS_MAX_IMAGE_HEIGHT]
	OneBufEnable Single buffer mode enable in online low-delay mode
	OneBufAddr Physical address of the buffer in single buffer mode when the online low-delay mode is enabled
TIMER WORK STATUS	CntPerSec Number of times that the timer works in the last second
	MaxCntPerSec Historical maximum number of times that the timer works in one second
	CostTm Time that the timer works last time
	MostCostTm Maximum time that the timer works
	CostTmPerSec Time that the timer works in the last second
	MCostTmPerSec Historical maximum time that the timer works in one second
DRV WORK STATUS	StartSuc0 Number of times that VPSS0 is successfully started
	StartSuc1 Number of times that VPSS1 is successfully started
	LinkInt Number of times that links are interrupted
	NodeInt Number of times that nodes are interrupted
	StartErr0 Number of times that VPSS0 fails to be started
	NodeIdErr0 Number of times that an error occurs when the VPSS0 node ID is read
	StartErr1 Number of times that VPSS1 fails to be started
	NodeIdErr1 Number of times that an error occurs when the VPSS1 node ID is read
	BusErr Number of bus error interrupts
DRV ONLINE INT STATUS	FrmCntPerSec Number of frames processed in the latest second
	FrmStartCnt Number of normal frame start interrupts
	FrmErrCnt Number of frame error interrupts
	FrmFlowCnt Number of frame overflow error interrupts
	Chn0LowDelayCnt Number of short delay interrupts for channel 0
	Chn1LowDelayCnt Number of short delay interrupts for channel 1
	Chn2LowDelayCnt Number of short delay interrupts for channel 2



Parameter		Description
DRV NODE QUEUE	FreeNum	Number of available nodes in the free queue
	WaitNum	Number of nodes to be processed in the wait queue
	Busy00	Number of nodes in busy0 queue of VPSS0
	Busy01	Number of nodes in busy1 queue of VPSS0
	Sel0	ID of the busy queue of VPSS0 for operation
	Busy10	Number of nodes in busy0 queue of VPSS1
	Busy11	Number of nodes in busy1 queue of VPSS1
	Sel1	ID of the busy queue of VPSS1 for operation
INT WORK STATUS of the VPSS	CntPerSec	Number of times that the interrupt is performed in the last second
	MaxCntPerSec	Historical maximum number of times that the interrupt is performed in one second
	CostTm	Time that the interrupt is performed last time
	MostCostTm	Maximum time that the interrupt is performed
	CostTmPerSec	Time that the interrupt is performed in the last second
	MCostTmPerSec	Historical maximum time that the interrupt is performed in one second

12.15 VDA

[Debugging Information]

```
# cat /proc/umap/vda
```

```
[VDA] Version: [Hi3516A_MPP_V1.0.0.0 B0 Debug], Build Time[Oct 17 2013,  
17:01:13]
```

```
-----VDA CHN ATTR-----
```

NO.	W	H	Mode	Alg	MbSz	MbBit	RfM	BufN	Itl	BgWt	RgnN
0	368	288	0	1	1	0	0	8	4	128	1

```
-----VDA CHN ATTR 2-----
```

NO.	Rgn	X	Y	W	H	SadTh	ArTh	OcTh	UnOcTh	ObjN	Sad	Obj	Plx
0	0	0	0	368	288	100	0	0	0	128	1	1	1

```
-----VDA STATE-----
```



NO.	Rgn	BgPhy	BgSrd	RfPhy	RfStd	bScdu	bFst	OcCt	UnOc	VdaC
0	0	0	0	0	0	Y	N	0	0	30

-----VDA STATE2-----

NO.	MemPhy	MemVir	MemSz	CRfPhy	RgnIx	ViSd	VdecSd	VpssSd
UserSd		Recv	LstPicC					
0	9148b000	c7f10000	13888	8828ec00	0	0	0	210
0	42		0					

-----VDA CALL VGS STATE-----

NO.	CallCnt	JobSuc	JobFail	TaskSuc	TaskFail	EndSuc
EndFail	CbCnt	JobFinOk	JobFinFail			
0	42	42	0	42	0	42
0	42	42	0			

[Analysis]

This section records the current attributes and status of the VDA.

[Parameter Description]

Parameter		Description
VDA CHN ATTR	NO.	Channel ID
	W	Channel width (in pixel)
	H	Channel height (in pixel)
	Mode	Channel mode 0: motion detection (MD) 1: occlusion detection (OD)
	Alg	Algorithm 0: background algorithm 1: frame reference algorithm
	MbSz	Macroblock size 0: 8x8 1: 16x16
	MbBit	SAD output bits 0: 8 bits 1: 16 bits
	RfM	Reference frame mode 0: dynamic mode 1: static mode 2: user mode
	BufN	Number of MD result buffers



Parameter	Description	
	Itl	VDA interval (in frame)
	BgWt	Background algorithm refresh weight
	RgnN	Number of regions in a channel The OD channel supports multiple regions.
VDA CHN ATTR 2 The MD channel contains only one region, whereas the OD channel contains multiple regions.	NO.	Channel ID
	Rgn	Region ID
	X	Start horizontal coordinate of a region relative to the channel
	Y	Start vertical coordinate of a region relative to the channel
	W	Region width (in pixel)
	H	Region height (in pixel)
	SadTh	Region SAD alarm threshold
	ArTh	Region alarm area threshold
	OcTh	Region occlusion count alarm threshold
	UnOcTh	Threshold of allowed uncover count when the region occlusion count is collected
	ObjN	Maximum number of output OBJ regions (some parts are motion parts)
	Sad	Whether to output the macroblock SAD value 0: not output 1: output
	Obj	Whether to output OBJ regions 0: not output 1: output
	Plx	Whether to output the number of alarm pixels 0: not output 1: output
VDA STATE	NO.	Channel ID
	Rgn	Region ID
	BgPhy	Background address
	BgSrd	Background picture stride
	RfPhy	Address of the reference frame
	RfStd	Reference frame stride



Parameter	Description	
	bScdu	Whether scheduling is being performed
	bFst	Whether it is the initial scheduling
	OcCt	Region occlusion count
	UnOc	Allowed uncover count when the region occlusion count is collected
	VdaC	VDA completion count
VDA STATE2	NO.	Channel ID
	MemPhy	Start physical address for the memory occupied by the channel
	MemVir	Start virtual address for the memory occupied by the channel
	MemSz	Size of the memory occupied by the channel
	CRfPhy	Physical address for the public reference frame
	RgnIx	Region that is being scheduled
	ViSd	Number of pictures transmitted by the VI channel
	VdecSd	Number of pictures transmitted by the VDEC channel
	VpssSd	Number of pictures transmitted by the VPSS
	UserSd	Number of pictures transmitted by users
	Recv	Number of pictures received by the VDA channel
	CalDsuC	Number of times that the DSU is called
VDA CALL VGS STATE	LstPicC	Number of discarded pictures due to VDA out-of-order
	NO.	Channel ID
	CallCnt	Number of times that the VDA copies data by using the VGS
	JobSuc	Number of times that the VDA successfully submits jobs to the VGS
	JobFail	Number of times that the VDA fails to submit jobs to the VGS
	TaskSuc	Number of times that tasks are successfully added to VGS jobs
	TaskFail	Number of times that tasks fail to be added to VGS jobs
	EndSuc	Number of times that VGS jobs are successfully



Parameter	Description
	completed
EndFail	Number of times that VGS jobs fail to be completed
CbCnt	Number of callback times after VGS jobs are complete
JobFinOk	Number of times that VGS jobs are successfully completed during callback
JobFinFail	Number of times that VGS jobs fail to be completed during callback

12.16 AI

[Debugging Information]

```
# cat /proc/umap/ai
[AI] Version: [Hi3516A_MPP_V1.0.0.0 B010 Debug], Build Time[Aug 25 2014,
11:13:50]
-----AI DEV ATTR-----
AiDev WorkMod SampR BitWid ChnCnt ClkSel SoundMod PoiNum ExFlag
FrmNum
0 i2s_mas 8kHz 16bit 1 1 mono 80 1 30

-----AI DEV STATUS-----
AiDev IntCnt fifoCnt buffInt FrmTime MaxFrmTime TranLen IsrTime
MaxIsrTime CBPhy CBSIZE ROFFSet WOFFSet
0 10568 0 0 28002 32028 960 208
244 4aad3000 384 0 0

-----AI DEV EXTEND STATUS-----
AiDev enTrack bMute Volume
0 0 N 0

-----AI CHN STATUS-----
AiDev AiChn State Read Write BufFul UsrQueLost UsrFrmDepth
u32Data0 u32Data1 UserGet UserRls
0 0 enable 0 0 0 0 0
0 0 0 0

-----AI CHN RESAMPLE STATUS-----
AiDev AiChn State bResmp PoiNum InSampR OutSampR
```



0 0 enable Y 80 8kHz 16kHz

-----AI CHN VQE STATUS0-----

AiDev	AiChn	State	bVqe	workmod	RATE	PoiNum	GainVol	bAnr	bAgc
bEq	bHpf	bAec	bRnr	bHdr	bDrc	bPeq	WrFile		
0	0	enable	Y	music	48kHz	80	0 N N N N N N	Y Y Y N	

-----AI CHN VQE STATUS1-----

AiDev	AiChn	State	bAnr	bUsrmod	NrIntensity	NoiseDbThr	SpProSwi	
0	0	enable	N	N	0	0	0	

-----AI CHN VQE STATUS2-----

AiDev	AiChn	State	bAgc	bUsrmod	NoiseSupSwi	AdjustSpeed	ImproveSNR	
MaxGain	NoiseFloor	OutputMode	TargetLevel	UseHPF				
0	0	enable	N	N	0	0	0	-50
0	-40	0						

-----AI CHN VQE STATUS3-----

-AiDev	AiChn	State	bEq	100	200	250	350	500	800
1.2k	2.5k	4k	8k						
0	0	enable	N	0	0	0	0	0	0
0	0								

-----AI CHN VQE STATUS4-----

AiDev	AiChn	State	bHpf	bUsrmod	HpfFreq	
0	0	enable	N	N	80	

-----AI CHN VQE STATUS5-----

AiDev	AiChn	State	bAec	bUsrmod	CngMode	DTHnlStQTh	NrAlPsEngy	NrClnSupEngy	
AecAo	AecFail								
0	0	enable	N	N	close	0	0	0 (-1,-1)	0

-----AI CHN VQE STATUS6-----

AiDev	AiChn	State	bAec	bUsrmod	VcPrtctFrqL	VcPrtctFrqL1	EcoBndLow	
EcoBndHgh	EcoBndLow2	EcoBndHgh2						
0	0	enable	N	N	0	0	0	0
0								

-----AI CHN VQE STATUS7-----

AiDev	AiChn	State	bAec	bUsrmod	ERLBND[0]	ERLBND[1]	ERLBND[2]	
ERLBND[3]	ERLBND[4]	ERLBND[5]						
0	0	enable	N	N	0	0	0	0
0								

-----AI CHN VQE STATUS8-----



```
AiDev AiChn State bAec bUsrmod ERL[0] ERL[1] ERL[2] ERL[3] ERL[4]
ERL[5] ERL[6]
0 0 enable N N 0 0 0 0 0 0 0 0
-----AI CHN VQE STATUS9-----
AiDev AiChn State bRnr bUsrmod MaxNrLevel NsThresh NrMode
0 0 enable Y N 2 -50 0
-----AI CHN VQE STATUS10-----
AiDev AiChn State bHdr bUsrmod MaxGain MinGain MicGainStp MicGain
pCallBack
0 0 enable N N 0 0 0 0 0 0
-----AI CHN VQE STATUS11-----
AiDev AiChn State bDrc bUsrmod AtkTime RlsTime
0 0 enable Y Y 20 20
-----AI CHN VQE STATUS12-----
AiDev AiChn State bDrc bUsrmod OldLev0 OldLev1 OldLev2 OldLev3 OldLev4
0 0 enable Y Y 0 -472 -792 -960 -1280
-----AI CHN VQE STATUS13-----
AiDev AiChn State bDrc bUsrmod NewLev0 NewLev1 NewLev2 NewLev3 NewLev4
0 0 enable Y Y 0 -174 -410 -606 -1021
-----AI CHN VQE STATUS14-----
AiDev AiChn State bPeq bUsrmod BandNum Filter0 Filter1 Filter2 Filter3
Filter4 Filter5 Filter6 Filter7 Filter8 Filter9
0 0 enable Y N 2 3 3 0 0 0
0 0 0 0 0
-----AI CHN VQE STATUS15-----
AiDev AiChn State bPeq bUsrmod BandNum GaindB0 GaindB1 GaindB2 GaindB3
GaindB4 GaindB5 GaindB6 GaindB7 GaindB8 GaindB9
0 0 enable Y N 2 13 9 0 0 0
0 0 0 0 0
-----AI CHN VQE STATUS16-----
AiDev AiChn State bPeq bUsrmod BandNum Freq0 Freq1 Freq2 Freq3
Freq4 Freq5 Freq6 Freq7 Freq8 Freq9
0 0 enable Y N 2 13211 5203 0 0 0
0 0 0 0 0
-----AI CHN VQE STATUS17-----
AiDev AiChn State bPeq bUsrmod BandNum Q0 Q1 Q2 Q3
```



Q4	Q5	Q6	Q7	Q8	Q9						
0	0	enable	Y	N	2	7	9	0	0	0	0
0	0	0	0								

[Analysis]

- This section records the current attributes and status of the AI device and AI channel.
- During the actual application, only the VQE status message of the enabled function modules is displayed.

[Parameter Description]

Parameter	Description	
AI DEV ATTR (for details, see AIO_ATTR_S.)	AiDev	AI device ID
	WorkMod	AI operating mode i2s_mas: master I ² S mode i2s_sla: slave I ² S mode pcm0_mt: standard master PCM mode pcm0_sl: standard slave PCM mode pcm1_mt: non-standard master PCM mode pcm1_sl: non-standard slave PCM mode
	SampR	Sampling rate Value: {8–96 kHz}
	BitWid	Bit width Value: {8 bits, 16 bits}
	ChnCnt	Number of channels Value {1, 2, 4, 8, 16}
	ClkSel	Clock select
	SoundMod	Sound mode mono: mono channel stereo: stereo channel
	PoiNum	Number of sampling points in a frame
	ExFlag	8-bit extension flag
AI DEV STATUS	FrmNum	Number of frames in a buffer
	AiDev	AI device ID
	IntCnt	Interrupt count IntCnt is incremented by 1 after a frame of audio data is captured. If IntCnt does not increase, the device is not properly connected to the external CODEC.
	fifoCnt	Number of FIFO overflow interrupts



Parameter	Description	
	fifoCnt	fifoCnt is incremented by 1 if overflow occurs when the RX FIFO of the AIU is full.
	buffInt	Number of DMA buffer full interrupts buffInt is incremented by 1 when the DMA buffer for storing data received by the AIU is full.
	FrmTime	Interval between frames (in μ s) The actual audio sampling rate can be calculated based on FrmTime . For example, when FrmTime is about 40,000 μ s and the number of sampling points in a frame is set to 320 , the number of points sampled within one second is 8000. That is, the sampling rate is 8 kHz.
	MaxFrmTime	Longest interval between frames MaxFrmTime indicates the longest interval between frames from the time the system starts till now.
	TranLen	DMA transfer length (in byte)
	IsrTime	DMA interrupt handling time
	MaxIsrTime	Maximum DMA interrupt handling time MaxIsrTime indicates the maximum DMA interrupt handling time from the time the system starts till now.
	CBPhy	Physical address for the DMA buffer
	CBSIZE	DMA buffer size
	ROffSet	Offset of the read pointer of the DMA buffer relative to the start address
	WOffSet	Offset of the write pointer of the DMA buffer relative to the start address
AI DEV EXTEND STATUS	AiDev	AI device ID
	enTrack	Track mode of an AI device
	bMute	Mute flag (invalid for the AI device)
	Volume	Volume (invalid for the AI device)
AI CHN STATUS	AiDev	AI device ID
	AiChn	AI channel ID
	State	Channel status Orig: initialization enable: enabled disable: disabled
	Read	Read pointer of a channel buffer



Parameter	Description	
	Write	Write pointer of a channel buffer
	BufFul	Number of times that the frame buffer is full
	UsrQueLost	Number of lost frames when frames are transmitted in user mode
	UsrFrmDepth	Depth of the audio frame buffer configured by the user
	u32Data0	First 32-bit data segment in a channel buffer
	u32Data1	Second 32-bit data segment in a channel buffer
	UserGet	Number of frames obtained by the user
	UserRls	Number of frames released by the user
AI CHN RESAMPLE STATUS	AiDev	AI device ID
	AiChn	AI channel ID
	State	Channel status orig: initialization enable: enabled disable: disabled
	bResmp	Channel resampling enable Y: enabled N: disabled
	PoiNum	Number of sampling points contained in the input data frames for resampling when this channel is resampled
	InSampR	Sampling rate of the input data frames for resampling when this channel is resampled
	OutSampR	Audio frame sampling rate after resampling when this channel is resampled
AI CHN VQE STATUS 0	AiDev	AI device ID
	AiChn	AI channel ID
	State	Channel status orig: initialization enable: enabled disable: disabled
	bVqe	Channel voice quality enhancement (VQE) enable Y: enabled N: disabled
	workmod	VQE working mode



Parameter	Description
	comm: general mode music: music mode noisy: noise mode
RATE	Sampling rate of the VQE data
PoiNum	Number of sampling points in the VQE frame length
GainVol	VQE volume (in dB) Value range: [-20, +10]
bAnr	Whether to enable the audio noise reduction (ANR) function Y: enabled N: disabled
bAgc	Whether to enable the automatic gain control (AGC) function Y: enabled N: disabled
bEq	Whether to enable the equalizer (EQ) function Y: enabled N: disabled
bHpf	Whether to enable the high-pass filtering (HPF) function Y: enabled N: disabled
bAec	Whether to enable the acoustic echo cancellation (AEC) function Y: enabled N: disabled
bRnr	Whether to enable the recording noise reduction (RNR) function Y: enabled N: disabled
bHdr	Whether to enable the high dynamic range (HDR) function Y: enabled N: disabled
bDrc	Whether to enable the dynamic range control (DRC) function Y: enabled N: disabled



Parameter	Description
	bPeq Whether to enable the parameter equalizer (PEQ) function Y: enabled N: disabled
	WrFile Whether to enable the function of storing channel data in a file for a channel Y: enabled N: disabled
AI CHN VQE EXTEND STATUS 1	AiDev AI device ID
	AiChn AI channel ID
	State Channel status orig: initialization enable: enabled disable: disabled
	bAnr Channel NR enable Y: enabled N: disabled
	bUsrmod Mode select N: automatic mode Y: user mode
	NrIntensity NR strength Value range: [0, 25]
	NoiseDbThr Noise threshold Value range: [30, 60]
	SpProSwi Music detection enable Value range: [0, 1]
	bHdr Enable of the high dynamic range (HDR) function for the AI channel enable: enabled disable: disabled
	bUsrmod Mode select auto: automatic mode manul: user mode
	MaxGain Maximum configured gain allowed by the CODEC Value range: [0, 120]
	MinGain Minimum configured gain allowed by the CODEC Value range: [0, 120]



Parameter	Description	
	MicGainStp	Gain adjustment step Value range: [1, 3]
	MicGain	Current configured gain of the CODEC Value range: [s32MinGaindB, s32MaxGaindB]
AI CHN VQE STATUS 2	AiDev	AI device ID
	AiChn	AI channel ID
	State	Channel status orig: initialization enable: enabled disable: disabled
	bAgc	Channel automatic gain control (AGC) enable Y: enabled N: disabled
	bUsrmod	Mode select N: automatic mode Y: user mode
	NoiseSupSwi	Noise suppression enable Value: 0 or 1
	AdjustSpeed	Adjustment speed Value range: [0 dB/s, 10 dB/s]
	ImproveSNR	Enable for improving the signal-to-noise ratio (SNR) 0: no improvement 1: improved by at most 3 dB 2: improved by at most 6 dB
	MaxGain	Maximum gain Value range: [0 dB, 30 dB]
	NoiseFloor	Noise bottom value Value range: [-65 dB, -20 dB]
	OutputMode	Output mode. The output is muted for signals lower than NoiseFloor . 0: disabled 1: enabled
	TargetLevel	Target level, 1 dB adjustment step Value range: [-40 dB, -1 dB]
	UseHPF	High-pass filtering enable 0: disabled



Parameter		Description
		1: 80 Hz 2: 120 Hz 3: 150 Hz 4: 300 Hz 5: 500 Hz
AI CHN VQE STATUS 3	AiDev	AI device ID
	AiChn	AI channel ID
	State	Channel status orig: initialization enable: enabled disable: disabled
	bEq	Channel equalizer enable Y: enabled N: disabled
	100	Gain range of the frequency band: [-50 dB, +50 dB]
	200	Gain range of the frequency band: [-50 dB, +50 dB]
	250	Gain range of the frequency band: [-50 dB, +50 dB]
	350	Gain range of the frequency band: [-50 dB, +50 dB]
	500	Gain range of the frequency band: [-50 dB, +50 dB]
	800	Gain range of the frequency band: [-50 dB, +50 dB]
AI CHN VQE STATUS 4	1.2k	Gain range of the frequency band: [-50 dB, +50 dB]
	2.5k	Gain range of the frequency band: [-50 dB, +50 dB]
	4k	Gain range of the frequency band: [-50 dB, +50 dB]
	8k	Gain range of the frequency band: [-50 dB, +50 dB]
	AiDev	AI device ID
	AiChn	AI channel ID
	State	Channel status orig: initialization enable: enabled disable: disabled
	bHpf	Channel high-pass filtering enable Y: enabled N: disabled
	bUsrmod	Mode select



Parameter	Description	
	N: automatic mode Y: user mode	
HpfFreq	Cut-off frequency (in Hz) Value: 80, 120, or 150	
bRnr	Chanel Hi-Fi noise suppression enable Y: enabled N: disabled	
bUsrmod	Mode select N: automatic mode Y: user mode	
MaxNrLevel	Maximum NR capability Value range: [2 dB, 20 dB]	
NsThresh	Noise threshold Value range: [-80 dB, -20 dB]	
NrMode	NR mode 0: Reduce the background noise. 1: Reduce the environment noise.	
GainVol	VQE volume (in dB) Value range: [-20, +10]	
AI CHN VQE STATUS5	AiDev	AI device ID
AI CHN VQE STATUS6	AiChn	AI channel ID
AI CHN VQE STATUS7	State	Channel status Orig: initialization enable: enabled disable: disabled
AI CHN VQE STATUS8	bAec	Channel acoustic echo cancellation (AEC) enable Y: enabled N: disabled
	bUsrmod	Mode select N: automatic mode Y: user mode
	CngMode	Comfort noise mode enable Close: disabled Open: enabled
	DTHnlStQTh	Threshold for distinguishing the single talk and double talk



Parameter	Description
	Value range: [0, 32767]
NrAlPsEngy	Remote energy threshold for determining whether passthrough is implemented at the near end 0: -59 dBm0 1: -49 dBm0 2: -39 dBm0
NrClnSupEngy	Energy threshold for forcibly clearing the near-end signal 0: 12 dB 1: 15 dB 2: 18 dB
AecAo	IDs of the AO device and AO channel for which the AEC function is enabled. The value -1 indicates that the AEC function is disabled.
AecFail	Number of times that AEC fails
VcPrtctFrqL	Near-end low-frequency protection region. The value range is [1, 63) at the 8 kHz sampling rate, and is [1, 127) at the 16 kHz sampling rate.
VcPrtctFrqL1	Near-end low-frequency protection region 1. The value range is (VcPrtctFrqL, 63] at the 8 kHz sampling rate, and is (VcPrtctFrqL, 127] at the 16 kHz sampling rate.
EcoBndLow	Low-frequency parameter for voice processing frequency band 1. The value range is [1, 63) at the 8 kHz sampling rate, and is [1, 127) at the 16 kHz sampling rate.
EcoBndHgh	High-frequency parameter for voice processing frequency band 1. The value range is (EcoBndLow, 63] at the 8 kHz sampling rate, and is (EcoBndLow, 127] at the 16 kHz sampling rate.
EcoBndLow2	Low-frequency parameter for voice processing frequency band 2. The value range is [1, 63) at the 8 kHz sampling rate, and is [1, 127) at the 16 kHz sampling rate.
EcoBndHgh2	High-frequency parameter for voice processing frequency band 2. The value range is (EcoBndLow2, 63] at the 8 kHz sampling rate, and is (EcoBndLow2, 127] at the 16 kHz sampling rate.
ERLBND[0,5]	Echo return loss (ERL) protection region. The value range is [1, 63) at the 8 kHz sampling rate, and is [1, 127) at the 16 kHz sampling rate.
ERL[0,6]	ERL protection value for the ERL protection region.



Parameter		Description
		A smaller value indicates greater protection strength. Value range: [0, 18]
AI CHN VQE STATUS9	AiDev	AI device ID
	AiChn	AI channel ID
	State	Channel status Orig: initialization enable: enabled disable: disabled
	bRnr	Chanel Hi-Fi noise suppression enable Y: enabled N: disabled
	bUsrmod	Mode select N: automatic mode Y: user mode
	MaxNrLevel	Maximum NR capability Value range: [2 dB, 20 dB]
	NsThresh	Noise threshold. Value Range: [-80 dB, -20 dB]
AI CHN VQE STATUS10	NrMode	NR mode 0: Reduce the background noise. 1: Reduce the environment noise.
	AiDev	AI device ID
	AiChn	AI channel ID
	State	Channel status Orig: initialization enable: enabled disable: disabled
	bHdr	Enable of the HDR function for the AI channel Y: enabled N: disabled
	bUsrmod	Mode select N: automatic mode Y: user mode
	MaxGain	Maximum configured gain allowed by the CODEC Value range: [0, 120]



Parameter	Description	
	MinGain	Minimum configured gain allowed by the CODEC Value range: [0, 120]
	MicGainStp	Gain adjustment step Value range: [1, 3]
	MicG-20ain	Current configured gain of the CODEC Value range: [s32MinGaindB, s32MaxGaindB]
	pCallBack	Pointer to the callback function
AI CHN VQE STATUS11, AI CHN VQE STATUS12, AI CHN VQE STATUS13	AiDev	AI device ID
	AiChn	AI channel ID
	State	Channel status Orig: initialization enable: enabled disable: disabled
	bDrc	Whether the DRC function is enabled for the channel Y: enabled N: disabled
	bUsrmod	Mode select N: automatic mode Y: user mode
	AtkTime	Time period for the signal to become weaker Value range: [10, 250]
	RlsTime	Time period for the signal to become stronger (ms) Value range: [10, 250]
	OldLev0–OldL ev4	Knee point level Q4 before dynamic curve adjustment Value range: [-1440, 0]
AI CHN VQE STATUS14, AI CHN VQE STATUS15, AI CHN VQE STATUS16, AI CHN VQE STATUS17	NewLev0–Ne wLev4	Knee point level Q4 after dynamic curve adjustment Value range: [-1440, 0]
	AiDev	AI device ID
	AiChn	AI channel ID
	State	Channel status Orig: initialization enable: enabled disable: disabled
	bPeq	Whether to enable the PEQ function Y: enabled



Parameter	Description
	N: disabled
bUsrmod	Mode select N: automatic mode Y: user mode
BandNum	Frequency band number adjustment Value range: (0, 10]
Filter0~Filter9	Type of filter, including high-pass (HP) filter, low-pass (LP) filter, high-pass shelf (HS) filter, low-pass shelf (LS) filter, and peak (PK) filter. <ul style="list-style-type: none">• 0: HP• 1: LS• 2: PK• 3: HS• 4: LP
GaindB0~Gain dB9	Gain of the PEQ frequency band (dB). The gains of the HP filter and LP filter are 0 dB. The value of other types of filter ranges from -15 to +15.
Freq0~Freq9	Center frequency of the PEQ frequency band (Hz). The value range is [20, 4000] for the HP/LS filter, [20, 22000] for the PK filter, and [4000, 22000] for the HS/LP filter.
Q0~Q9	Value Q. The value range is [7, 10] for the HS/LS filter, [5, 100] for the PK filter, and 7 for the HP/LP filter.

NOTE

The actual ID in the AI and AO proc information may be different from that in this document. For example, the PEQ that is displayed in the actual proc information may not be AI CHN VQE STATUS15. This case is related to the actual enabled VQE functions.

12.17 AO

[Debugging Information]

```
# cat /proc/umap/ao
[AI] Version: [Hi3516A_MPP_V1.0.0.0 B010 Debug], Build Time [Aug 25 2014,
11:13:50]

-----AO DEV ATTR-----
AoDev WorkMod SampR BitWid ChnCnt ClkSel SoundMod PoiNum ExFlag
FrmNum
```



0	i2s_mas	8kHz	16bit	1	1	mono	80	1	30
---	---------	------	-------	---	---	------	----	---	----

-----AO DEV STATUS-----

AoDev	IntCnt	fifoCnt	buffInt	FrmTime	MaxFrmTime	TranLen
IsrTime	MaxIsrTime	CBPhy	CBSIZE	ROffSet	WOFFSet	
0	45892	0	0	28012	32048	960
145	163	4aac8000	512	0	180	

-----AO DEV EXTEND STATUS-----

AoDev	enTrack	bMute	Volume
0	0	N	0

-----AO CHN STATUS-----

AoDev	AoChn	State	Read	Write	BufEmp	u32Data0	u32Data1	bResmp
PoiNum	InSampR	OutSampR						
0	0	enable	24	27	7	ffffeffe	ffffeffe	Y
16kHz		8kHz						80

-----AO CHN VQE STATUS0-----

AoDev	AoChn	State	bVqe	workmod	RATE	PoiNum	bAnr	bAgc	bEq
bHpf	WrFile								
0	0	enable	Y	comm	(null)	0	N	N	N

-----AO CHN VQE STATUS1-----

AoDev	AoChn	State	bAnr	bUsrmod	NrIntensity	NoiseDbThr	SpProSwi
0	0	enable	N	N	0	0	0

-----AO CHN VQE STATUS2-----

AoDev	AoChn	State	bAgc	bUsrmod	NoiseSupSwi	AdjustSpeed	ImproveSNR
MaxGain	NoiseFloor	OutputMode	TargetLevel	UseHPF			
0	0	enable	N	N	0	0	0
0	0	0	0				

-----AO CHN VQE STATUS3-----

AoDev	AoChn	State	bEq	100	200	250	350	500	800
1.2k	2.5k	4k	8k						
0	0	enable	N	0	0	0	0	0	0
0	0								

-----AO CHN VQE STATUS4-----

AoDev	AoChn	State	bHpf	bUsrmod	HpfFreq
0	0	enable	N	N	0

[Analysis]



This section records the attributes and status of the current AO device.

[Parameter Description]

Parameter	Description	
AO DEV ATTR (for details, see AIO_ATTR_S.)	AoDev	AO device ID
	WorkMod	AO operating mode i2s_mas: master I ² S mode i2s_sla: slave I ² S mode pcm0_mt: standard master PCM mode pcm0_sl: standard slave PCM mode pcm1_mt: non-standard master PCM mode pcm1_sl: non-standard slave PCM mode
	SampR	Sampling rate Value: {8–96 kHz}
	BitWid	Bit width Value: {8 bits, 16 bits}
	ChnCnt	Number of channels Value {1, 2}
	ClkSel	Clock select
	SoundMod	Sound mode mono: mono channel stereo: stereo channel
	PoiNum	Number of sampling points in a frame
	ExFlag	8-bit extension flag
AO DEV STATUS	FrmNum	Number of frames in a buffer
	AoDev	AO device ID
	IntCnt	Interrupt count IntCnt is incremented by 1 after a frame of audio data is transmitted. If IntCnt does not increase, the device is not properly connected to the external CODEC.
	fifoCnt	Number of FIFO underflow interrupts fifoCnt is incremented by 1 if underflow occurs when the TX FIFO of the AOU is empty.
	buffInt	Number of DMA buffer empty interrupts buffInt is incremented by 1 when the DMA buffer for storing the data transmitted by the AOU is empty.
	FrmTime	Interval between frames (in μ s)



Parameter	Description
	The actual audio sampling rate can be calculated based on FrmTime . For example, when FrmTime is about 40,000 μ s and the number of sampling points in a frame is set to 320 , the number of points sampled within one second is 8000. That is, the sampling rate is 8 kHz.
MaxFrmTime	Longest interval between frames. MaxFrmTime indicates the longest interval between frames from the time the system starts till now.
TranLen	DMA transfer length (in byte)
IsrTime	DMA interrupt handling time
MaxIsrTime	Maximum DMA interrupt handling time MaxIsrTime indicates the maximum DMA interrupt handling time from the time the system starts till now.
CBPhy	Physical address for the DMA buffer
CBSIZE	DMA buffer size
ROffSet	Offset of the read pointer of the DMA buffer relative to the start address
WOffSet	Offset of the write pointer of the DMA buffer relative to the start address
AO DEV EXTEND STATUS	AoDev
	enTrack
	bMute
	Volume
AO CHN STATUS	AoDev
	AoChn
	State
	Read
	Write
	BufEmp
	u32Data0



Parameter	Description
	u32Data1 Second 32-bit data segment in a channel buffer
	bResmp Channel resampling enable enable: Resampling is enabled for this channel. disable: Resampling is disabled for this channel.
	PoiNum Number of sampling points in the data frames input for resampling when channel resampling is enabled
	InSampR Sampling rate in the data frames input for resampling when channel resampling is enabled
	OutSampR Audio frame sampling rate after resampling when this channel is resampled
AO CHN VQE STATUS 0	AoDev AO device ID
	AoChn AO channel ID
	State Channel status orig: initialization enable: enabled disable: disabled
	bVqe Channel VQE enable Y: enabled N: disabled
	workmod VQE working mode comm: general mode music: music mode noisy: noise mode
	RATE Data sampling rate of VQE
	PoiNum Number of sampling points in the VQE data frame
	bAnr ANR enable Y: enabled N: disabled
	bAgc AGC enable Y: enabled N: disabled
	bEq EQ enable Y: enabled N: disabled
	bHpf HPF enable Y: enabled



Parameter		Description
		N: disabled
	WrFile	Whether to enable the function of storing channel data in a file for a channel Y: enabled N: disabled
AO CHN VQE STATUS 1	AoDev	AO device ID
	AoChn	AO channel ID
	State	Channel status orig: initialization enable: enabled disable: disabled
	bAnr	Channel NR enable Y: enabled N: disabled
	bUsrmod	Mode select N: automatic mode Y: user mode
	NrIntensity	NR strength Value range: [0, 25]
	NoiseDbThr	Noise threshold Value range: [30, 60]
	SpProSwi	Music detection enable Value: 0 or 1
AO CHN VQE STATUS2	AoDev	AO device ID
	AoChn	AO channel ID
	State	Channel status orig: initialization enable: enabled disable: disabled
	bAgc	Channel AGC enable Y: enabled N: disabled
	bUsrmod	Mode select N: automatic mode Y: user mode



Parameter	Description	
	NoiseSupSwi	Noise suppression enable Value: 0 or 1
	AdjustSpeed	Adjustment speed Value range: [0 dB/s, 10 dB/s]
	ImproveSNR	Enable for improving the SNR 0: no improvement 1: improved by at most 3 dB 2: improved by at most 6 dB
	MaxGain	Maximum gain Value range: [0 dB, 30 dB]
	NoiseFloor	Noise bottom value Value range: [-65 dB, -20 dB]
	OutputMode	Output mode. The output is muted for signals lower than NoiseFloor . 0: disabled 1: enabled
	TargetLevel	Target level, 1 dB adjustment step Value range: [-40 dB, -1 dB]
	UseHPF	High-pass filtering enable 0: disabled 1: 80 Hz 2: 120 Hz 3: 150 Hz 4: 300 Hz 5: 500 Hz
AO CHN VQE STATUS3	AoDev	AO device ID
	AoChn	AO channel ID
	State	Channel status orig: initialization enable: enabled disable: disabled
	bEq	Channel equalizer enable Y: enabled N: disabled
	100	Gain range of the frequency band: [-50 dB, +50 dB]
	200	Gain range of the frequency band: [-50 dB, +50 dB]



Parameter	Description
250 350 500 800 1.2k 2.5k 4k 8k	Gain range of the frequency band: [-50 dB, +50 dB]
	Gain range of the frequency band: [-50 dB, +50 dB]
	Gain range of the frequency band: [-50 dB, +50 dB]
	Gain range of the frequency band: [-50 dB, +50 dB]
	Gain range of the frequency band: [-50 dB, +50 dB]
	Gain range of the frequency band: [-50 dB, +50 dB]
	Gain range of the frequency band: [-50 dB, +50 dB]
	Gain range of the frequency band: [-50 dB, +50 dB]
AO CHN VQE STATUS4	AoDev
	AoChn
	State
	bHpf
	bUsrmod
	HpfFreq

12.18 AENC

[Debugging Information]

[AENC] Version: [Hi3516A_MPP_V1.0.0.0 B010 Debug], Build Time [Aug 25 2014, 11:13:50]

-----AENC CHN ATTR-----
ChnId PlType ADPCMType PoiNum BufSize G726Rate
0 adpcm DVI4 320 30 NULL

-----AENC CHN STATUS-----



ChnId	RcvFrm	AiQueLost	EncOk	FrmErr	BufFull	GetStrm
RlsStrm	WtFile					
0	307	0	307	0	0	307
307	disable					

[Analysis]

This section records the attributes and status of the current AENC channel.

[Parameter Description]

Parameter	Description
Attribute of AENC Channel	ChnId
	PIType
	ADPCMType
	PoiNum
	BufSize
	G726Rate
Status of AENC Channel	ChnId
	RcvFrm
	AiQueLost
	EncOk
	EncErr
	BufFull
	GetStrm
	RlsStrm
WtFile	Whether to enable the function of storing channel data in a file for a channel
	enable: enabled
	disable: disabled



12.19 ADEC

[Debugging Information]

```
# cat /proc/umap/adec
```

```
[ADEC] Version: [Hi3516A_MPP_V1.0.0.0 B010 Debug], Build Time [Aug 4 2014,  
14:11:13]  
-----ADEC CHN ATTR-----  
ChnId PlType ADPCMType BufSize G726Rate Mode OriSendCnt  
SendCnt GetCnt PutCnt  
0 g726 NULL 50 G726_16K stream 476  
476 473 473
```

[Analysis]

This section records the attributes and status of the current ADEC channel.

[Parameter Description]

Parameter	Description	
Attribute of ADEC Channel	ChnId	ADEC channel ID
	PlType	Type of the decoding protocol
	ADPCMType	Decoding type of ADPCM ADPCM_TYPE_DVI4 ADPCM_TYPE_IMA ADPCM_TYPE_ORG_DVI4
	BufSize	Frame buffer size
	G726Rate	Decoding bit rate during the G726 protocol decoding
	Mode	Decoding by stream or decoding by frame
	OriSendCnt	Number of stream frames that are transmitted from the front end to the decoder for decoding
	SendCnt	Number of audio frames that are successfully transmitted to and decoded by the decoder
	GetCnt	Number of audio frames that are obtained by the user
	PutCnt	Number of audio frames that are released by the user

12.20 H265E



This section is not supported by Hi3519 V100



[Debugging Information]

```
# cat /proc/umap/h265e
[H265E] Version: [Hi3516A_MPP_V1.0.0.0 Debug], Build Time[Sep 6 2011,
09:51:15]
-----MODULE PARAM-----
OnePack      H265eMiniBufMode
0            0
-----CHN ATTR-----
ID  MaxWidth  MaxHeight   Width   Height   profile  C2GEn  BufSize  ByFrame
MaxStrCnt
0    720       576        720     576      mp        Y     829440     Y
200
-----PICTURE INFO-----
ID  EncdStart  EncdSucceed  Lost   Disc   Skip  Pskip  Recode  RlsStr
UnrdStr
0    29         29          0      0      0      0      0      29      0
-----STREAM BUFFER-----
ID  Base       RdTail     RdHead     WrTail     WrHead     DataLen
BufFree
0  0xd8300000  0x6de80     0x6de80     0x6de80     0x6de80     0
829376
-----RefParam INFO-----
ID  EnPred   Base   Enhance  bVirtualIEnable  VirtualIIInterval
VirtualIQpDelta GetVbFail
0    Y       1      0           N                   30      0
0
-----ROI INFO-----
ID  Index  bRoiEn  bAbsQp   Qp   Width   Height  StartX  StartY  BgSrcFr
BgTarFr
0    0       N       N      -10     256     256      0       0      30
15
-----Syntax INFO1-----
ID  SlcspltEn  Slcmode     Slcsize  bCrossSlc
0    Y       ByteNum    10000      N
-----Syntax INFO2-----
ID  DblkEn   Tc     Beta   Saoluma  Saochroma  EntrpyFlag
0    1       0       0       1       1       1
-----Syntax INFO3-----
ID  TimInfFlag  NumInTick  TimeScal  DiffOne
0    N       1       60      1
-----Pu INFO-----
ID  Pu32En  Pu16En  Pu8En  Pu4En  ConsIntra  IntraSmo  IpcmEn  PcmLoFil
```



```
NumMergCan
0   1     1   1   1   0     1     1   1   2
-----Trans INFO-----
ID  TranByPass   TranSkip  CbQpOffset  CrQpOffset
0      0          0          0            0
```

[Analysis]

During multi-channel H.265 encoding, the upper-layer software may not fetch streams in a timely manner. As a result, the stream buffer is insufficient, which causes frame loss. The best way of checking whether frames are lost is to view the **Disc** and **UnrdStr** parameters. **Disc** indicates the number of discarded frames due to stream buffer insufficiency. **UnrdStr** indicates the number of remaining frames in the stream buffer. A larger value indicates streams are fetched in a less timely manner.

[Parameter Description]

Parameter		Description
MODULE PARAM	OnePack	Mode in which streams are obtained 0: Streams are obtained in multiple-packet mode. 1: Streams are obtained in single-packet mode.
	H265eMiniBufMode	Configuration mode of the H.265 encoding stream buffer 0: normal mode 1: memory reduction mode
CHN ATTR	ID	Channel ID
	MaxWidth	Maximum width of the encoding channel (in pixel)
	MaxHeight	Maximum height of the encoding channel (in pixel)
	Width	Width (in pixel)
	Height	Height (in pixel)
	profile	Encoding channel profile Mp: main profile
	C2GEn	Color-to-gray enable
	BufSize	Stream buffer size (in byte)
	ByFrame	Whether to obtain streams by frame
PICTURE INFO	MaxStrCnt	Maximum number of frames in the stream buffer Default value: 200
	ID	Channel ID
	EncdStart	Number of received pictures This value is incremented by 1 after the pictures



Parameter	Description
	to be encoded are received.
EncdSucceed	Number of frames that are successfully encoded
Lost	Number of frames that are discarded during encoding. The discarded frames include: <ul style="list-style-type: none">• Frames discarded due to encoder exceptions or jumbo frames. The number of these frames is the value of Disc.• Frames that are discarded for controlling the frame rate. The frame rate is controlled by the RC. The number of these frames is the value of Skip.• Frames that are discarded when the configured transient bit rate exceeds the threshold
Disc	Number of frames discarded for the following reasons: <ul style="list-style-type: none">• Jumbo frames occur.• An exception occurs in the encoder. For example, the encoder crashes.• The encoder has detected stream buffer insufficiency during encoding.
Skip	Number of frames that are discarded for controlling the frame rate or discarded when the transient bit rate is above the threshold during encoding.
Pskip	Number of frames that are encoded as Pskip frames
Recode	Number of frames that are re-encoded by the encoder
RlsStr	Number of frames that are obtained and released by users
UnrdStr	Number of frames that are not obtained by users
STREAM BUFFER	ID
	base
	RdTail
	RdHead
	WrTail
	WrHead
	datalen
	buffree



Parameter	Description	
RefParam INFO	ID	Channel ID
	EnPred	Whether some frames at the base layer are referenced by other frames at the base layer
	Base	Base layer period
	Enhance	Enhance layer period
	bVirtualIEnable	Whether to insert virtual I frames
	u32VirtualIInterval	Interval of virtual I frames
	s32VirtualIQpDelta	QpDelta of the virtual I frame relative to the common P frame
	GetVbFail	Number of times that the VB of reconstruction frames fails to be obtained
ROI INFO	ID	Channel ID
	Index	ROI index
	bRoiEn	ROI enable
	bAbsQp	Whether the ROI uses the QP mode
	Qp	QP value configured by the ROI
	Width	ROI width (in pixel)
	Height	ROI height (in pixel)
	StartX	Start horizontal coordinate of the ROI (in pixel)
	StartY	Start vertical coordinate of the ROI (in pixel)
	BgSrcFr	Source frame rate of a non-ROI
Syntax INFO 1	BgTarFr	Target frame rate of a non-ROI
	ID	Channel ID
	SlcspltEn	Slice split enable
	Slcmode	Slice split mode <ul style="list-style-type: none">• ByteNum: split by byte• MbLine: split by macroblock
	Slcsize	Size of each slice When the slice is split by byte, it indicates the number of bytes of each slice. When the slice is split by macroblock, it indicates the number of macroblock lines of each slice.
	bCrossSlc	Whether filtering is performed on the left and top borders of slices



Parameter	Description	
Syntax INFO 2	ID	Channel ID
	DblkEn	1 – Value of the syntax element <u>slice_deblocking_filter_disabled_flag</u>
	Tc	Value of the syntax element <u>slice_tc_offset_div2</u>
	Beta	Value of the syntax element <u>slice_beta_offset_div2</u>
	Saoluma	Value of the syntax element <u>slice_sao_luma_flag</u>
	Saochroma	Value of the syntax element <u>slice_sao_chroma_flag</u>
	EntrpyFlag	Value of the syntax element <u>cabac_init_flag</u>
Syntax INFO3	ID	Channel ID
	TimInfFlag	Value of the syntax element <u>timing_info_present_flag</u>
	NumInTick	Value of the syntax element <u>num_units_in_tick</u>
	TimeScal	Value of the syntax element <u>time_scale</u>
	DiffOne	Value of the syntax element <u>num_units_in_tick</u>
Pu INFO	ID	Channel ID
	Pu32En	PU32x32 enable
	Pu16En	PU16x16 enable
	Pu8En	PU8x8 enable
	Pu4En	PU4x4 enable
	ConsIntra	Value of the syntax element <u>constrained_intra_pred_flag</u>
	IntraSmo	Value of the syntax element <u>strong_intra_smoothing_enabled_flag</u>
	IpcmEn	Value of the syntax element <u>pcm_enabled_flag</u>
	PcmLoFil	Value of the syntax element <u>pcm_loop_filter_disabled_flag</u>
	NumMergCan	Maximum number of Merg candidate points
Trans INFO	ID	Channel ID
	TranByPass	Value of the syntax element <u>transquant_bypass_enabled_flag</u>
	TranSkip	Value of the syntax element <u>transform_skip_enabled_flag</u>



Parameter	Description
	CbQpOffset
	CrQpOffset

12.21 ACODEC

[Debugging Information]

[ACODEC] Version: [Hi3516A_MPP_V1.0.0.0 B00 Debug], Build Time [Dec 14 2014, 10:30:46]

-----ACODEC PARAM-----

LGain(dB)	RGain(dB)	DacLVol(dB)	DacRVol(dB)	AdcLVol(dB)	AdcRVol(dB)
MicLMut	MicRMut	DacLMut	DacRMut	BoostL	BoostR
0.0	0.0	-1	-1	0	0
0	0	0	0	0	0

[Analysis]

This section records the input and output volume status of the current audio CODEC.

[Parameter Description]

Parameter	Description
Status of ACODEC Volume	LGain
	Input analog gain of the audio-left channel
	RGain
	Input analog gain of the audio-right channel
	DacLVol
	Output digital gain of the audio-left channel
	DacRVol
	Output digital gain of the audio-right channel
	AdcLVol
	Input digital gain of the audio-left channel
	AdcRVol
	Input digital gain of the audio-right channel
	MicLMut
	Input digital mute enable for the audio-left channel
	0: disabled
	1: enabled
	MicRMut
	Input digital mute enable for the audio-right channel
	0: disabled
	1: enabled
	DacLMut
	Output digital mute enable for the audio-left channel
	0: disabled
	1: enabled



	DacRMut	Output digital mute enable for the audio-right channel 0: disabled 1: enabled
	BoostL	Input gain boost control for the audio-left channel 0: 0 dB 1: 26 dB (Different chips differ in the value. For Hi3519 V100, the value is 20 dB.)
	BoostR	Input gain boost control for the audio-right channel 0: 0 dB 1: 26 dB (Different chips differ in the value. For Hi3519 V100, the value is 20 dB.)



Contents

12 Proc2 Debugging Information.....	12-1
12.1 Overview.....	12-1
12.2 H264E	12-2
12.3 JPEGE	12-7
12.4 RC	12-9
12.5 VENC.....	12-21
12.6 H265E	12-27
12.7 Fisheye	12-32



12 Proc2 Debugging Information

12.1 Overview

The debugging information is obtained from the proc file system of Linux. The information reflects the status of the current system and can be used to locate and analyze problems.



CAUTION

The proc2 debugging information is supported only by Hi3519 V100.

[Directory]

/proc/umap

[Checklist]

File	Description
h265e	Records the encoding attributes, status, and history statistics of each channel during H.265 encoding.
h264e	Records the encoding attributes, status, and history statistics of each channel during H.264 encoding.
jpege	Records the encoding attributes, status, and history statistics of each channel during JPEG encoding.
rc	Records the stream control attributes, status, and history statistics of each encoding channel.
venc	Records the information about the video encoder.
fisheye	Records the information about the fisheye correction module.

[View Methods]



- You can run the **cat** command such as **cat /proc/umap/venc** on the console. You can also run file operation commands such as **cp /proc/umap/ ./ -rf** to copy all the proc files under umap to the current directory.
- You can read the preceding files as common read-only files through applications such as fopen and fread.

NOTE

Note the following when reading the parameter descriptions:

- For the parameter whose value is **0** or **1**, if the mapping between the values and the definitions is not specified, the value 1 indicates affirmative and the value 0 indicates negative.
- For the parameter whose value is aaa, bbb, or ccc, if the mapping between the values and the definitions is not specified, you can identify the parameter definitions based on **aaa**, **bbb**, or **ccc**.

12.2 H264E

[Debugging Information]

```
# cat /proc/umap/h264e
[H264E] Version: [Hi3519V100_MPP_V1.0.0.0 B010 Debug], Build Time [Sep 6 2015,
09:51:15]
-----MODULE PARAM-----
OnePack      H264eVBSource
0             1
-----CHN ATTR-----
ID  MaxWidth  MaxHeight Width   Height   profile  C2GEn   BufSize  ByFrame
GopMode  MaxStrCnt
0       720      576        720      576      hp       Y        829440    Y
NormalP     200
-----PICTURE INFO-----
ID  EncdStart  EncdSucceed  Lost    Disc    Pskip   Recode  RlsStr  UnrdStr
0       29         29          0       0       0       0       29       0
-----STREAM BUFFER-----
ID  Base       RdTail     RdHead     WrTail     WrHead     DataLen
BufFree
0  0xd8300000  0x6de80     0x6de80     0x6de80     0x6de80     0
829376
-----RefParam INFO-----
ID  EnPred     Base       Enhance
0       Y         1           0
-----ROI INFO-----
ID  Index      bRoiEn   bAbsQp    Qp      Width   Height  StartX  StartY  BgSrcFr
BgTarFr
0       0         1         0        -10      256      256      0         0        30
15
-----Syntax INFO1-----
ID  SlcspltEn  Slcmode   Slcsize   IntraRefresh enIslice
```



```
RefreshLine QpOfIDR
0 Y ByteNum 10000 N N
17 51
-----Inter & Intra prediction INFO-----
ID profile HWsize VWsize P16x16 P16x8 P8x16 P8x8 MvExt I16x16
Inxn Ipcm
0 hp 3 2 Y Y Y Y Y Y Y Y Y
-----Syntax INFO2-----
ID profile EntrpyI EntrpyP EntrpyB Itrans Ptrans QMatrix POC
DblkIdc Alpha Beta
0 hp cavlc cabac cabac all all N 2 0
5 5
```

[Analysis]

During multi-channel H.264 encoding, the upper-layer software may not fetch streams in a timely manner. As a result, the stream buffer is insufficient, which causes frame loss. The best way of checking whether frames are lost is to view the **Disc** and **UnrdStr** parameters. **Disc** indicates the number of discarded frames due to stream buffer insufficiency. **UnrdStr** indicates the number of remaining frames in the stream buffer. A larger value indicates streams are fetched in a less timely manner.

[Parameter Description]

Parameter		Description
MODULE PARAM	OnePack	Mode in which streams are obtained 0: Streams are obtained in multiple-packet mode. 1: Streams are obtained in single-packet mode.
	H264eVBSource	Mode in which the VB is obtained for the reference frame and reconstruction frame 1: The private VBs are used. 2: The VBs are allocated by the user.
CHN ATTR	ID	Channel ID
	MaxWidth	Maximum width of the encoding channel (in pixel)
	MaxHeight	Maximum height of the encoding channel (in pixel)
	Width	Width (in pixel)
	Height	Height (in pixel)
	profile	Encoding channel profile Base: baseline Mp: main profile Hp: high profile
	C2GEn	Color-to-gray enable Value: {0, 1}



Parameter	Description	
	BufSize	Stream buffer size (in byte)
	ByFrame	Whether to obtain streams by frame Value: {0, 1}
	GopMode	GOP mode
	MaxStrCnt	Maximum number of frames in the stream buffer Default value: 200
PICTURE INFO	ID	Channel ID
	EncdStart	Number of received pictures This value is incremented by 1 after the pictures to be encoded are received.
	EncdSucceed	Number of frames that are encoded successfully
	Lost	Number of frames that are discarded during encoding. The discarded frames include: <ul style="list-style-type: none">• Frames that are discarded due to encoder exceptions or jumbo frames. That is, the number of frames is the value of Disc.• Frames that are discarded due to insufficient VBs when the user VBs are used
	Disc	Number of frames discarded for the following reasons: <ul style="list-style-type: none">• Jumbo frames occur.• An exception occurs in the encoder. For example, the encoder crashes.• The frame rate is controlled or the instantaneous bit rate is above the threshold.
	Pskip	Number of frames that are encoded as Pskip frames
	Recode	Number of frames that are re-encoded by the encoder
	RlsStr	Number of frames that are obtained and released by users
	UnrdStr	Number of frames that are not obtained by users
	STREAM BUFFER	ID base RdTail RdHead WrTail WrHead datalen



Parameter		Description
	buffree	Free buffer size
RefParam INFO	ID	Channel ID
	EnPred	Whether some frames at the base layer are referenced by other frames at the base layer
	Base	Base layer period
	Enhance	Enhance layer period
ROI INFO	ID	Channel ID
	Index	Region of interest (ROI) index
	bRoiEn	ROI enable
	bAbsQp	Whether the ROI uses the absolute quantizer parameter (QP) mode
	Qp	QP value configured by the ROI
	Width	ROI width (in pixel)
	Height	ROI height (in pixel)
	StartX	Start horizontal coordinate of the ROI (in pixel)
	StartY	Start vertical coordinate of the ROI (in pixel)
	BgSrcFr	Source frame rate of a non-ROI
	BgTarFr	Target frame rate of a non-ROI
Syntax INFO	ID	Channel ID
	SlcsplEn	Slice split enable
	Slcmode	Slice split mode <ul style="list-style-type: none">• ByteNum: split by byte• MbLine: split by macroblock
	Slcsize	Size of each slice When the slice is split by byte, it indicates the number of bytes of each slice. When the slice is split by macroblock, it indicates the number of macroblock lines of each slice.
	IntraRefresh	Enable for refreshing the I macroblock in the P frame
	enIslice	Enable for converting the first refreshed I macroblock into Islice 0: disabled 1: enabled
	RefreshLine	Number of refreshed rows in the I macroblock for each frame



Parameter		Description
	QpOffIDR	QP value of the IDR frame when IDR frame is requested
Inter & Intra prediction INFO	ID	Channel ID
	profile	Encoding channel profile type <ul style="list-style-type: none">• Base: baseline• MP: main profile• HP: high profile
	HWsize	Size of the horizontal search window
	VWsize	Size of the vertical search window
	P16x16	Inter16x16 prediction mode enable
	P16x8	Inter16x8 prediction mode enable
	P8x16	Inter8x16 prediction mode enable
	P8x8	Inter8x8 prediction mode enable
	MvExt	Whether to extend the boundaries when search reaches the boundaries of the pictures
	I16x16	Intra16x16 prediction mode enable
Syntax INFO	Inxn	Intra8x8 and intra4x4 prediction mode enable
	Ipcm	Ipcm prediction mode enable
	ID	Channel ID
	Profile	Encoding channel profile type <ul style="list-style-type: none">• Base: baseline• MP: main profile• HP: high profile
	EntrpyI	Entropy encoding mode of the I frame, context-based adaptive binary arithmetic coding (CABAC) or context-based adaptive variable-length coding (CAVLC)
	EntrpyP	Entropy encoding mode of the P frame, CABAC or CAVLC
	EntrpyB	Entropy encoding mode of the B frame, CABAC or CAVLC
	Itrans	Transform mode used by I frame <ul style="list-style-type: none">• All: The trans8x8 mode and trans4x4 mode are enabled simultaneously.• 4x4: The trans4x4 mode is enabled.• 8x8: The trans8x8 mode is enabled.



Parameter	Description
Ptrans	Transform mode used by P frame <ul style="list-style-type: none">• All: The trans8x8 mode and trans4x4 mode are enabled simultaneously.• 4x4: The trans4x4 mode is enabled.• 8x8: The trans8x8 mode is enabled.
QMatrix	Whether a customized quantization table is used
POC	Value of the syntax element pic_order_cnt_type
DblkIdc	Value of the syntax element disable_deblocking_filter_idc
Alpha	Value of the syntax element slice_alpha_c0_offset_div2
Beta	Value of the syntax element slice_beta_offset_div2

12.3 JPEGE

[JPEGE] Version: [Hi3519V100_MPP_V1.0.0.0 B010 Debug], Build Time [Sep 6 2015, 09:51:15]

-----MODULE PARAM-----

OnePack

0

-----ATTRIBUTE1-----

ID	bMjpeg	PicType	MaxWidth	MaxHeight	Width	Height	BufSize
ByFrm	MCU	Qfactor	C2GEn	DcfEn			
12	Y	YUV422	32	32	32	32	2048
0	90	Y	N				Y

-----STATUS1-----

ID	Buflen	FreeLen	StrmCnt	MaxStrm
12	829440	829376	0	100000

-----STATUS2-----

ID	PicRec	PicCoded	PicDropped	PicDisc	NoStmCnt	RcFail	PicRecode
UnrdStr							
12	189	8	181	0	0	0	0
0							

-----STREAM BUFFER-----

ID	Base	RdTail	RdHead	WrTail	WrHead	Buflen	DataLen	Buffree
12	0xc3400000	12800	12800	12800	12800	829440	0	829376

[Analysis]



This section records the encoding attributes, status, and history statistics of each channel during JPEG encoding. A maximum of 64 encoding channels are supported. The information is used to locate problems such as system blocking and frame loss.

[Parameter Description]

Parameter		Description
MODULE PARAM	OnePack	Mode in which streams are obtained 0: Streams are obtained in multiple-packet mode. 1: Streams are obtained in single-packet mode.
ATTRIBUTE1	ID	Channel ID
	bMjpeg	MJPEG encoding N: JPEG snapshot Y: MJPEG encoding
	PicType	Picture type: YUV422 or YUV420
	MaxWidth	Maximum width of the encoding channel (in pixel)
	MaxHeight	Maximum height of the encoding channel (in pixel)
	Width	Picture width
	Height	Picture height
	BufSize	Stream buffer size (in byte)
	MCU	Number of minimum coded units (MCUs) in each embedded CPU subsystem (ECS)
	ByFrm	Whether to obtain streams by frame
	Qfactor	Channel Qfactor
	C2GEn	Color-to-gray enable
	DcfEn	Whether the JPEG picture has a thumbnail
Status1	ID	Channel ID
	BufLen	Total length of the stream buffer
	FreeLen	Length of the free stream buffer
	StrmCnt	Number of frames in the current stream buffer
	MaxStrm	Maximum number of frames in the stream buffer Default value: 100,000
Status2	ID	Channel ID
	PicRec	Number of pictures that are transmitted for encoding



Parameter	Description
	PicCoded Number of frames that are successfully encoded
	PicDropped Total number of discarded pictures before encoding, including the discarded pictures of Nostment , and ReFail
	PicDisc Number of frames discarded for the following reasons: <ul style="list-style-type: none">• Jumbo frames occur.• The encoder is abnormal, for example, the system is crashed.• The encoder detects stream buffer insufficiency during encoding.
	Nostmcnt Number of times that frames are discarded because the number of frames in the stream buffer exceeds upper limit
	RcFail Number of times that frames are discarded when the frame rate is controlled or the instantaneous bit rate is above the threshold
	PicRecode Number of jumbo frames that are encoded again
	UnrdStr Number of frames that are not obtained by users
STREAM BUFFER	ID Channel ID
	base Base address for a stream buffer
	RdTail Read tail pointer
	RdHead Read head pointer
	WrTail Write tail pointer
	WrHead Write head pointer
	Buflen Stream buffer length
	datalen Data length
	buffree Free buffer size

12.4 RC

[Debugging Information]

The following is the proc information in CBR mode:

[RC] Version: [Hi3519V100_MPP_V1.0.0.0 B010 Debug], Build Time [Aug 10 2016, 12:00:46]



```
----- BASE PARAMS 1-----
ChnId      Gop      StatTm   ViFr     TrgFr  ProType   RcMode   Br(kbps)  FluLev IQp
PQp       BQp
          0        25       4        25      25/0      96        CBR       2048        1        N/A
N/A      N/A

----- BASE PARAMS 2-----
ChnId      EnableIdr QpMapMode bQpMapAbsQp
          0           Y           N/A           N/A

-----RUN COMM PARAM 1-----
ChnId  RowQpDelta                                ThrdI(16)
          0        2  [0  0  0  0  3  3  5  5  8  8  8  15  15  20  25  25]

-----RUN COMM PARAM 2-----
ChnId  FirstFrmStartQP                            ThrdP(16)
          0        -1  [0  0  0  0  3  3  5  5  8  8  8  15  15  20  25  25]

-----RUN COMM PARAM 3-----
ChnId  PARAM                                ThrdB(16)
          0  DEFAULT  [0  0  0  0  3  3  5  5  8  8  8  15  15  20  25  25]

-----RUN COMM PARAM 4-----
ChnId                                LevelI(16)
          0  [1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1]

-----RUN COMM PARAM 5-----
ChnId                                LevelP(16)
          0  [1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1]

-----RUN COMM PARAM 6-----
ChnId                                LevelB(16)
          0  [1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1]

-----RUN COMM PARAM 7-----
ChnId      bLost    LostThr   LostFrmStr   EncGap   RCPriority   SprFrmMod
SprIFrm    SprPFrm   SprBFrm
          0        N        83886080      NORMAL      0        BITRATE      None
5000000    5000000    5000000

-----GOP MODE ATTR-----
ChnId      GopMode   IpQpDelta  SPInterval  SPQpDelta  BFrmNum   BQpDelta
BgInterval ViQpDelta
          0  NormalP        3        N/A        N/A        N/A        N/A
```



N/A N/A

-----RUN CBR PARAM-----
ChnId IPDltQp MinIprop MaxIprop MaxQp MinQp MaxIQp MinIQp QLevel
MaxReEncTimes
0 3 1 20 51 10 51 10 3
2

-----RUN INFO1-----
ChnId InsBr(kbps) InsFr WatL CfgBt(kb) RealBt(kb) IPRatio
TarPercent StartQp MinQp MaxQp
0 1962 10 894 46 37 20
N/A 26 10 51

-----RC PERFORMANCE INFO-----
ChnId StaOfstaTim TotaOfstaTim StaOfEndTim TotaOfEndTim TotalTime
0 4204255658 473 4204258429 113 586

The following is the proc information in VBR mode:

[RC] Version: [Hi3519V100_MPP_V1.0.0.0 B010 Debug], Build Time [Aug 10 2016, 12:00:46]

----- BASE PARAMS1 -----
ChnId Gop StatTm ViFr TrgFr ProType RcMode Br(kbps) FluLev IQp
PQp BQp
1 30 1 30 30/0 96 VBR 2048 N/A N/A
N/A N/A

----- BASE PARAMS2 -----
ChnId EnableIdr QpMapMode bQpMapAbsQp
1 Y N/A N/A

-----RUN COMM PARAM 1-----
ChnId RowQpDelta ThrdI(16)
1 2 [0 0 0 0 3 3 5 5 8 8 8 15 15 20 25 25]

-----RUN COMM PARAM 2-----
ChnId FirstFrmStartQP ThrdP(16)
1 -1 [0 0 0 0 3 3 5 5 8 8 8 15 15 20 25 25]

-----RUN COMM PARAM 3-----
ChnId PARAM ThrdB(16)
1 DEFAULT [0 0 0 0 3 3 5 5 8 8 8 15 15 20 25 25]

-----RUN COMM PARAM 4-----



ChnId	LevelI(16)
1	[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]

-----RUN COMM PARAM 5

ChnId	LevelP(16)
1	[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]

-----RUN COMM PARAM 6

ChnId	LevelB(16)
1	[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]

-----RUN COMM PARAM 7

ChnId	bLost	LostThr	LostFrmStr	EncGap	RCPriority	SprFrmMod
SprIFrm	SprPfrm	SprBFrm				
1	N	83886080	NORMAL	0	BITRATE	None
500000	500000	500000				

-----GOP MODE ATTR

ChnId	GopMode	IpQpDelta	SPIInterval	SPQpDelta	BFrmNum	BQpDelta
BgInterval	ViQpDelta					
1	NormalP	3	N/A	N/A	N/A	N/A
N/A	N/A					

-----RUN VBR PARAM

-----RUN INFO1

ChnId	InsBr (kbps)	InsFr	WatL	CfgBt (kb)	RealBt (kb)	IPRatio	TarPercent
StartQp	MinQp	MaxQp					
1	1093	10	177	43	14	13	NA
	29	16	40				

-----RC PERFORMANCE INFO-----

ChnId	StaOfstaTim	TotaOfstaTim	StaOfEndTim	TotaOfEndTim	TotalTime
1	4204255758	234	4204261104	78	312

The following is the proc information in AVBR mode:

[RC] Version: [Hi3519V100_MPP_V1.0.0.0 B010 Debug], Build Time[Aug 10 2016, 12:00:46]

----- BASE PARAMS1



ChnId	Gop	StatTm	ViFr	TrgFr	ProType	RcMode	Br(kbps)	FluLev	IQp
PQp	BQp								
2	30	1	30	30/0	96	AVBR	2048	N/A	N/A
N/A	N/A								
----- BASE PARAMS2 -----									
ChnId	EnableIdr	QpMapMode	bQpMapAbsQp						
1	Y	N/A	N/A						
-----RUN COMM PARAM 1-----									
ChnId	RowQpDelta				ThrdI(16)				
2	2	[0 0 0 0 3 3 5 5 8 8 8 15 15 20 25 25]							
-----RUN COMM PARAM 2-----									
ChnId	FirstFrmStartQP				ThrdP(16)				
2	-1	[0 0 0 0 3 3 5 5 8 8 8 15 15 20 25 25]							
-----RUN COMM PARAM 3-----									
ChnId	PARAM				ThrdB(16)				
2	DEFAULT	[0 0 0 0 0 3 3 5 5 8 8 8 15 15 20 25							
25]									
-----RUN COMM PARAM 4-----									
ChnId				LevelI(16)					
2	[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]								
-----RUN COMM PARAM 5-----									
ChnId				LevelP(16)					
2	[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]								
-----RUN COMM PARAM 6-----									
ChnId				LevelB(16)					
2	[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]								
-----RUN COMM PARAM 7-----									
ChnId	bLost	LostThr	LostFrmStr	EncGap	RCPriority	SprFrmMod			
SprIFrm	SprPFrm	SprBFrm							
2	N	83886080	NORMAL	0	BITRATE	None			
500000	500000	500000							
-----GOP MODE ATTR-----									
ChnId	GopMode	IpQpDelta	SPInterval	SPQpDelta	BfrmNum	BQpDelta			
BgInterval	ViQpDelta								
2	NormalP	3	N/A	N/A	N/A	N/A			
N/A	N/A								



```
-----RUN VBR PARAM-----
ChnId IPDltQp ChgPs MinIprop MaxIprop MaxQp MinQp MaxIQp MinIQp
MaxReEncTimes MaxStillQP MinPercent MinStillPSNR
2 3 90 1 100 51 16 51 16
2 35 25 0

-----RUN INFO1-----
ChnId InsBr(kbps) InsFr WatL CfgBt(kb) RealBt(kb) IPRatio
TarPercent StartQp MinQp MaxQp
2 576 30 383 7 2 123 52
28 16 51

-----RC PERFORMANCE INFO-----
ChnId StaOfstaTim TotaOfstaTim StaOfEndTim TotaOfEndTim TotalTime
2 11487278472 3800 11487283728 540 4340
```

The following is proc information in FixQp mode:

[RC] Version: [Hi3519V100_MPP_V1.0.0.0 B010 Debug], Build Time [Aug 10 2016, 15:07:01]

```
----- BASE PARAMS1 -----
ChnId Gop StatTm ViFr TrgFr ProType RcMode Br(kbps) FluLev IQp
PQp BQp
0 30 1 30 30/0 96 FIXQP N/A N/A 20
23 23

----- BASE PARAMS2 -----
ChnId EnableIdr QpMapMode bQpMapAbsQp
1 Y N/A N/A

-----RUN COMM PARAM 1-----
ChnId RowQpDelta ThrdI(16)
0 0 [ 0 0 0 0 0 0 0 0 0 255 255 255 255 255 255 255 255 255 ] 255

-----RUN COMM PARAM 2-----
ChnId FirstFrmStartQP ThrdP(16)
0 -1 [ 0 0 0 0 0 0 0 0 0 255 255 255 255 255 255 255 255 255 ] 255

-----RUN COMM PARAM 3-----
ChnId PARAM ThrdB(16)
0 DEFAULT [ 0 0 0 0 0 0 0 0 255 255 255 255 255 255 255 255 255 ] 255

-----RUN COMM PARAM 4-----
ChnId LevelI(16)
```



```
0 [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

-----RUN COMM PARAM 5-----

ChnId	LevelP(16)
0	[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]

-----RUN COMM PARAM 6-----

ChnId	LevelB(16)
0	[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]

-----RUN COMM PARAM 7-----

ChnId	bLost	LostThr	LostFrmStr	EncGap	RCPriority	SprFrmMod
SprIFrm	SprPFrm	SprBFrm				
0	N	83886080	NORMAL	0	BITRATE	None
500000	500000	500000				

-----GOP MODE ATTR-----

ChnId	GopMode	IpQpDelta	SPInterval	SPQpDelta	BfrmNum	BQpDelta
BgInterval	ViQpDelta					
0	NormalP	3	N/A	N/A	N/A	N/A
N/A	N/A					

-----RUN INFO1-----

ChnId	InsBr(kbps)	InsFr	WatL	CfgBt(kb)	RealBt(kb)	IPRatio
TarPercent	StartQp	MinQp	MaxQp			
0	1008	12	0	0	0	0
N/A	23	23	23			

-----RC PERFORMANCE INFO-----

ChnId	StaOfstaTim	TotaOfstaTim	StaOfEndTim	TotaOfEndTim	TotalTime
0	15778255608	409	15778258368	61	470

The following is the proc information in QPMAP mode:

[RC] Version: [Hi3519V100_MPP_V1.0.0.0 B010 Debug], Build Time [Nov 22 2015, 10:24:02]

-----BASE PARAMS 1-----

ChnId	Gop	StatTm	ViFr	TrgFr	ProType	RcMode	Br(kbps)	FluLev	IQp
PQp	BQp								
0	30	1	30	30/0	96	QPMAP	N/A	N/A	N/A
N/A	N/A								

----- BASE PARAMS2 -----

ChnId	EnableIdr	QpMapMode	bQpMapAbsQp
1	Y	MEANQP	Y





```
-----RUN VBR PARAM-----
ChnId IPDltQp ChgPs MinIprop MaxIprop MaxQp MinQp MaxIQp MinIQp
MaxReEncTimes MaxStillQP MinPercent MinStillPSNR

-----RUN INFO1-----
ChnId InsBr(kbps) InsFr WatL CfgBt (kb) RealBt (kb) IPRatio
TarPercent StartQp
0 2189 30 0 0 66 356 N/A 0
MinQp MaxQp
0 51

-----RC PERFORMANCE INFO-----
ChnId StaOfstaTim TotaOfstaTim StaOfEndTim TotaOfEndTim
TotalTime
0 147020223491 24694442 147020229141 1062991 35324357
```

[Analysis]

This section records the bit rate control information during encoding.

[Parameter Description]

Parameter	Description	
BASE PARAMS 1	ChnId	VENC channel ID
	Gop	Encoding group of pictures (GOP)
	StatTm	Bit rate statistics (in second)
	ViFr	Frame rate for transmitting pictures by the VIU
	TrgFr	Target frame rate for encoding
	ProType	Encoding type
	RcMode	Bit rate control mode (CBR, VBR, AVBR, or FixQp)
	Br(kbps)	In CBR mode, it indicates that average bit rate. In VBR mode, it indicates that the maximum bit rate. The unit of bit rate kbit/s.
	FluLev	Fluctuation level, valid only for the CBR mode
	IQp	I frame QP, valid only in FixQp mode
BASE PARAMS 2	PQp	P frame QP, valid only in FixQp mode
	BQp	B frame QP, valid only in FixQp mode
	ChnId	VENC channel ID
EnableIdr	EnableIdr	IDR enable switch
	QpMapMode	Method of assigning the QP value for CU2 or CU64 when the current channel uses the QPMAP bit rate control mode



Parameter		Description
	bQpMapAbsQp	Whether CU32 or CU64 uses the absolute QP when the current channel uses the QPMAP bit rate control mode
RUN COMM PARAM 1	ChnId	VENC channel ID
	RowQpDelta	Fluctuation amplitude of the start QP value of each macroblock row relative to the start QP value of the frame during macroblock-level bit rate control Value range: [0, 10]
	FirstFrameStart Qp	Start QP value of the first frame If FirstFrameStartQp is -1, the start QP value of the first frame is internally calculated by the encoder. If FirstFrameStartQp is set to any other value, the user specified the start QP value of the first frame.
	ThrdI(16)	Mad threshold for controlling the macroblock-level bit rate of I frames Value range: [0, 255]
RUN COMM PARAM 2	ChnId	VENC channel ID
	FirstFrmStartQP	Start QP value of the first frame If the parameter value is -1, the start QP value of the first frame is internally calculated by the encoder. If this parameter is set to any other valid value, the user specified the start QP value of the first frame.
	ThrdP(16)	Mad threshold for controlling the macroblock-level bit rate of P frames Value range: [0, 255]
RUN COMM PARAM 3	ChnId	VENC channel ID
	PARAM	No meaning. The default value is used.
	ThrdB(16)	Mad threshold for controlling the macroblock-level bit rate of B frames Value range: [0, 255]
RUN COMM PARAM 4	ChnId	VENC channel ID
	Levell(16)	QP delta level that is added to or subtracted from the start QP value of the current macroblock row to calculate the QP value of the current macroblock when the picture complexity of the current macroblock is between two consecutive thresholds during macroblock-level bit rate control for the I frame Value range: [0, 3]
RUN COMM	ChnId	VENC channel ID



Parameter		Description
PARAM 5	LevelP(16)	QP delta level that is added to or subtracted from the start QP value of the current macroblock row to calculate the QP value of the current macroblock when the picture complexity of the current macroblock is between two consecutive thresholds during macroblock-level bit rate control for the P frame Value range: [0, 3]
RUN COMM PARAM 6	ChnId	VENC channel ID
	LevelB(16)	QP delta level that is added to or subtracted from the start QP value of the current macroblock row to calculate the QP value of the current macroblock when the picture complexity of the current macroblock is between two consecutive thresholds during macroblock-level bit rate control for the B frame Value range: [0, 3]
RUN COMM PARAM 7	ChnId	VENC channel ID
	bLost	Whether to discard frames when the instantaneous bit rate exceeds the threshold
	LostThr	Threshold for discarding frames
	LostFrmStr	Frame discarding policy when the instantaneous bit rate exceeds the threshold
	EncGap	Interval of encoding frames. For details, see the description of VENC_PARAM_LOSTFRM_S in chapter 6 "VENC" of the <i>HiMPP IPC V2.0 Media Processing Software Development Reference</i> .
	RCPriority	Bit rate control priority BITRATE: The target bit rate takes priority. FRAMEBITS: The jumbo frame threshold takes priority.
RUN COMM PARAM 8	ChnId	VENC channel ID
	SprFrmMod	Jumbo frame processing mode
	SprIFrm	Threshold for the jumbo I frame
	SprPFrm	Threshold for the jumbo P frame
	SprBfrm	Threshold for the jumbo B frame
GOP MODE ATTR	ChnId	VENC channel ID
	GopMode	GOP mode
	IpQpDelta	QP delta of the I/P frame Value range: [-10, +30]



Parameter		Description
	SPInterval	Interval of the special P frames Value range: less than or equal to the GOP value. For details, see VENC_CHN_ATTR_S in chapter 6 "VENC" of the <i>HiMPP IPC V2.0 Media Processing Software Development Reference</i> .
	SPQpDelta	QP delta for the special P frame and other P frames Value range: [-10, +30]
	BfrmNum	Number of B frames Value range: [1, 3]
	BQpDelta	QP delta for the B frame and P frame Value range: [-10, +30]
	BgInterval	Interval of Bg frames Value range: greater than or equal to the GOP value
	ViQpDelta	QP delta for the virtual I frame and P frame Value range: [-10, +30]
RUN CBR PARAM		For details, see the description of VENC_PARAM_H264_CBR_S in chapter 6 "VENC" of the <i>HiMPP IPC V2.0 Media Processing Software Development Reference</i> .
RUN VBR PARAM		For details, see the descriptions of VENC_PARAM_H264_VBR_S and VENC_PARAM_H264_AVBR_S in chapter 6 "VENC" of the <i>HiMPP IPC V2.0 Media Processing Software Development Reference</i> .
RUN INFO1	ChnId	VENC channel ID
	InsBr(kbps)	Instant bit rate (in kbit/s)
	InsFr	Instant frame rate
	WatL	Bit rate threshold (for internal debugging)
	CfgBt(kb)	Target size of the current frame (in kbit)
	RealBt(kb)	Actual size of the streams in the previous frame (in kbit)
	IPRatio	Ratio of the I frame size to P frame size
	TarPercent	Current target bit rate of AVBR (percentage)
	StartQp	Start QP
	MinQp	Minimum QP
	MaxQp	Maximum QP
RC	ChnId	VENC channel ID



Parameter	Description	
PERFORMANCE INFO	StaOfstaTim	Time spent by the RC for calculating the configuration information about a frame such as QP before encoding starts
	TotaOfstaTim	Time spent by the RC for calculating the configuration information about n frames to be encoded such as QP before encoding starts
	StaOfEndTim	Time spent by the RC for updating the information about a frame such as the number of bytes after the frame is encoded
	TotaOfEndTim	Time spent by the RC for updating the information about n frames such as the number of bytes after the n frames are encoded
	TotalTime	The total time is calculated as follows: TotalTime = TotaOfstaTim + TotaOfEndTim

12.5 VENC

[Debugging Information]

```
# cat /proc/umap/venc
```

```
[VENC] Version: [Hi3519V100_MPP_V1.0.0.0 B010 Debug], Build Time [Sep 6 2015,  
09:51:16]  
-----MODULE PARAM-----  
VencBufferCache FrameBufRecycle  
0 0  
-----VENC CHN ATTR1-----  
ID Width Height Type ByFrame Timeout Sequence  
0 720 576 96 1 1 2  
LeftBytes LeftFrm CurPacks GopMode Prio  
0 0 0 NormalP 1  
  
----- VENC CHN ATTR 2-----  
VeStr SrcFr TarFr Timeref PixFmt PicAddr  
Y -1 -1 24 YUV420 0x89170000  
  
----- VENC CHN RECEIVE STAT-----  
ID Start StartEx RecvLeft EncLeft  
0 1 0 0 0  
  
----- VENC VPSS QUERY-----
```



ID.	Query	QueryOk	QueryFR	Invld	Full	VbFail	QueryFail
InfoErr	Stop						
0	0	0	0	0	0	0	0
0							0

----- VENC SEND1-----

ID	VpssSnd	VInfErr	OthrSnd	OInfErr	Send	Stop
Full	CropErr	DrectSnd	SizeErr			
0	0	0	12	0	12	0
0	0	0	0			

----- VENC SEND2-----

ID	SendVgs	StartOk	StartFail	IntOk	IntFail	SrcAdd
SrcSub	DestAdd	DestSub				
1	496	474	22	473	0	496
495	474	0				

----- VENC PIC QUEUE STATE-----

ID	Free	Busy	Vgs
0	2	4	0

----- VENC CHNL INFO-----

ID	Inq	InqOk	Start	StartOk	Config	VencInt
ChaResLost	OverLoad	Skip				
1	499	469	69	469	469	468
0	0	0				

----- VENC CROP INFO-----

ID	CropEn	StartX	StartY	Width	Height
0	1	0	0	1280	720

-----VENC STREAM STATE-----

ID	FreeCnt	BusyCnt	UserCnt	UserGet	UserRls	GetTimes	Interval
FrameRate							
0	4	0	0	875		875	850
						15	25

[Analysis]

This section records the attributes and status of the current VENC channel.

[Parameter Description]

Parameter	Description	
MODULE PARAM	VencBufferCache	Whether the encoding stream buffer uses the cache mode 0: no 1: yes



Parameter		Description
	FrameBufRecycle	Whether the idle RefBuffer and AdvSmartPBuffer are recycled during encoding 0: not recycled 1: recycled
VENC CHN ATTR1	ID	VENC channel ID
	Width	VENC channel width
	Height	VENC channel height
	Type	VENC channel type
	ByFrame	Mode of obtaining streams 0: by packet 1: by frame
	Timeout	Timeout period for obtaining streams -1: block mode 0: non-block mode (0, +∞): timeout period
	Sequence	Sequence number When the streams are obtained by frame, it represents the frame sequence number. When the streams are obtained by packet, it represents the packet sequence number.
	LeftBytes	Remaining bytes in a stream buffer
	LeftFrm	Number of remaining stream frames in a stream buffer
	CurPacks	Number of stream packets for the current frame (invalid currently)
VENC CHN ATTR 2	GopMode	GOP mode
	Prio	Channel priority
	VeStr	Whether to start encoding
	SrcFr	Source frame rate (input frame rate) used by the VENC for controlling the frame rate
	TarFr	Target frame rate used by the VENC for controlling the frame rate
	Timeref	Timeref of the latest frame in the busy queue
VENC CHN	PixFmt	Format of the frame that is being encoded
	PicAddr	Address for the frame that is being encoded
VENC CHN	ID	VENC channel ID



Parameter		Description
RECEIVE STAT	Start	Whether the VENC channel starts to receive pictures
	StartEx	Whether the VENC channel starts to receive pictures by frame
	RecvLeft	Number of frames that are to be received by the VENC. This parameter is used with HI_MPI_VENC_StartRecvPicEx().
	EncLeft	Number of frames that are to be encoded by the VENC. This parameter is used with HI_MPI_VENC_StartRecvPicEx().
VENC VPSS QUERY	ID	VENC channel ID
	Query	Total number of times that the VENC is queried by the VPSS
	QueryOk	Number of times that the VENC is successfully queried by the VPSS
	QueryFR	Number of times the VENC discards frames due to frame rate control, which is queried by the VPSS
	Invld	Number of times that the source picture of the VENC is empty, which is queried by the VPSS
	Full	Number of times that the picture queue of the VENC is full, which is queried by the VPSS
	VbFail	Number of times that the VENC fails to request VBs, which is queried by the VPSS
	QueryFail	Number of times that the VPSS fails to query the VENC
	InfoErr	Number of times that the VPSS fails to query the VENC due to incorrect information
	Stop	Number of times that the VPSS fails to query the VENC because the VENC stops receiving pictures
VENC SEND1	ID	VENC channel ID
	VpssSnd	Number of times that the VPSS transmits pictures
	VInfErr	Number of times that the VPSS fails to transmit pictures due to incorrect picture information
	OthrSnd	Number of times that other modules transmit pictures
	OInfErr	Number of times that other modules fail to transmit pictures due to incorrect picture information
	Send	Number of times that external modules transmit pictures properly



Parameter	Description	
	Stop	Number of times that the VENC stops receiving pictures when external modules are transmitting pictures
	Full	Number of times that the picture queue is full when external modules are transmitting pictures
	CropErr	Number of lost frames caused by errors of the crop parameter
	DrectSnd	Number of times that the external modules transmit pictures to the picture busy queue directly (pictures are transmitted to the VGS)
	SizeErr	Number of times that the VENC refuses to receive pictures. This item increases when the width or height of the picture transmitted by the front-end is less than the width or height of the VENC channel.
VENC SEND2	ID	VENC channel ID
	SendVgs	Number of times that pictures are transmitted to the VGS
	StartOk	Number of times that the VGS is started successfully
	StartFail	Number of times that the VGS fails to be started
	IntOk	Number of interrupts indicating that the VGS tasks are performed successfully
	IntFail	Number of interrupts indicating that VGS tasks fail to be performed
	SrcAdd	Number of times that the count of using the source picture VBs increases
	SrcSub	Number of times that the count of using the source picture VBs decreases
	DestAdd	Number of times that the count of using the target picture VBs increases
	DestSub	Number of times that the count of using the target picture VBs decreases
VENC PIC QUEUE STATE	ID	VENC channel ID
	Free	Number of the nodes in the VENC free queue (number of frames allowed)
	Busy	Number of the nodes in the VENC busy queue (allowed number of used frames)
	Vgs	Number of frames being processed by the VGS
VENC CHNL	ID	VENC channel ID



Parameter		Description
INFO	Inq	Number of times that VENC channel is queried
	InqOk	Number of times that VENC channel is queried successfully
	Start	Number of times that the VENC channel starts encoding
	StartOk	Number of times that the VENC channel starts encoding successfully
	Config	Number of times that the VENC channel configures encoding hardware
	VencInt	Number of times that the VENC channel receives interrupts
	ChaResLost	Number of times that frames are discarded due to resolution switching of the VENC channel
	OverLoad	Number of times that CHNL fails to start encoding due to performance insufficiency
VENC CROP INFO	Skip	Number of frames that are discarded for controlling the frame rate or discarded when the instantaneous bit rate is above the threshold during encoding
	ID	VENC channel ID
	CropEn	Whether to enable the cropping function of the VENC channel
	StartX	Start horizontal coordinate of the picture to be cropped
	StartY	Start vertical coordinate of the picture to be cropped
	Width	Width of the cropped picture
VENC STREAM STATE	Height	Height of the cropped picture
	ID	VENC channel ID
	FreeCnt	Number of free nodes in a stream buffer
	BusyCnt	Number of busy nodes in a stream buffer
	UserCnt	Number of user nodes in a stream buffer This number is incremented by 1 each time users obtain a frame successfully. This number decreased by 1 each time users release a frame successfully.
	UserGet	Number of stream packets from which streams are obtained successfully
	UserRls	Number of stream packets whose streams are



Parameter	Description
	released successfully
	GetTimes Number of times that streams are obtained
	Interval Interval of obtaining two frames from the encoder (in μ s)
	FrameRate Number of frames obtained from the encoder in one second, that is, frame rate of the encoder

12.6 H265E

[Debugging Information]

```
# cat /proc/umap/h265e
[H265E] Version: [Hi3519V100_MPP_V1.0.0.0 B010 Debug], Build Time [Sep 6 2015,
09:51:15]
-----MODULE PARAM-----
OnePack
0
-----CHN ATTR-----
ID      MaxWidth   MaxHeight   Width    Height   Profile   C2GEn   BufSize
ByFrame GopMode MaxStrCnt
0        720       576         720      576      mp        Y        829440      Y
NormalP 200
-----PICTURE INFO-----
ID      EncdStart   EncdSucceed   Lost     Disc     Skip     Pskip    Recode   RlsStr
UnrdStr
0        29          29          0        0        0        0        0        29          0
-----STREAM BUFFER-----
ID      Base        RdTail      RdHead      WrTail      WrHead      DataLen
BufFree
0  0xd8300000  0x6de80      0x6de80      0x6de80      0x6de80      0
829376
-----RefParam INFO-----
ID      EnPred      Base        Enhance
0        Y           1           0
-----ROI INFO-----
ID      Index      bRoiEn    bAbsQp     Qp      Width    Height   StartX   StartY   BgSrcFr
BgTarFr
0        0          Y          N        -10      256      256      0          0          30          15
-----Syntax INFO1-----
ID      SlcspltEn   Slcmode     Slcsize    bCrossSlc IntraRefresh enIslice
```



```
RefreshLine QpOfIDR
1      Y     LcuLine        1          N          N          N
3                  51
-----Syntax INFO2-----
ID   DblkEn   Tc    Beta   Saoluma   Saochroma   EntrpyFlag
2      1       0       0       1       1       1
-----Syntax INFO3-----
ID   TimInfFlag   NumInTick   TimeScal   DiffOne
0      N           1           60          1
-----Pu INFO-----
ID   Pu32En   Pu16En   Pu8En   Pu4En   ConsIntra   IntraSmo   IpcmEn   PcmLoFil
NumMergCan
2      Y           Y           Y           0           0           N
1      2
-----Trans INFO-----
ID   TranByPass   TranSkip   CbQpOffset   CrQpOffset
0      0           0           0           0
```

[Analysis]

During multi-channel H.265 encoding, the upper-layer software may not fetch streams in a timely manner. As a result, the stream buffer is insufficient, which causes frame loss. The best way of checking whether frames are lost is to view the **Disc** and **UnrdStr** parameters. **Disc** indicates the number of discarded frames due to stream buffer insufficiency. **UnrdStr** indicates the number of remaining frames in the stream buffer. A larger value indicates streams are fetched in a less timely manner.

[Parameter Description]

Parameter		Description
MODULE PARAM	OnePack	Mode in which streams are obtained 0: Streams are obtained in multiple-packet mode. 1: Streams are obtained in single-packet mode.
CHN ATTR	ID	Channel ID
	MaxWidth	Maximum width of the encoding channel (in pixel)
	MaxHeight	Maximum height of the encoding channel (in pixel)
	Width	Width (in pixel)
	Height	Height (in pixel)
	Profile	Encoding channel profile Mp: main profile
	C2GEn	Color-to-gray enable



Parameter	Description	
	BufSize	Stream buffer size (in byte)
	ByFrame	Whether to obtain streams by frame
	MaxStrCnt	Maximum number of frames in the stream buffer Default value: 200
PICTURE INFO	ID	Channel ID
	EncdStart	Number of received pictures This value is incremented by 1 after the pictures to be encoded are received.
	EncdSucceed	Number of frames that are successfully encoded
	Lost	Number of frames that are discarded during encoding. The discarded frames include: <ul style="list-style-type: none">• Frames discarded due to encoder exceptions or jumbo frames. The number of these frames is the value of Disc.• Frames that are discarded when the configured instantaneous bit rate exceeds the threshold
	Disc	Number of frames discarded for the following reasons: <ul style="list-style-type: none">• Jumbo frames occur.• An exception occurs in the encoder. For example, the encoder crashes.• The encoder has detected stream buffer insufficiency during encoding.
	Pskip	Number of frames that are encoded as Pskip frames
	Recode	Number of frames that are re-encoded by the encoder
	RlsStr	Number of frames that are obtained and released by users
	UnrdStr	Number of frames that are not obtained by users
	STREAM BUFFER	Channel ID
	base	Base address for a stream buffer
	RdTail	Read tail pointer
	RdHead	Read head pointer
	WrTail	Write tail pointer
	WrHead	Write head pointer
	datalen	Data length



Parameter		Description
	buffree	Free buffer size
RefParam INFO	ID	Channel ID
	EnPred	Whether some frames at the base layer are referenced by other frames at the base layer
	Base	Base layer period
	Enhance	Enhance layer period
ROI INFO	ID	Channel ID
	Index	ROI index
	bRoiEn	ROI enable
	bAbsQp	Whether the ROI uses the QP mode
	Qp	QP value configured by the ROI
	Width	ROI width (in pixel)
	Height	ROI height (in pixel)
	StartX	Start horizontal coordinate of the ROI (in pixel)
	StartY	Start vertical coordinate of the ROI (in pixel)
	BgSrcFr	Source frame rate of a non-ROI
	BgTarFr	Target frame rate of a non-ROI
Syntax INFO 1	ID	Channel ID
	SlcsplEn	Slice split enable
	Slcmode	Slice split mode <ul style="list-style-type: none">• ByteNum: split by byte• MbLine: split by macroblock
	Slcsize	Size of each slice When the slice is split by byte, it indicates the number of bytes of each slice. When the slice is split by macroblock, it indicates the number of macroblock lines of each slice.
	bCrossSlc	Whether filtering is performed on the left and top borders of slices
	IntraRefresh	Enable for refreshing the I macroblock in the P frame
	enIslice	Enable for converting the first refreshed I macroblock into Islice 0: disabled 1: enabled



Parameter	Description	
	RefreshLine	Number of refreshed rows in the I macroblock for each frame
	QpOfIDR	QP value of the IDR frame when the IDR frame needs to be requested
Syntax INFO 2	ID	Channel ID
	DblkEn	1 – Value of the syntax element slice_deblocking_filter_disabled_flag
	Tc	Value of the syntax element slice_tc_offset_div2
	Beta	Value of the syntax element slice_beta_offset_div2
	Saoluma	Value of the syntax element slice_sao_luma_flag
	Saochroma	Value of the syntax element slice_sao_chroma_flag
	EntrpyFlag	Value of the syntax element cabac_init_flag
Syntax INFO3	ID	Channel ID
	TimInfFlag	Value of the syntax element timing_info_present_flag
	NumInTick	Value of the syntax element num_units_in_tick
	TimeScal	Value of the syntax element time_scale
	DiffOne	Value of the syntax element num_units_in_tick
Pu INFO	ID	Channel ID
	Pu32En	PU32x32 enable
	Pu16En	PU16x16 enable
	Pu8En	PU8x8 enable
	Pu4En	PU4x4 enable
	ConsIntra	Value of the syntax element constrained_intra_pred_flag
	IntraSmo	Value of the syntax element strong_intra_smoothing_enabled_flag
	IpcmEn	Value of the syntax element pcm_enabled_flag
	PcmLoFil	Value of the syntax element pcm_loop_filter_disabled_flag
	NumMergCan	Maximum number of Merg candidate points
Trans INFO	ID	Channel ID



Parameter	Description
TranByPass	Value of the syntax element transquant_bypass_enabled_flag
TranSkip	Value of the syntax element transform_skip_enabled_flag
CbQpOffset	Value of the syntax element slice_cb_qp_offset
CrQpOffset	Value of the syntax element slice_cr_qp_offset

12.7 Fisheye

[Debugging Information]

```
~ # cat /proc/umap/fisheye
[FISHEYE] Version: [Hi3519V100_MPP_V1.0.0.0 B010 Debug], Build Time [Aug 29
2015, 10:07:38]
-----MODULE PARAM-----
      max_job_num   max_task_num   max_node_num
          128           200           200
-----RECENT JOB INFO-----
SeqNo ModName JobHdl TaskNum State  InSize  OutSize  CostTime HwTime
    0   viu     104     2  Proced  16588800  16588800  13296  13208
    1   viu     103     2  Proced  16588800  16588800  13298  13209
    2   viu     102     2  Proced  16588800  16588800  13298  13207
    3   viu     101     2  Proced  16588800  16588800  13297  13206
    4   viu     100     2  Proced  16588800  16588800  13299  13208
    5   viu      99     2  Proced  16588800  16588800  13299  13207
    6   viu      98     2  Proced  16588800  16588800  13303  13207
    7   viu      97     2  Proced  16588800  16588800  13297  13210
-----MAX WASTE TIME JOB INFO-----
SeqNo ModName JobHdl TaskNum State  InSize  OutSize  CostTime HwTime
    8   viu       6     2  Proced  16588800  16588800  13303  13207
-----MAX WASTE TIME JOB INFO-----
SeqNo ModName JobHdl TaskNum State  InSize  OutSize  CostTime
    8   viu       6     2  Proced  4147200   4147200   99498
-----FISHEYE JOB STATUS-----
Success      Fail      Cancel  AllJobNum   FreeNum   BeginNum  BusyNum
ProcingNum
    233        0        0      128        128        0        0        0
```



-----FISHEYE TASK STATUS-----

Success	Fail	Cancel	AllTaskNum	FreeNum	BusyNum
466	0	0	200	200	0

-----FISHEYE NODE STATUS-----

AllNodeNum	BusyNum	MinFree	MaxInJob	SubmitFail	IntFail
200	0	192	2	0	0

-----FISHEYE INT STATUS-----

IntNum	IntTm	HalProcTm
233	0	0

-----FISHEYE MEM REQ STATUS -----

ReqOk	FreeOk	ReqFail	FreeFail
0	0	0	0

-----FISHEYE CALL VGS COVER STATUS-----

BeginSuc	BeginFail	TaskSuc	TaskFail	EndSuc	EndFail
CbCnt					

236	0	236	0	236	0	233
-----	---	-----	---	-----	---	-----

-----FISHEYE CALL VGS SCALE STATUS-----

BeginSuc	BeginFail	TaskSuc	TaskFail	EndSuc	EndFail
CbCnt					

233	0	466	0	233	0	229
-----	---	-----	---	-----	---	-----

-----FISHEYE CALL CORRECTION STATUS-----

BeginSuc	BeginFail	TaskSuc	TaskFail	EndSuc	EndFail
CbCnt					

233	0	233	0	233	0	233
-----	---	-----	---	-----	---	-----

-----FISHEYE CALL LDC STATUS-----

BeginSuc	BeginFail	TaskSuc	TaskFail	EndSuc	EndFail

0	0	0	0	0	0
---	---	---	---	---	---

[Analysis]

This section records the information about the fisheye module, including information related to the recently accomplished tasks, the most time-consuming task, history statistics, and interrupts. Only Hi3519 V100 supports the fisheye module.

[Parameter Description]



Parameter	Description
MODULE PARAM	max_job_num Maximum number of jobs Default value: 128 This parameter needs to be specified when the module is loaded only if the number of jobs supported by the fisheye module needs to be adjusted.
	max_task_num Maximum number of tasks Default value: 200 This parameter needs to be specified when the module is loaded only if the number of tasks supported by the fisheye module needs to be adjusted.
	max_node_num Maximum number of nodes Default value: 200 This parameter needs to be specified when the module is loaded only if the number of nodes supported by the fisheye module needs to be adjusted.
RECENT JOB INFO	SeqNo Print sequence number Value range: [0, 7]
	ModName Name of the module that submits a job
	JobHdl Handle ID of a job, which is used for debugging Value range: [0, 127]
	TaskNum Number of tasks in a job Value range: [0, 200]
	State Processing status of a job Value range: {ProcErr, Proced, Procing} ProcErr: The job fails to be executed. Proced: The job is successfully executed. Procing: The job is being processed by hardware.
	InSize Total size of the input pictures involved in all the tasks of a job Unit: pixel Each time a task is added to the job, the size of the input pictures of the task is added and the value of InSize is increased. The performance of the fisheye module is affected by the total sizes of the input pictures and output pictures (InSize and OutSize). In general, a larger total size of the input/output



Parameter	Description	
		picture (InSize/OutSize) indicates that the time required for processing the job (CostTime) is longer.
	OutSize	Total size of the output pictures involved in all the tasks of a job Unit: pixel Each time a task is added to the job, the size of the output pictures of the task is added and the value of OutSize is increased.
	CostTime	Time consumption from the time a job is submitted to the time the job is successfully processed Unit: μ s The time consumption contains the processing time of the software, hardware, and interrupt service routine (ISR) relevant to the job. If the performance of the fisheye module is insufficient (for example, too many 360 panoramic correction tasks are submitted and the performance limit is exceeded), the actual value of CostTime may exceed the expected value. For example, if time required for processing a job is greater than 40 ms, intermittence may occur for the previewed picture. In this case, you can check whether the performance of the fisheye module is exceeded.
	HwTime	Time period during which the job is processed in hardware Unit: μ s The time period indicates the processing time of the hardware, which is shorter than CostTime .
MAX WASTE TIME JOB INFO	The items are the same as those of RECENT JOB INFO .	Information about the most time-consuming job among the recent 500 jobs The members are the same as those of RECENT JOB INFO . For details, see the preceding description. When a job consumes more time or the number of jobs is greater than 500, the values of these items are updated. The values reflect the recent running performance of the fisheye module and whether some jobs are not processed by the fisheye module in a timely manner.
FISHEYE JOB	Success	Number of jobs that are successfully processed



Parameter	Description	
STATUS		The value of this parameter is incremented by 1 when a job is successfully processed by hardware.
	Fail	Number of jobs that fail to be processed The value of this parameter is incremented by 1 when the fisheye module fails to submit a job to the driver layer. If the value of this parameter is increased, you can learn the causes by viewing the log.
	Cancel	Number of jobs that are proactively canceled by users The value of this parameter is incremented by 1 each time users call the cancelJob interface. Users call the cancelJob interface when a task fails to be added to a job. If the value of this parameter is increased, you can learn the causes by viewing the log.
	AllJobNum	Number of fisheye job nodes. The value of AllJobNum is consistent with that of max_job_num and is generally set to 128 .
	FreeNum	Number of idle job nodes
	BeginNum	Number of jobs that are created but not submitted (by using the EndJob interface)
	BusyNum	Number of jobs that are submitted (by using the EndJob interface) but not transferred to hardware for processing
FISHEYE TASK STATUS	ProcingNum	Number of jobs that are being processed by hardware
	Success	Number of tasks that are successfully processed A job contains one or more scaling tasks. If a job is successfully processed, all tasks in the job are successfully processed. Therefore, the value of this parameter is increased faster than that of the Success parameter for the job. When a job is successfully processed by hardware, the number of tasks in the job are added to this parameter.
	Fail	Number of tasks that fail to be processed If a job fails to be processed, all tasks in the job fail to be processed. When a job fails to be processed, the number of tasks in the job are added to this parameter. If the value of this parameter is increased, you



Parameter	Description	
	Cancel	Number of tasks that are proactively canceled by users When users call the cancelJob interface to cancel a job, all the tasks in the job are cancelled and the value of this parameter is increased. If the value of this parameter is increased, you can learn the causes by viewing the log.
	AllTaskNum	Number of fisheye task nodes. The value is generally 200.
	FreeNum	Number of idle task nodes
	BusyNum	Number of tasks that are added to a job
	AllNodeNum	Number of task nodes
	BusyNum	Number of occupied nodes
FISHEYE NODE STATUS	MinFree	Minimum number of idle nodes
	MaxInJob	Maximum number of nodes in a job
	SubmitFail	Number of times that a task fails to be submitted due to insufficient nodes
	IntFail	Number of times that a task fails to be submitted in interrupt mode due to insufficient nodes
	IntNum	Number of fisheye interrupts The value of this parameter is incremented by 1 when a job is successfully processed by hardware.
	IntTm	Time for handling fisheye interrupts Unit: μ s IntTm includes the time for handling a fisheye interrupt and calling the callback interface.
FISHEYE INT STATUS	HalProcTm	Time for submitting a job to the driver layer, for internal debugging Unit: μ s
	ReqOk	Number of times that the application for the VB is successful The value of this parameter is incremented by 1 each time the application for the VB is successful.
	FreeOk	Number of times that VBs are released successfully



Parameter	Description	
	The value of this parameter is incremented by 1 each time a VB is released successfully. In general, the value of FreeOk should be consistent with the value of ReqOk , indicating that the memory is not leaked.	
ReqFail	Number of times that the application for the VB fails The value of this parameter is incremented by 1 each time the application for the VB fails. The value is 0 in general.	
FreeFail	Number of times that VBs fail to be released The value of this parameter is incremented by 1 each time the VB fails to be released. The value is 0 in general.	
FISHEYE CALL VGS COVER STATUS	BeginSuc	Number of times that the fisheye module creates a VGS job successfully
	BeginFail	Number of times that the fisheye module fails to create a VGS job
	TaskSuc	Number of times that the fisheye module succeeds in adding tasks to a VGS job
	TaskFail	Number of times that the fisheye module fails to add tasks to a VGS job
	EndSuc	Number of times that the fisheye module submits a VGS job successfully
	EndFail	Number of times that the fisheye module fails to submit a VGS job
	CbCnt	Number of callback times after a VGS Cover overlaying task is complete
FISHEYE CALL VGS SCALE STATUS	BeginSuc	Number of times that the fisheye module creates a VGS job successfully
	BeginFail	Number of times that the fisheye module fails to create a VGS job
	TaskSuc	Number of times that the fisheye module succeeds in adding tasks to a VGS job
	TaskFail	Number of times that the fisheye module fails to add tasks to a VGS job
	EndSuc	Number of times that the fisheye module submits a VGS job successfully
	EndFail	Number of times that the fisheye module fails to submit a VGS job
	CbCnt	Number of callback times after a VGS scaling



Parameter	Description	
	task is complete	
FISHEYE CALL CORRECTION STATUS	BeginSuc	Number of times that the fisheye module creates a fisheye job successfully
	BeginFail	Number of times that the fisheye module fails to create a fisheye job
	TaskSuc	Number of times that the fisheye module succeeds in adding tasks to a fisheye job
	TaskFail	Number of times that the fisheye module fails to add tasks to a fisheye job
	EndSuc	Number of times that the fisheye module submits a fisheye job successfully
	EndFail	Number of times that the fisheye module fails to submit a fisheye job
	CbCnt	Number of callback times after a fisheye correction task is complete
FISHEYE CALL LDC STATUS	BeginSuc	Number of times that the fisheye module creates a LDC job successfully
	BeginFail	Number of times that the fisheye module fails to create a LDC job
	TaskSuc	Number of times that the fisheye module succeeds in adding tasks to a LDC job
	TaskFail	Number of times that the fisheye module fails to add tasks to a LDC job
	EndSuc	Number of times that the fisheye module submits a LDC job successfully
	EndFail	Number of times that the fisheye module fails to submit a LDC job