AW MPP IPC 媒体处理软件开发参考

文档版本号：SDK-V1.0

发布日期：2019-03-30

商标声明


、全志和其他全志商标均为珠海全志科技股份有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受全志公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，全志公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保

前 言

概述

产品版本

产品名称	产品版本
V316	V1.0

读者对象

本文档（本指南）主要适用于以下工程师：

- 技术支持工程师
- 软件开发工程师

修订记录

版本号	修订日期	修订内容
V1.0	2019-03-30	V316 初始化版本

目 录

1. 系统控制 1
1.1. 概述 1
1.2. 功能描述 1
1.2.1. 状态 1
1.2.2. 系统绑定 1
1.2.3. 组件端口数据传递模式 2
1.2.4. 媒体内存分配 4
1.3. API 参考 4
AW_MPI_SYS_Init 4
AW_MPI_SYS_Exit 5
AW_MPI_SYS_SetConf 5
AW_MPI_SYS_GetConf 6
AW_MPI_SYS_Bind 7
AW_MPI_SYS_UnBind 7
AW_MPI_SYS_GetBindbyDest 8
AW_MPI_SYS_GetVersion 8
AW_MPI_SYS_GetCurPts 9
AW_MPI_SYS_InitPtsBase 9
AW_MPI_SYS_SyncPts 10
AW_MPI_SYS_MmzAlloc_Cached 10
AW_MPI_SYS_MmzFree 11
AW_MPI_SYS_MmzFlushCache 11
AW_MPI_SYS_Mmap 12
AW_MPI_SYS_Munmap 12
AW_MPI_SYS_SetReg 13
AW_MPI_SYS_GetReg 13
AW_MPI_SYS_SetProfile 14
AW_MPI_SYS_GetVirMemInfo 14
1.4. 数据类型 15
1.4.1. 视频公共类型 15
VIDEO_FRAME_ S 15
VIDEO_FRAME_INFO_S 17
BITMAP_S 17

1.4.2. 组件公共类型 18
MPPCallbackInfo 18
1.5. 错误码 19
2. 视频输入 20
2.1. 概述 20
2.2. 功能框图 20
2.3. VIPP Buffer 管理和使用 22
2.3.1. ViPP 非绑定情况下 22
2.3.2. ViPP 绑定情况下 22
2.4. API 和状态图 23
2.5. API 接口 24
AW_MPI_VI_CreateVipp 25
AW_MPI_VI_DestoryVipp 25
AW_MPI_VI_SetVippAttr 26
AW_MPI_VI_GetVippAttr 27
AW_MPI_VI_SetVIFreq 28
AW_MPI_VI_EnableVipp 28
AW_MPI_VI_DisableVipp 29
AW_MPI_VI_SetVippFlip 30
AW_MPI_VI_GetVippFlip 31
AW_MPI_VI_SetVippMirror 31
AW_MPI_VI_GetVippMirror 32
AW_MPI_VI_CreateVirChn 33
AW_MPI_VI_DestoryVirChn 34
AW_MPI_VI_SetVirChnAttr 35
AW_MPI_VI_GetVirChnAttr 36
AW_MPI_VI_EnableVirChn 36
AW_MPI_VI_DisableVirChn 37
AW_MPI_VI_GetFrame 38
AW_MPI_VI_ReleaseFrame 39
2.6. 数据结构 40
VI_ATTR_S 40
RGN_ATTR_S 43
RGN_CHN_ATTR_S 44
BITMAP_S 45

VI_OsdMaskRegion..... 48
VIDEO_FRAME_INFO_S..... 51
2.7. 错误码..... 53
3. 视频输出..... 54
3.1. 概述..... 54
3.1.1. 文档目的..... 54
3.1.2. VO 简介..... 55
3.1.3. 术语解释..... 55
3.2. 图层..... 55
3.2.1. 图层操作说明..... 55
3.2.2. 显示输出设备操作说明..... 56
3.2.3. 图层 size 与 crop..... 56
3.2.4. 图层 crop 和 screen_win..... 57
3.2.5. alpha..... 57
3.3. 输出设备介绍..... 58
3.4. 模块状态转换..... 58
3.5. API 接口..... 59
AW_MPI_VO_Enable..... 59
AW_MPI_VO_Disable..... 59
AW_MPI_VO_SetPubAttr..... 60
AW_MPI_VO_GetPubAttr..... 61
AW_MPI_VO_GetHdmiHwMode..... 62
AW_MPI_VO_EnableVideoLayer..... 63
AW_MPI_VO_DisableVideoLayer..... 64
AW_MPI_VO_AddOutsideVideoLayer..... 65
AW_MPI_VO_RemoveOutsideVideoLayer..... 66
AW_MPI_VO_OpenVideoLayer..... 67
AW_MPI_VO_CloseVideoLayer..... 68
AW_MPI_VO_SetVideoLayerAttr..... 69
AW_MPI_VO_GetVideoLayerAttr..... 70
AW_MPI_VO_SetVideoLayerPriority..... 71
AW_MPI_VO_GetVideoLayerPriority..... 72
AW_MPI_VO_SetVideoLayerAlpha..... 73
AW_MPI_VO_GetVideoLayerAlpha..... 74
AW_MPI_VO_EnableChn..... 75

AW_MPI_VO_DisableChn....76
AW_MPI_VO_RegisterCallback....77
AW_MPI_VO_SetChnDispBufNum....79
AW_MPI_VO_GetChnDispBufNum....80
AW_MPI_VO_GetDisplaySize....81
AW_MPI_VO_StartChn....82
AW_MPI_VO_StopChn....82
AW_MPI_VO_PauseChn....83
AW_MPI_VO_ResumeChn....84
AW_MPI_VO_Seek....85
AW_MPI_VO_SetStreamEof....86
AW_MPI_VO_ShowChn....87
AW_MPI_VO_HideChn....88
AW_MPI_VO_GetChnPts....88
AW_MPI_VO_SendFrame....89
AW_MPI_VO_Debug_StoreFrame....90
3.6. 数据结构....91
VO_CHN_ATTR_S....91
VO_PUB_ATTR_S....92
VO_VIDEO_LAYER_ATTR_S....93
VO_VIDEO_LAYER_ALPHA_S....94
VIDEO_FRAME_INFO_S....94
VIDEO_FRAME_S....95
VIDEO_FIELD_E....96
VIDEO_FORMAT_E....96
COMPRESS_MODE_E....97
3.7. 错误码....98
4. 图像拼接....100
4.1. 概述....100
4.1.1. ISE 简介....100
4.1.2. 功能框图....101
4.1.3. ISE 组件 buffer 管理和使用....102
4.1.4. ISE 组件典型应用场景....102
4.2. 状态转换与 API 接口....104
4.2.1. 状态图....104

4.2.2. API 接口 104
AW_MPI_ISE_CreateGroup 104
AW_MPI_ISE_DestroyGroup 105
AW_MPI_ISE_SetGrpAttr 106
AW_MPI_ISE_GetGrpAttr 106
AW_MPI_ISE_CreatePort 107
AW_MPI_ISE_DestroyPort 108
AW_MPI_ISE_GetPortAttr 108
AW_MPI_ISE_SetPortAttr 109
AW_MPI_ISE_Start 110
AW_MPI_ISE_Stop 110
AW_MPI_ISE_GetData 111
AW_MPI_ISE_ReleaseData 112
AW_MPI_ISE_SendPic 112
AW_MPI_ISE_RegisterCallback 113
AW_MPI_ISE_SetISEFreq 114
4.3. 数据结构 114
4.4. 错误码 126
5. 视频编码 128
5.1. 概述 128
5.2. 功能描述 128
5.2.1. 缩放功能 129
5.2.2. 旋转功能 129
5.2.3. 码率控制 129
5.2.4. 裁剪编码 130
5.2.5. ROI 编码 130
5.2.6. 非 ROI 区域低帧率 131
5.2.7. P 帧帧内刷新 131
5.2.8. osd 叠加 131
5.2.9. 输入数据压缩模式省带宽 131
5.3. 状态转换与 API 接口 132
5.3.1. 状态图 132
5.3.2. API 和状态 133
5.3.3. API 参考 139
AW_MPI_VENC_CreateChn 139

AW_MPI_VENC_DestroyChn..... 139
AW_MPI_VENC_ResetChn..... 140
AW_MPI_VENC_StartRecvPic..... 141
AW_MPI_VENC_StartRecvPicEx..... 141
AW_MPI_VENC_StopRecvPic..... 142
AW_MPI_VENC_Query..... 143
AW_MPI_VENC_RegisterCallback..... 143
AW_MPI_VENC_SetChnAttr..... 144
AW_MPI_VENC_GetChnAttr..... 145
AW_MPI_VENC_GetStream..... 145
AW_MPI_VENC_ReleaseStream..... 146
AW_MPI_VENC_SendFrame..... 147
AW_MPI_VENC_RequestIDR..... 148
AW_MPI_VENC_GetHandle..... 148
AW_MPI_VENC_SetRoiCfg..... 149
AW_MPI_VENC_GetRoiCfg..... 150
AW_MPI_VENC_GetH264SpsPpsInfo..... 151
AW_MPI_VENC_GetH265SpsPpsInfo..... 151
AW_MPI_VENC_SetJpegParam..... 152
AW_MPI_VENC_GetJpegParam..... 153
AW_MPI_VENC_SetJpegExifInfo..... 153
AW_MPI_VENC_GetJpegExifInfo..... 154
AW_MPI_VENC_GetJpegThumbBuffer..... 155
AW_MPI_VENC_SetFrameRate..... 155
AW_MPI_VENC_GetFrameRate..... 156
AW_MPI_VENC_SetTimeLapse..... 157
AW_MPI_VENC_GetTimeLapse..... 158
AW_MPI_VENC_SetColor2Grey..... 158
AW_MPI_VENC_GetColor2Grey..... 159
AW_MPI_VENC_SetCrop..... 160
AW_MPI_VENC_GetCrop..... 160
AW_MPI_VENC_GetStreamBufInfo..... 161
AW_MPI_VENC_SetIntraRefresh..... 162
AW_MPI_VENC_GetIntraRefresh..... 162
AW_MPI_VENC_SetSmartP..... 163

AW_MPI_VENC_GetSmartP 164
AW_MPI_VENC_SetBrightness 164
AW_MPI_VENC_GetBrightness 165
AW_MPI_VENC_SetVEFreq 166
AW_MPI_VENC_Set3DNR 166
AW_MPI_VENC_Get3DNR 167
AW_MPI_VENC_GetCacheState 167
AW_MPI_VENC_SetRefParam 168
AW_MPI_VENC_GetRefParam 169
AW_MPI_VENC_SetHorizonFlip 169
AW_MPI_VENC_GetHorizonFlip 170
AW_MPI_VENC_SetAdaptiveIntraInP 171
AW_MPI_VENC_EnableNullSkip 171
AW_MPI_VENC_EnablePSkip 172
AW_MPI_VENC_SaveBsFile 173
AW_MPI_VENC_SetProcSet 173
5.4. 数据结构说明 174
H264E_NALU_TYPE_E 174
JPEGE_PACK_TYPE_E 175
H265E_NALU_TYPE_E 176
VENC_DATA_TYPE_U 177
VENC_PACK_S 177
VENC_STREAM_S 178
VENC_ATTR_H264_S 179
VENC_ATTR_H265_S 181
VENC_ATTR_MJPEG_S 183
VENC_ATTR_JPEG_S 184
VENC_ATTR_S 185
VENC_ATTR_H264_CBR_S 187
VENC_ATTR_H264_VBR_S 187
VENC_ATTR_H264_FIXQP_S 188
VENC_ATTR_MJPEG_CBR_S 189
VENC_ATTR_MJPEG_FIXQP_S 190
VENC_ATTR_H265_CBR_S 190
VENC_ATTR_H265_VBR_S 191

VENC_ATTR_H265_FIXQP_S....192
VENC_ATTR_H265_ABR_S....193
VENC_RC_ATTR_S....194
VENC_CHN_ATTR_S....195
VENC_CHN_STAT_S....196
VENC_EXIFINFO_S....197
VENC_JPEG_THUMB_BUFFER_S....199
VENC_PARAM_JPEG_S....199
VENC_ROI_CFG_S....200
VENC_COLOR2GREY_S....201
VENC_CROP_CFG_S....201
VENC_FRAME_RATE_S....202
VENC_STREAM_BUF_INFO_S....203
VENC_PARAM_INTRA_REFRESH_S....203
VENC_PARAM_REF_S....204
VencHeaderData....205
VencSmartFun....205
VencBrightnessS....206
CacheState....206
VencSaveBSFile....207
VeProcSet....207
5.5. 错误码....208
6. 视频解码....210
6.1. 概述....210
6.2. 功能描述....210
6.3. 状态转换与 API 接口....211
6.3.1. 状态图....211
6.3.2. API 和状态....212
6.4. API 参考....214
AW_MPI_VDEC_CreateChn....214
AW_MPI_VDEC_DestroyChn....215
AW_MPI_VDEC_GetChnAttr....215
AW_MPI_VDEC_StartRecvStream....216
AW_MPI_VDEC_StopRecvStream....217
AW_MPI_VDEC_Pause....217

AW_MPI_VDEC_Resume....218
AW_MPI_VDEC_Seek....219
AW_MPI_VDEC_Query....219
AW_MPI_VDEC_RegisterCallback....220
AW_MPI_VDEC_SetStreamEof....221
AW_MPI_VDEC_ResetChn....221
AW_MPI_VDEC_SetChnParam....222
AW_MPI_VDEC_GetChnParam....223
AW_MPI_VDEC_SendStream....224
AW_MPI_VDEC_GetImage....225
AW_MPI_VDEC_ReleaseImage....225
AW_MPI_VDEC_SetRotate....226
AW_MPI_VDEC_GetRotate....227
AW_MPI_VDEC_ReopenVideoEngine....228
6.5. 数据结构说明....228
VIDEO_MODE_E....228
VDEC_STREAM_S....230
VDEC_DECODE_ERROR_S....231
VDEC_CHN_STAT_S....232
VDEC_CHN_PARAM_S....233
VDEC_PRTCL_PARAM_S....233
6.6. 错误码....234
7. MUX 模块....236
7.1. 概述....236
7.2. 功能描述与使用....236
7.2.1. muxGroup 和 muxChannel....237
7.2.2. 状态图....238
7.2.3. API 和状态....238
7.3. API 参考....240
AW_MPI_MUX_CreateGrp....240
AW_MPI_MUX_DestroyGrp....241
AW_MPI_MUX_StartGrp....241
AW_MPI_MUX_StopGrp....242
AW_MPI_MUX_GetGrpAttr....242
AW_MPI_MUX_SetGrpAttr....243

AW_MPI_MUX_SetH264SpsPpsInfo....244
AW_MPI_MUX_SetH265SpsPpsInfo....244
AW_MPI_MUX_CreateChn....245
AW_MPI_MUX_DestroyChn....246
AW_MPI_MUX_GetChnAttr....246
AW_MPI_MUX_SetChnAttr....247
AW_MPI_MUX_SwitchFd....248
AW_MPI_MUX_RegisterCallback....248
7.4. 数据类型....249
MUX_GRP_ATTR_S....249
MUX_CHN_ATTR_S....250
CdxFdT....251
7.5. 错误码....252
8. DEMUX 模块....254
8.1. 概述....254
8.2. 功能描述与使用....254
8.2.1. 状态图....255
8.2.2. API 和状态....255
8.3. API 参考....258
AW_MPI_DEMUX_CreateChn....258
AW_MPI_DEMUX_DestroyChn....258
AW_MPI_DEMUX_RegisterCallback....259
AW_MPI_DEMUX_SetChnAttr....260
AW_MPI_DEMUX_GetChnAttr....260
AW_MPI_DEMUX_GetMediaInfo....261
AW_MPI_DEMUX_Start....262
AW_MPI_DEMUX_Stop....262
AW_MPI_DEMUX_Pause....263
AW_MPI_DEMUX_ResetChn....264
AW_MPI_DEMUX_Seek....264
AW_MPI_DEMUX_getDmxOutPutBuf....265
AW_MPI_DEMUX_releaseDmxBuf....266
8.4. 数据类型....266
STREAMTYPE_E....266
SOURCETYPE_E....267

CEDARX_MEDIA_TYPE....268
DEMUX_DISABLE_TRACKINFO....269
DEMUX_CHN_ATTR_S....269
DEMUX_VIDEO_STREAM_INFO_S....270
DEMUX_AUDIO_STREAM_INFO_S....271
DEMUX_MEDIA_INFO_S....271
DemuxCompOutputBuffer....272
8.5. 错误码....274
9. 音频....276
9.1. 概述....276
9.2. 功能描述....277
9.2.1. AI 设备状态图....277
9.2.2. AO 设备状态图....278
9.2.3. AI 通道状态图....278
9.2.4. AO 通道状态图....279
9.2.5. AEnc 通道状态图....279
9.2.6. ADec 通道状态图....280
9.2.7. AIO 设备与通道....280
9.3. 音频接口调用流程介绍....281
9.3.1. AI 通道使用流程....281
9.3.2. AO 通道使用流程....282
9.3.3. AEnc 通道调用流程....283
9.3.4. ADec 通道调用流程....283
9.4. API 接口....284
9.4.1. 音频输入....284
AW_MPI_AI_SetPubAttr....284
AW_MPI_AI_GetPubAttr....285
AW_MPI_AI_Enable....285
AW_MPI_AI_Disable....286
AW_MPI_AI_CreateChn....287
AW_MPI_AI_DestroyChn....288
AW_MPI_AI_ResetChn....288
AW_MPI_AI_PauseChn....289
AW_MPI_AI_ResumeChn....290
AW_MPI_AI_EnableChn....290

AW_MPI_AI_DisableChn....291
AW_MPI_AI_GetFrame....292
AW_MPI_AI_ReleaseFrame....293
AW_MPI_AI_SetChnParam....294
AW_MPI_AI_GetChnParam....295
AW_MPI_AI_EnableReSmp....295
AW_MPI_AI_DisableReSmp....296
AW_MPI_AI_SetVqeAttr....297
AW_MPI_AI_GetVqeAttr....298
AW_MPI_AI_EnableVqe....298
AW_MPI_AI_DisableVqe....299
AW_MPI_AI_SetTrackMode....300
AW_MPI_AI_GetTrackMode....301
AW_MPI_AI_ClrPubAttr....301
AW_MPI_AI_SaveFile....302
AW_MPI_AI_QueryFileStatus....303
AW_MPI_AI_SetVqeVolume....304
AW_MPI_AI_GetVqeVolume....304
AW_MPI_AI_RegisterCallback....305
AW_MPI_AI_SetVolume....306
AW_MPI_AI_GetVolume....306
AW_MPI_AI_SetMute....307
AW_MPI_AI_GetMute....308
9.4.2. 音频输出....309
AW_MPI_AO_SetPubAttr....309
AW_MPI_AO_GetPubAttr....310
AW_MPI_AO_ClrPubAttr....310
AW_MPI_AO_Enable....311
AW_MPI_AO_Disable....312
AW_MPI_AO_EnableChn....312
AW_MPI_AO_DisableChn....313
AW_MPI_AO_StartChn....314
AW_MPI_AO_StopChn....315
AW_MPI_AO_RegisterCallback....315
AW_MPI_AO_SendFrame....316

AW_MPI_AO_EnableReSmp..... 317
AW_MPI_AO_DisableReSmp..... 318
AW_MPI_AO_PauseChn..... 319
AW_MPI_AO_ResumeChn..... 319
AW_MPI_AO_Seek..... 320
AW_MPI_AO_ClearChnBuf..... 321
AW_MPI_AO_QueryChnStat..... 321
AW_MPI_AO_SetTrackMode..... 322
AW_MPI_AO_GetTrackMode..... 323
AW_MPI_AO_SetVolume..... 324
AW_MPI_AO_GetVolume..... 324
AW_MPI_AO_SetMute..... 325
AW_MPI_AO_GetMute..... 326
AW_MPI_AO_SetVqeAttr..... 327
AW_MPI_AO_GetVqeAttr..... 328
AW_MPI_AO_EnableVqe..... 329
AW_MPI_AO_DisableVqe..... 329
AW_MPI_AO_SetStreamEof..... 330
AW_MPI_AO_SaveFile..... 331
AW_MPI_AO_QueryFileStatus..... 331
9.4.3. 音频编码..... 332
AW_MPI_AENC_CreateChn..... 332
AW_MPI_AENC_DestroyChn..... 333
AW_MPI_AENC_SendFrame..... 334
AW_MPI_AENC_GetStream..... 334
AW_MPI_AENC_ReleaseStream..... 335
AW_MPI_AENC_StartRecvPcm..... 336
AW_MPI_AENC_StopRecvPcm..... 337
AW_MPI_AENC_ResetChn..... 338
AW_MPI_AENC_Query..... 338
AW_MPI_AENC_RegisterCallback..... 339
AW_MPI_AENC_SetChnAttr..... 340
AW_MPI_AENC_GetChnAttr..... 340
AW_MPI_AENC_GetHandle..... 341
9.4.4. 音频解码..... 342

AW_MPI_ADEC_CreateChn....342
AW_MPI_ADEC_DestroyChn....342
AW_MPI_ADEC_ResetChn....343
AW_MPI_ADEC_RegisterCallback....344
AW_MPI_ADEC_SendStream....344
AW_MPI_ADEC_ClearChnBuf....345
AW_MPI_ADEC_GetFrame....346
AW_MPI_ADEC_ReleaseFrame....346
AW_MPI_ADEC_SetStreamEof....347
AW_MPI_ADEC_StartRecvStream....348
AW_MPI_ADEC_StopRecvStream....349
AW_MPI_ADEC_SetChnAttr....349
AW_MPI_ADEC_GetChnAttr....350
AW_MPI_ADEC_Pause....351
AW_MPI_ADEC_Seek....352
9.5. 数据结构....352
9.5.1. 音频输入输出....352
AIO_ATTR_S....352
AI_CHN_PARAM_S....354
AUDIO_FRAME_S....354
AEC_FRAME_S....355
AUDIO_AGC_CONFIG_S....356
AI_AEC_CONFIG_S....357
AUDIO_ANR_CONFIG_S....358
AUDIO_HPF_CONFIG_S....359
AI_RNR_CONFIG_S....360
AUDIO_EQ_CONFIG_S....360
AI_VQE_CONFIG_S....361
AO_VQE_CONFIG_S....363
AUDIO_STREAM_S....364
AO_CHN_STATE_S....365
AUDIO_FADE_S....365
AUDIO_SAMPLE_RATE_E....366
AUDIO_BIT_WIDTH_E....367
AIO_MODE_E....368

AIO_SOUND_MODE_E..... 369
AUDIO_HPF_FREQ_E..... 369
AQE_WORKSTATE_E..... 370
AUDIO_TRACK_MODE_E..... 371
AUDIO_FADE_RATE_E..... 372
G726_BPS_E..... 373
ADPCM_TYPE_E..... 374
9.5.2. 音频编码..... 375
AENC_CHN_ATTR_S..... 375
9.5.3. 音频解码..... 376
ADEC_CHN_ATTR_S..... 376
9.5.4. 音频编解码器类型与数据格式要求..... 377
9.6. 错误码..... 378
9.6.1. 音频输入错误码..... 378
9.6.2. 音频输出错误码..... 378
9.6.3. 音频编码错误码..... 379
9.6.4. 音频解码错误码..... 379
10. Region 模块..... 381
10.1. 概述..... 381
10.2. 功能描述..... 381
10.2.1. 状态..... 381
10.3. API 参考..... 381
AW_MPI_RGN_Create..... 381
AW_MPI_RGN_Destroy..... 382
AW_MPI_RGN_GetAttr..... 383
AW_MPI_RGN_SetAttr..... 384
AW_MPI_RGN_SetBitMap..... 384
AW_MPI_RGN_AttachToChn..... 385
AW_MPI_RGN_DetachFromChn..... 386
AW_MPI_RGN_SetDisplayAttr..... 387
AW_MPI_RGN_GetDisplayAttr..... 387
10.4. 数据类型..... 388
RGN_TYPE_E..... 388
RGN_AREA_TYPE_E..... 389
OVERLAY_ATTR_S..... 389

OVERLAY_INVERT_COLOR_S 390
OVERLAY_CHN_ATTR_S 391
COVER_CHN_ATTR_S 392
RGN_ATTR_U 393
RGN_CHN_ATTR_U 393
RGN_ATTR_S 394
RGN_CHN_ATTR_S 394
10.5. 错误码 395
11. Proc 调试节点用户指南 397
11.1. ISE 模块 397
11.2. VI 模块 399
11.3. VO 模块 400
11.4. Video Encode 模块 402
11.5. Video Decode 模块 404
11.6. Audio AIO 模块 405
12. Declaration 407

1. 系统控制

1.1. 概述

MPP 系统控制模块，根据芯片特性，完成硬件各个部件的复位、基本初始化工作，同时负责完成 MPP（Media Process Platform 媒体处理平台）系统各个业务模块的初始化、去初始化以及管理 MPP 系统各个业务模块的工作状态、提供当前 MPP 系统的版本信息等功能。

应用程序启动 MPP 业务前，必须完成 MPP 系统初始化工作。同理，应用程序退出 MPP 业务后，也要完成 MPP 系统去初始化工作，释放资源。

1.2. 功能描述

(1)初始化 MPP 组件的运行环境，完成音频输入输出、视频输入输出等硬件设备的初始化配置。

(2)提供绑定组件的接口。

(3)提供媒体内存分配、释放、查询的接口。

1.2.1. 状态

本组件没有内部线程，所以没有状态转换。

1.2.2. 系统绑定

MPP 提供系统绑定接口（AW_MPI_SYS_Bind），即通过数据接收者绑定数据源来建立两者之间的关联关系（只允许数据接收者绑定数据源）。绑定后，数据源生成的数据将自动发送给接收者。绑定关系是相互的，接收者处理完数据，如果传输数据的 Buffer 来自数据源，需归还 Buffer 给数据源。一个组件可以和多个组件建立绑定关系，绑定关系精确到组件端口。

目前 MPP 支持的绑定关系如下表 1-1 所示:

数据源	数据接收者
VI	VO
	VENC
	ISE
ISE	VO
	VENC
VENC	MUX
AI	AO
	AENC
AENC	MUX
DEMUX	VDEC

	ADEC
VDEC	VO
ADEC	AO
AO	AI
CLOCK	AO
	VO
	DEMUX
	VDEC
ISE	VO
	VENC

表 1-1 MPP 支持的绑定关系

附：

下列智能算法组件不支持绑定：

libaiMOD（运动目标检测），

libVLPR（车牌识别），

libeveface（人脸检测）。

1.2.3. 组件端口数据传递模式

MPP 组件有两个端口(inport/outport)，inport 端口用于接收数据，在组件内部线程处理后生成新的数据，添加到输出队列的数据链表中进行管理，等待用户从 outport 端口主动拿数据或通过 outport 自动送到所绑定的下个组件。

组件端口数据传递分为 tunnel 模式和 non-tunnle 模式。自动传递数据到下个组件称为 tunnel 模式，手动管理、传递数据方式称为 non-tunnel 模式。Tunnel 模式及 non-tunnel 模式工作数据传递方式见下面的 1-1 和 1-2 图示。

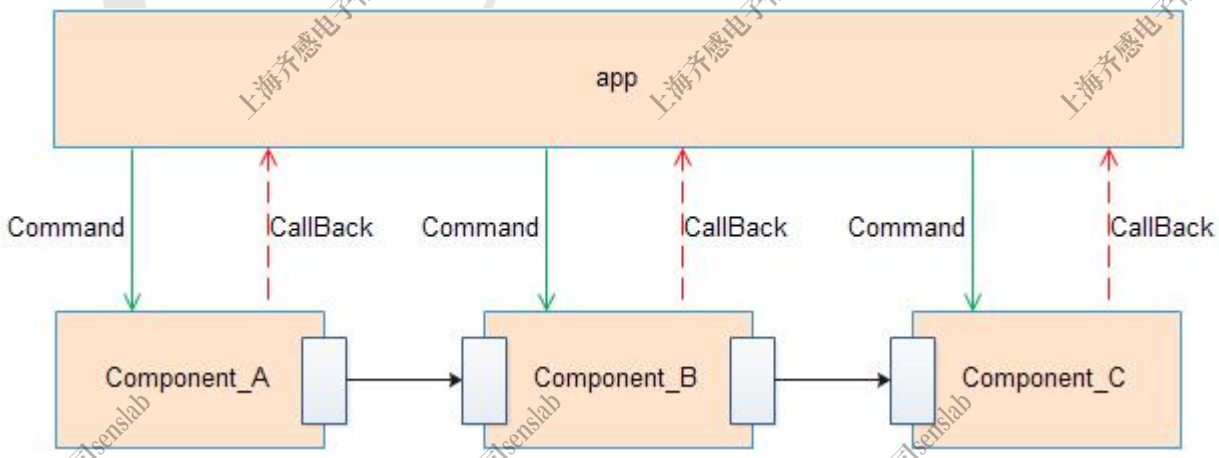


图 1-1 MPP 组件 tunnel 模式

上图显示了组件间 tunnel 模式传递数据的工作原理。应用只需通过几个简单的 command 来创建、

启动、停止、销毁组件，启动命令控制组件内部线程运行起来后，会源源不断地产生数据，并在内部的数据链表中进行统一管理，接下来把生成的数据数据自动发送到下个组件，下个组件内部线程利用输入端口中送来的数据生产出一笔数据，添加到其数据链表中进行管理，接下来将已经使用过的输入端中的数据还给前一个组件，使前一个组件释放该数据占用的 buffer 空间。

例如，当 ai 组件和 aenc 组件绑定时，即意味着 ai 的 outport 和 aenc 的 inport 进行绑定，那么当 ai 通道中存在 pcm 数据时，会自动将数据通过其 outport 端口送到 aenc 的 inport 端口，aenc 组件内部线程进行编码，生成压缩的音频数据后，进行管理，待送到 mux 组件或等待用户取数据，这取决于 aenc 组件输出端口的数据传递模式。

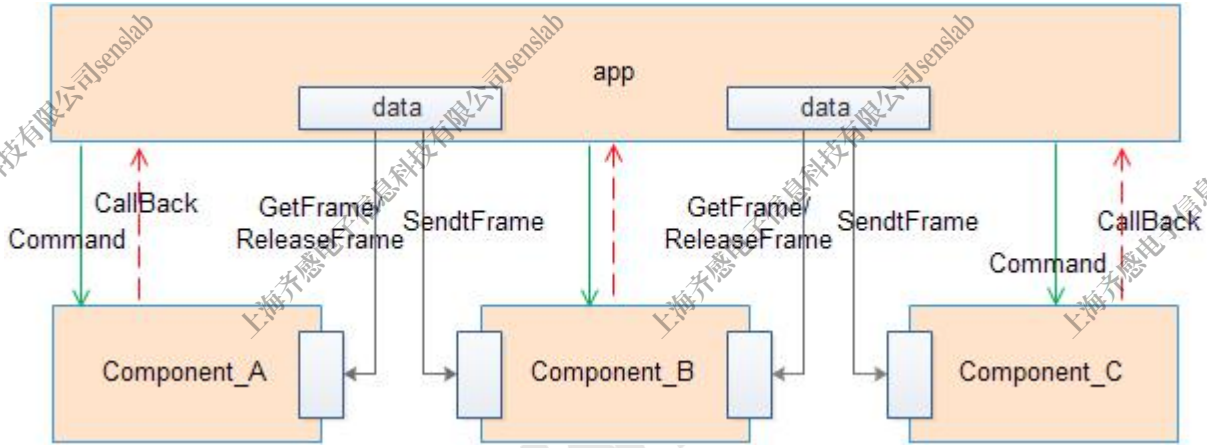


图 1-2 MPP 组件 non-tunnel 模式

上图显示了 non-tunnel 模式的组件间数据传递方式的工作原理。应用创建、启动组件后，需通过 SendFrame()/SendStream() 等接口，往组件的 inport 输入端口送数据，然后应用调用 GetStream()/GetFrame()等接口去取生产出的数据(分为阻塞方式和超时等待方式)，待组件内部线程利用 inport 端的数据生产出数据后，添加到输出数据队列中进行管理，此时应用的取数据函数调用方可退出(阻塞方式)，应用拿到生成的这笔数据进行处理，接下来仍需要利用这笔数据还帧给组件，主动告诉组件应用已经使用完这笔数据，可以释放其占用的 buffer 空间。

Notice：使用 non-tunnel 模式时，应用如果往组件 inport 端口送数据不及时，不会造成“饿死”的严重后果；但如果不及时取走数据和还帧，会导致“撑死”现象，因为内部线程一直源源不断地生产出新的数据，输出缓冲区队列逐渐变满直至爆仓，除非应用不往组件 inport 端口送数据，那么就不会生产出数据，输出缓冲区也不会爆仓。

各组件输入端口和输出端口绑定、非绑定支持如下表 1-2 中所示。

组件类型	输入端	输出端
VI	——	tunnel、non-tunnel
ISE	tunnel、non-tunnel	tunnel、non-tunnel
AI	——	tunnel、non-tunnel

VENC	tunnel、non-tunnel	tunnel、non-tunnel
AENC	tunnel、non-tunnel	tunnel、non-tunnel
VDEC	tunnel、non-tunnel	tunnel、non-tunnel
ADEC	tunnel、non-tunnel	tunnel、non-tunnel
VO	tunnel、non-tunnel	——
AO	tunnel、non-tunnel	tunnel、non-tunnel
MUX	tunnel	——
DEMUX	——	tunnel、non-tunnel

表 1-2 MPP 支持的绑定关系

一般的规律是，source 型组件，其输入端接硬件设备用于获取 raw-data，输出端口支持 tunnel 和 non-tunnel 两种模式；sink 型组件，其输入端支持 tunnel 和 non-tunnel 两种模式，输出端直接接 render 类型硬件设备，进行数据的呈现，如声音的播放、图像的显示；filter 型组件，输入和输出端口都支持 tunnel 和 non-tunnel 两种模式。

1.2.4. 媒体内存分配

用于多媒体处理的物理连续的内存分配，使用 ION 方式。系统控制模块封装了 ION 接口，提供给 APP 使用。

1.3. API 参考

系统控制实现 MPP（Media Process Platform）系统初始化、系统绑定解绑、获取 MPP 版本号等功能。

AW_MPI_SYS_Init

【描述】

初始化 MPP 系统。包括音频输入输出、视频输入输出、视频编码、视频叠加区域、视频侦测分析等都会被初始化。

【语法】

ERRORTYPE AW_MPI_SYS_Init();

【参数】

参数名称	描述	输入/输出
无		

【返回值】

返回值	描述
0	成功

非 0	失败，参见错误码。

【注意】

- 必须先调用 AW_MPI_SYS_SetConf 配置 MPP 系统后才能初始化，否则初始化会失败。
- 如果多次初始化，仍会返回成功，但实际上系统不会对 MPP 的运行状态有任何影响。

【举例】

无。

AW_MPI_SYS_Exit

【描述】

去初始化 MPP 系统。包括音频输入输出、视频输入输出、视频编码、视频叠加区域、视频侦测分析通道等都会被销毁或者禁用。

【语法】

ERRORTYPE AW_MPI_SYS_Exit();

【参数】

参数名称	描述	输入/输出
无		

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【注意】

- 去初始化时，如果有阻塞在 MPI 上的用户进程，则去初始化会失败。如果所有阻塞在 MPI 上的调用都返回，则可以成功去初始化。
- 可以反复去初始化，不返回失败。
- 由于系统去初始化不会销毁音频的编解码通道，因此这些通道的销毁需要用户主动进行。如果创建这些通道的进程退出，则通道随之被销毁。

【举例】

无。

AW_MPI_SYS_SetConf

【描述】

配置系统控制参数。

【语法】

ERRORTYPE AW_MPI_SYS_SetConf(const MPP_SYS_CONF_S* pstSysConf);

【参数】

参数名称	描述	输入/输出
pstSysConf	系统控制参数指针。 静态属性（指只能在系统未初始化、未启用设备或通道时，才能设置的属性）。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【注意】

1. 只有在 MPP 整个系统处于未初始化状态，才可调用此函数配置 MPP 系统，否则会配置失败。

【举例】

无。

AW_MPI_SYS_GetConf

【描述】

获得系统控制参数。

【语法】

ERRORTYPE AW_MPI_SYS_GetConf(MPP_SYS_CONF_S* pstSysConf);

【参数】

参数名称	描述	输入/输出
pstSysConf	系统控制参数指针。 静态属性。	输出

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【注意】

- 必须先调用 AW_MPI_SYS_SetConf 成功后才能获取配置。

【举例】

无。

AW_MPI_SYS_Bind

【描述】

绑定数据源通道端口和数据接收者通道端口。

【语法】

ERRORTYPE AW_MPI_SYS_Bind(MPP_CHN_S* pstSrcChn, MPP_CHN_S* pstDestChn);

【参数】

参数名称	描述	输入/输出
pstSrcChn	源通道指针	输入
pstDestChn	目的通道指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【注意】

无。

【举例】

无。

AW_MPI_SYS_UnBind

【描述】

数据源到数据接收者解绑定接口。

【语法】

ERRORTYPE AW_MPI_SYS_UnBind(MPP_CHN_S* pstSrcChn, MPP_CHN_S* pstDestChn);

【参数】

参数名称	描述	输入/输出
pstSrcChn	源通道指针。	输入
pstDestChn	目的通道指针。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【注意】

无。

【举例】

无。

AW_MPI_SYS_GetBindbyDest

【描述】

获取此通道上绑定的源通道的信息。

【语法】

ERRORTYPE AW_MPI_SYS_GetBindbyDest(MPP_CHN_S* pstDestChn, MPP_CHN_S* pstSrcChn);

【参数】

参数名称	描述	输入/输出
pstDestChn	源通道指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【注意】

- 如果该通道绑定了 2 个以上的源通道（例如 Muxer 模块同时接收音频、视频编码通道的编码数据），只返回最先绑定的源通道信息。

【举例】

无。

AW_MPI_SYS_GetVersion

【描述】

获取 MPP 的版本号。

【语法】

ERRORTYPE AW_MPI_SYS_GetVersion(MPP_VERSION_S* pstVersion);

【参数】

参数名称	描述	输入/输出
pstVersion	版本号描述指针。 动态属性（指在任何时刻都可以设置的属性）。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【注意】

无。

【举例】

无。

AW_MPI_SYS_GetCurPts

【描述】

获取 MPP 的当前时间戳。

【语法】

ERRORTYPE AW_MPI_SYS_GetCurPts(uint64_t* pu64CurPts);

【参数】

参数名称	描述	输入/输出
pu64CurPts	当前时间戳指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【注意】

无。

【举例】

无。

AW_MPI_SYS_InitPtsBase

【描述】

初始化 MPP 的时间戳基准。

【语法】

ERRORTYPE AW_MPI_SYS_InitPtsBase(uint64_t u64PtsBase);

【参数】

参数名称	描述	输入/输出
u64PtsBase	时间戳基准。单位：微秒。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【注意】

- 初始化时间戳基准会将当前系统的时间戳强制置成 u64PtsBase，与系统原有时间戳没有任何约束。因此，建议在媒体业务没有启动时（例如操作系统刚启动），调用这个接口。如果媒体业务已经启动，建议调用 AW_MPI_SYS_SyncPts 进行时间戳微调。

【举例】

无。

AW_MPI_SYS_SyncPts

【描述】

同步 MPP 的时间戳。

【语法】

ERRORTYPE AW_MPI_SYS_SyncPts(uint64_t u64PtsBase);

【参数】

参数名称	描述	输入/输出
u64PtsBase	时间戳基准。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【注意】

- 对当前系统时间戳（微秒级）进行微调，微调后不会出现时间戳回退现象。在多片之间做同步时，由于单板的时钟源误差可能比较大，建议一秒钟进行一次时间戳微调。

【举例】

无。

AW_MPI_SYS_MmzAlloc_Cached

【描述】

在用户态分配 MMZ 内存。内部从 ION 分配物理连续内存。

【语法】

ERRORTYPE AW_MPI_SYS_MmzAlloc_Cached(unsigned int* pu32PhyAddr, void** ppVirtAddr, unsigned int u32Len);

【参数】

参数名称	描述	输入/输出
pu32PhyAddr	物理地址指针	输出
ppVirtAddr	指向虚拟地址指针的指针	输出
u32Len	内存块大小	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【注意】

无。

【举例】

无。

AW_MPI_SYS_MmzFree

【描述】

在用户态释放 MMZ 内存。

【语法】

ERRORTYPE AW_MPI_SYS_MmzFree(unsigned int u32PhyAddr, void* pVirtAddr);

【参数】

参数名称	描述	输入/输出
u32PhyAddr	物理地址	输入
pVirtAddr	虚拟地址指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【注意】

无。

【举例】

无。

AW_MPI_SYS_MmzFlushCache

【描述】

刷新 cache 里的内容到内存并且使 cache 里的内容无效。

【语法】

ERRORTYPE AW_MPI_SYS_MmzFlushCache(unsigned int u32PhyAddr, void* pVitAddr, unsigned int u32Size);

【参数】

参数名称	描述	输入/输出

u32PhyAddr	待操作数据的起始物理地址。	输入
pVitAddr	待操作数据的起始虚拟地址指针。	输入
u32Size	待操作数据的大小。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【注意】

无。

【举例】

无。

AW_MPI_SYS_Mmap

【描述】

存储映射接口。

【语法】

void* AW_MPI_SYS_Mmap(unsigned int u32PhyAddr, unsigned int u32Size);

【参数】

参数名称	描述	输入/输出
u32PhyAddr	需映射的内存单元起始地址。	输入
u32Size	映射的字节数。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【注意】

无。

【举例】

无。

AW_MPI_SYS_Munmap

【描述】

存储反映射接口。

【语法】

ERRORTYPE AW_MPI_SYS_Munmap(void* pVirAddr, unsigned int u32Size);

【参数】

参数名称	描述	输入/输出
pVirAddr	mmap 后返回的地址。	输入
u32Size	映射区的字节长度。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【注意】

无。

【举例】

无。

AW_MPI_SYS_SetReg

【描述】

设置寄存器的值。

【语法】

ERRORTYPE AW_MPI_SYS_SetReg(unsigned int u32Addr, unsigned int u32Value);

【参数】

参数名称	描述	输入/输出
u32Addr	写入的地址。	输入
u32Value	写入的值。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【注意】

无。

【举例】

无。

AW_MPI_SYS_GetReg

【描述】

获取寄存器的值。

【语法】

ERRORTYPE AW_MPI_SYS_GetReg(unsigned int u32Addr, unsigned int* pu32Value);

【参数】

参数名称	描述	输入/输出
u32Addr	物理地址。	输入
pu32Value	此内存地址中的值。	输出

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【注意】

- 必须在解码启动前设置，解码过程中设置无效。如果不设置，解码通道使用默认值。

【举例】

无。

AW_MPI_SYS_SetProfile

【描述】

设置功耗场景。该接口主要用于实现媒体电源域的低功耗方案。该接口通过输入参数控制各个媒体域芯片模块的工作频率；达到控制功耗的目的。

【语法】

ERRORTYPE AW_MPI_SYS_SetProfile(PROFILE_TYPE_E enProfile);

【参数】

参数名称	描述	输入/输出
enProfile	功耗场景类型参数。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【注意】

无。

【举例】

无。

AW_MPI_SYS_GetVirMemInfo

【描述】

根据虚拟地址获取对应的内存信息，包括物理地址及 cached 属性。

【语法】

ERRORTYPE AW_MPI_SYS_GetVirMemInfo(const void* pVitAddr, SYS_VIRMEM_INFO_S* pstMemInfo);

【参数】

参数名称	描述	输入/输出
enProfile	功耗场景类型参数。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【注意】

无。

【举例】

无。

1.4. 数据类型

1.4.1. 视频公共类型

VIDEO_FRAME_ S

【说明】

定义视频原始图像帧结构。

【定义】

```
typedef struct VIDEO_FRAME_S
{
    unsigned int        mWidth;
    unsigned int        mHeight;
    VIDEO_FIELD_E       mField;
    PIXEL_FORMAT_E      mPixelFormat;

    VIDEO_FORMAT_E      mVideoFormat;
    COMPRESS_MODE_E mCompressMode;

    unsigned int        mPhyAddr[3];// Y, U, V; Y, UV; Y, VU
    void*               mpVirAddr[3];
```

```
    unsigned int            mStride[3];

    unsigned int            mHeaderPhyAddr[3];
    void*           mpHeaderVirAddr[3];
    unsigned int            mHeaderStride[3];

    short           mOffsetTop;         /* top offset of show area */
    short           mOffsetBottom;  /* bottom offset of show area */
    short           mOffsetLeft;        /* left offset of show area */
    short           mOffsetRight;       /* right offset of show area */

    uint64_t            mpts; //unit:us
    unsigned int            mTimeRef;

    unsigned int            mPrivateData;
    VIDEO_SUPPLEMENT_S mSupplement;
    int mEnvLV; //environment luminance value
} VIDEO_FRAME_S;
```

【成员】

成员名称	描述
mWidth	装填图像的 buffer 的宽度。
mHeight	装填图像的 buffer 的高度。
mField	帧场模式，目前只支持 VIDEO_FIELD_FRAME。
mPixelFormat	视频图像像素格式。
mVideoFormat	视频图像格式。只支持 VIDEO_FORMAT_LINEAR。未使用。
mCompressMode	视频压缩模式。未使用。
mPhyAddr[3]	视频帧的 yuv 分量的物理地址。
mpVirAddr[3]	视频帧的 yuv 分量的虚拟地址。
mStride[3]	视频帧的 yuv 分量的一行的跨距，单位为字节。
mHeaderPhyAddr[3]	未使用。
mpHeaderVirAddr[3]	未使用。
mHeaderStride[3]	未使用。

mOffsetTop	图像顶部剪裁宽度，单位为像素。是图像帧第一行像素的 Y 坐标。
mOffsetBottom	图像底部剪裁宽度，单位为像素。是图像帧最后一行像素的 Y 坐标加 1。
mOffsetLeft	图像左侧剪裁宽度，单位为像素。是图像帧左侧像素的 X 坐标。
mOffsetRight	图像右侧剪裁宽度，单位为像素。是图像帧右侧像素的 X 坐标加 1。
mpts	视频帧 pts。单位微秒。
mTimeRef	未使用。
mPrivateData	未使用。
mSupplement	未使用。
mEnvLV	采集图像帧时的环境亮度值。

【注意事项】

【相关数据类型及接口】

VIDEO_FRAME_INFO_S

【说明】

定义视频图像帧信息结构体。

【定义】

```
typedef struct VIDEO_FRAME_INFO_S
{
    VIDEO_FRAME_ S VFrame;
    unsigned int mId;     //id identify frame uniquely
} VIDEO_FRAME_INFO_S;
```

【成员】

成员名称	描述
VFrame	视频图像帧。
mId	装填图像帧的 buffer 的 id。

【注意事项】

【相关数据类型及接口】

BITMAP_S

【说明】

定义位图图像信息结构。

【定义】

```
typedef struct BITMAP_S
{
```

```
    PIXEL_FORMAT_E mPixelFormat;    /* Bitmap's pixel format */
    unsigned int mWidth;            /* Bitmap's width */
    unsigned int mHeight;           /* Bitmap's height */
    void* mpData;                   /* Address of Bitmap's data */
} BITMAP_S;
```

【成员】

成员名称	描述
mPixelFormat	位图像素格式，支持 MM_PIXEL_FORMAT_RGB_8888 和 MM_PIXEL_FORMAT_RGB_1555。
mWidth	位图宽度。
mHeight	位图高度。
mpData	位图数据起始虚地址。

【注意事项】

【相关数据类型及接口】

1.4.2. 组件公共类型

MPPCallbackInfo

【说明】

通道的 callback 回调注册信息。

【定义】

```
typedef ERRORTYPE (*MPPCallbackFuncType)(void *cookie, MPP_CHN_S *pChn, MPP_EVENT_TYPE event, void *pEventData);
typedef struct MPPCallbackInfo {
    void *cookie;                   //EyeseeRecorder*
    MPPCallbackFuncType callback;   //MPPCallbackWrapper
} MPPCallbackInfo;
```

【成员】

成员名称	描述
cookie	回调函数的 app 数据结构参数
callback	回调函数类型的指针

【注意事项】

【相关数据类型及接口】

1.5. 错误码

错误代码	宏定义	描述
0xA0028006	ERR_SYS_NULL_PTR	空指针错误
0xA0028010	ERR_SYS_NOTREADY	系统控制属性未配置
0xA0028009	ERR_SYS_NOT_PERM	操作不允许
0xA002800C	ERR_SYS_NOMEM	分配内存失败，如系统内存不足
0xA0028003	ERR_SYS_ILLEGAL_PARAM	参数设置无效
0xA0028012	ERR_SYS_BUSY	系统忙
0xA0028008	ERR_SYS_NOT_SUPPORT	不支持的操作或类型。

2. 视频输入

2.1. 概述

视频输入模块实现的功能：用于接收并解析不同协议（Parallel、MIPI、Sub-lvds、Hispi、Bt601/656/1120、Digital camera）传输过来的图像，通过 ISP 和 VIPP 模块处理后输出。

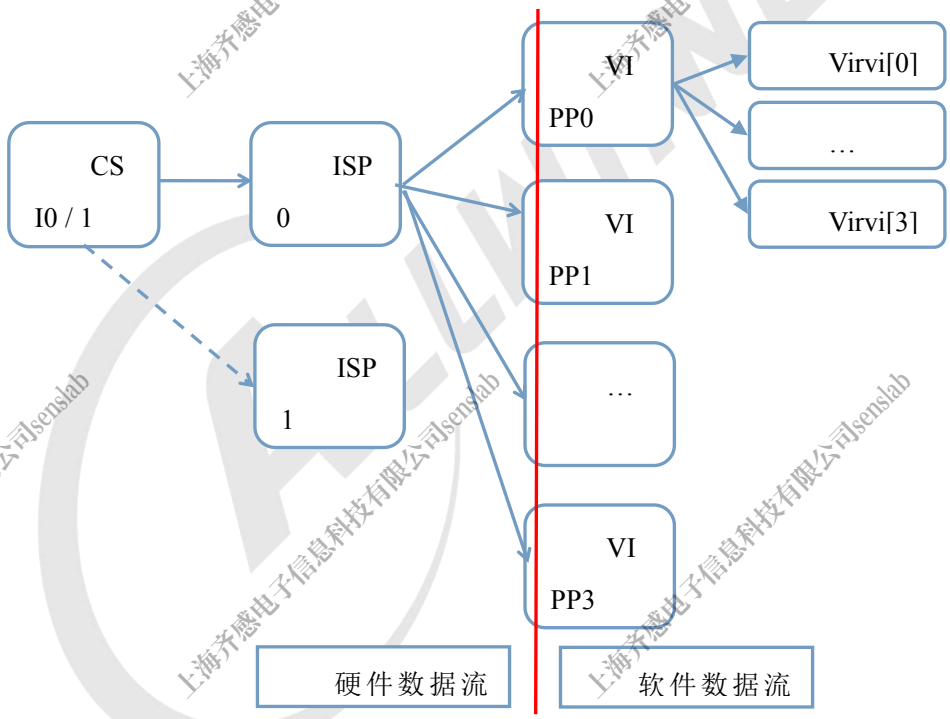
主要功能如下：

- 支持 Combo (Parallel、MIPI、sub-LVDS、Hi-spi)、BT.656、BT1120、BT601、Digital Camera 时序。
- 支持双 ISP：ISP0 : 4224x4224-30Fps, ISP1:3260x3260-30Fps。

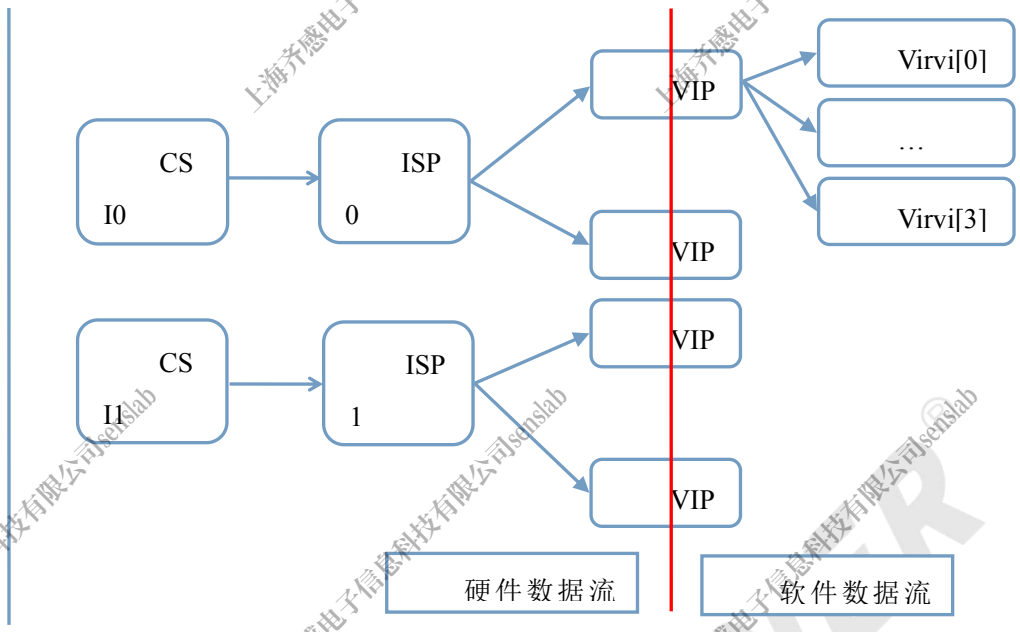
2.2. 功能框图

1) 当支持 1 路 CSI 时，默认通路为：

CSI[0 ~ 1] > ISP0 > VIPP[0 ~ 3] 或者 CSI[0 ~ 1] > ISP1 > VIPP[0 ~ 3]



2) 当支持 2 路 CSI 时，默认通路为：



CSI[0~1]：表示物理 Camera Signal Input Pasrse Device 的接口; CSI0/1 可以选择连接任意一个 ISP

ISP[0~1]：表示物理 ISP; ISP 可以选择连接任意多个 VIPP

VIPP[0~3]：表示物理 Scale + OSD + Mask 通道。每个 VIPP 配合一个 DMA 输出一路 Video 给到 DDR

Virvi[0~3]：表示每个物理通道虚拟 4 个虚通道输出。默认情况下推荐使用一个物理通道和一个虚通道来采集视频数据。虚拟通道的图像属性是物理通道的复制品。

注意：图表中红线表示 DDR。红线左边的硬件模块的配置写在 sys_config.fex 中。可以通过修改 sys_config.fex 实现不同的模块连线。红色右边属于软件。

CSI 输入设备

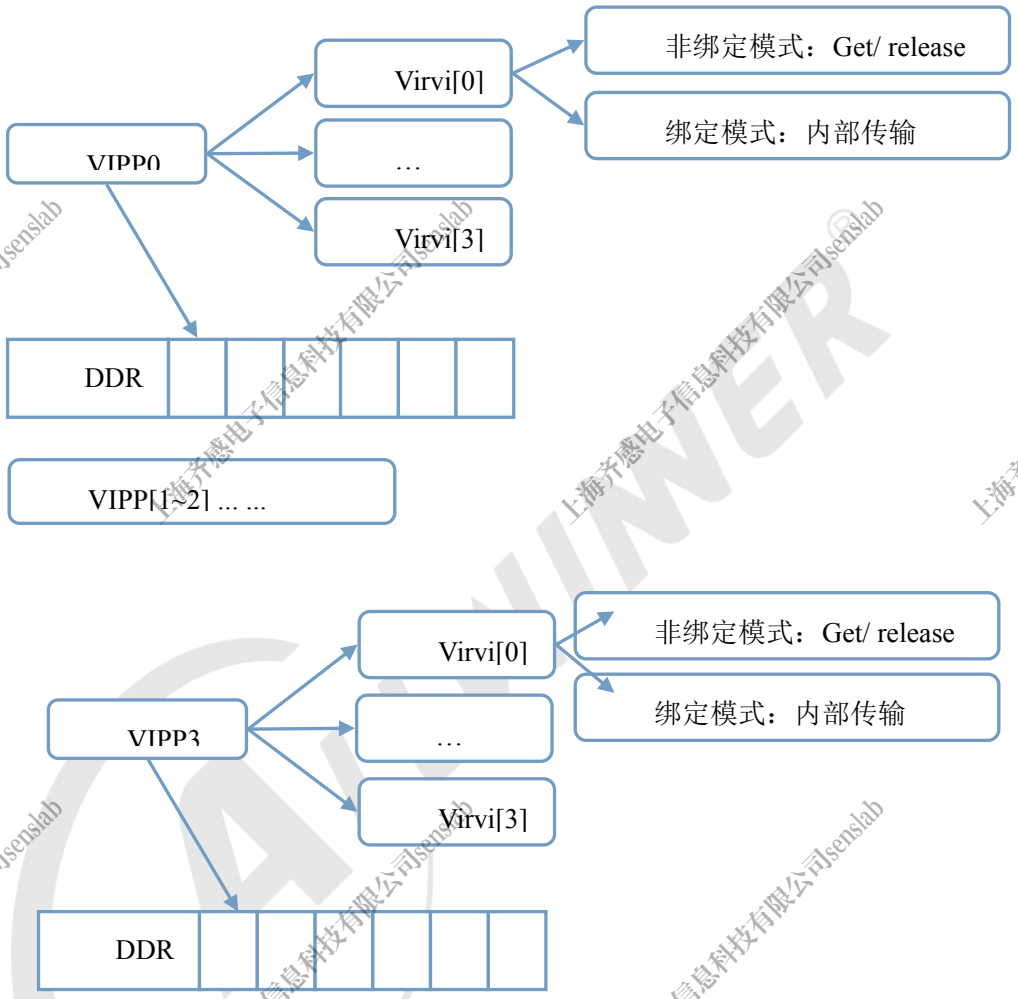
支持不同的时序输入，对输入的时序进行配置和解析

CSI、ISP、VIPP

数据处理不经过 DDR

2.3. VIPP Buffer 管理和使用

2.3.1. ViPP 非绑定情况下



- 每个 VIPP 物理通道对应一段 Buff 空间，Buff 空间数目由 MPI 函数设定，由 Kernel Driver 层统一管理、分配、使用，默认为 5 个 Buff ：ABCDE。
- 同一个 VIPP 设备下所有的虚通道共用同一个 VIPP buff。
- 用户通过 AW_MPI_VI_GetFrame 获取 buff 数据，AW_MPI_VI_ReleaseFrame 释放 buff 数据。必须成对使用。

2.3.2. ViPP 绑定情况下

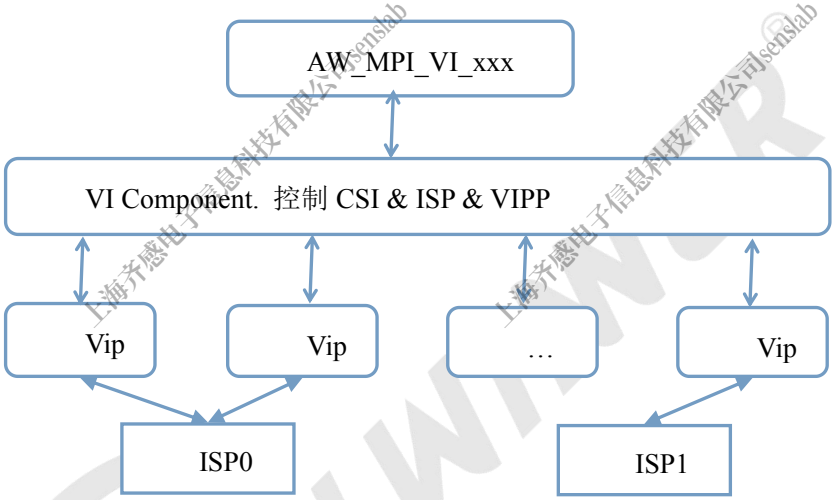
- 绑定情况下的 buff 和非绑定情况下的 buff 分配是一致的。

- 绑定情况下的 Buff 数据在组件直接内部传递。AW_MPI_VI_GetFrame 与 AW_MPI_VI_ReleaseFrame 函数不可使用。

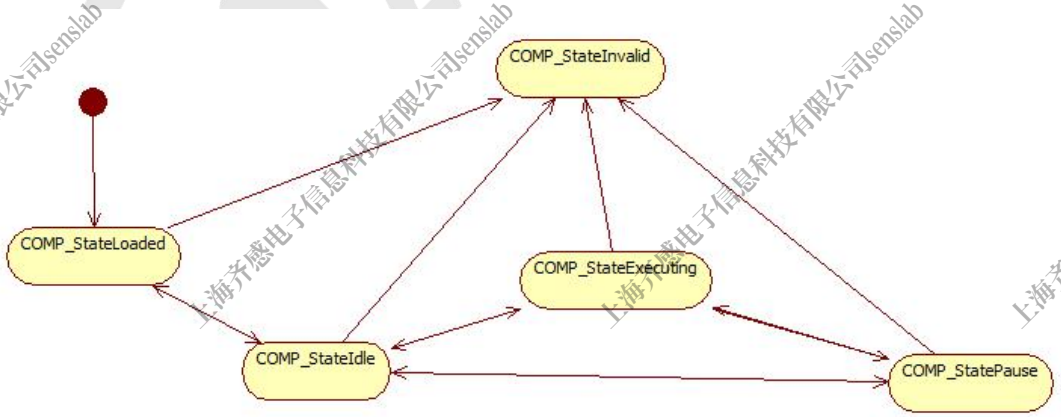
注意：同一个虚拟通道，同一时间只能使用绑定，或者非绑定其中一种方式获取 YUV 数据，不支持两种方式同时存在。

2.4. API 和状态图

1) 程序结构：



2) 状态转换图：



说明：

COMP_StateLoaded: 组件完成初始化。

COMP_StateIdle: 组件准备就绪。

COMP_StateExecuting: 组件运行状态。

COMP_StatePause: 组件暂停（挂起）状态。

COMP_StateInvalid: 组件非法状态。

2.5. API 接口

AW_MPI_VI_CreateVipp 创建 VIPP 物理设备
AW_MPI_VI_DestoryVipp 销毁 VIPP 物理设备
AW_MPI_VI_SetVippAttr 设置 VIPP 物理设备属性
AW_MPI_VI_GetVippAttr 获取 VIPP 物理设备属性
AW_MPI_VI_SetVippMirro 设置 VIPP 水平镜像
AW_MPI_VI_GetVippMirror获取 VIPP 水平镜像
AW_MPI_VI_SetVippFlip 设置 VIPP 垂直镜像
AW_MPI_VI_GetVippFlip 获取 VIPP 垂直镜像
AW_MPI_VI_EnableVipp 启动 VIPP 物理设备
AW_MPI_VI_DisableVipp 停止 VIPP 物理设备

AW_MPI_VI_SetRegion 设置显示区域
AW_MPI_VI_DeleteRegion 更新显示区域
AW_MPI_VI_UpdateOverlayBitmap 更新顶层覆盖的位图数据
AW_MPI_VI_UpdateRegionChnAttr 更新显示区域的通道属性

AW_MPI_VI_SetVippMirror 设置 vipp 的镜像（水平翻转）
AW_MPI_VI_SetVippFlip 设置 vipp 的垂直翻转

AW_MPI_VI_CreateVirChn 基于某个 VIPP，创建虚通道
AW_MPI_VI_DestoryVirChn 销毁虚通道
AW_MPI_VI_GetVirChnAttr 设置虚通道属性
AW_MPI_VI_SetVirChnAttr 获取虚通道属性
AW_MPI_VI_EnableVirChn 启动虚通道
AW_MPI_VI_DisableVirChn 停止虚通道

/* 绑定情况下以下 API 不可用，数据在组件直接内部传递 */
AW_MPI_VI_GetFrame 获取视频帧
AW_MPI_VI_ReleaseFrame 释放视频帧
AW_MPI_VI_Debug_StoreFrame 捕获一帧数据到指定文件夹

AW_MPI_VI_CreateVipp

【目的】

创建一个 VIPP 设备。

【语法】

AW_S32 AW_MPI_VI_CreateVipp(VI_DEV ViDev);

【参数】

参数	描述
ViDev	需要创建的 VIPP 设备号

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_common_vi.h 中的错误码描述。

【需求】

头文件：mpi_vi.h

库文件：libmpp_vi.so

【注意】

VI 所需系统资源（数据源类型、接口、数据位宽、时序、场、输入/输出格式 PIN、CLK）配置完成，系统启动后才会形成/dev/videoX 节点，此处针对节点进行初始化操作。

【举例】

无。

AW_MPI_VI_DestoryVipp

【目的】

销毁 VIPP 物理设备

【语法】

AW_S32 AW_MPI_VI_DestoryVipp(VI_DEV ViDev);

【参数】

参数	描述
ViDev	需要销毁的的VIPP设备号

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_common_vi.h 中的错误码描述。

【需求】

- 头文件：mpi_vi.h
- 库文件：libmpp_vi.so

【注意】

- 该函数会关闭/dev/videoX 设备节点，并销毁 AW_MPI_VI_CreateVipp 函数申请的所有的资源。

【举例】

- 无。

AW_MPI_VI_SetVippAttr

【目的】

设置 VIPP 物理设备属性

【语法】

AW_S32 AW_MPI_VI_SetVippAttr(VI_DEV ViDev, VI_ATTR_S *pstAttr);

【参数】

参数	描述
ViDev	需要设置属性的VIPP设备号
VI_DEV_ATTR_S	属性结构体（静态属性）

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_common_vi.h 中的错误码描述。

【需求】

- 头文件：mpi_vi.h
- 库文件：libmpp_vi.so

【注意】

VIPP设备创建成功后，需要设置format格式、buf数量、nbufs个数、memtype、nplanes、type。参见VI_ATTR_S 结构体描述。

【举例】

```
int ret;
VI_ATTR_S stAttr;

AW_MPI_VI_GetVippAttr(0, &stAttr);
stAttr.format.width    = 1920;
stAttr.format.height = 1080;
stAttr.format.pixelformat      = V4L2_PIX_FMT_NV21M;
stAttr.format.field              = V4L2_FIELD_NONE;
stAttr.fps                         = 30;
stAttr.memtype                   = V4L2_MEMORY_MMAP;
stAttr.nbufs                      = 10;
stAttr.nplanes                   = 2;
stAttr.type                        = V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE;
ret = AW_MPI_VI_SetVippAttr(0, &stAttr);
if (SUCCESS != ret) {
return -1;
}
```

AW_MPI_VI_GetVippAttr

【目的】

获取VIPP物理设备属性

【语法】

```
AW_S32 AW_MPI_VI_GetVippAttr(VI_DEV ViDev, VI_ATTR_S *pstAttr);
```

【参数】

参数	描述
ViDev	需要获取属性的VIPP设备号
VI_DEV_ATTR_S	属性（可以动态获取）

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_common_vi.h 中的错误码描述。

【需求】

头文件：mpi_vi.h

库文件：libmpp_vi.so

【注意】

获取 format 格式、buf 数量、nbufs 个数、memtype、nplanes、type 等。

【举例】

参见上一条的例子。

AW_MPI_VI_SetVIFreq

【目的】

设置VIPP物理设备的运行频率

【语法】

AW_S32 AW_MPI_VI_SetVIFreq(VI_DEV ViDev, int nFreq);

【参数】

参数	描述
ViDev	需要获取属性的VIPP设备号
nFreq	频率值（单位：Hz） 静态属性

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_common_vi.h 中的错误码描述。

【需求】

头文件：mpi_vi.h

库文件：libmpp_vi.so

【注意】

默认 432000000Hz。

【举例】

- 无。

AW_MPI_VI_EnableVipp

【目的】

启动VIPP物理设备

【语法】

AW_S32 AW_MPI_VI_EnableVipp(VI_DEV ViDev);

【参数】

参数	描述
ViDev	使能VIPP设备

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_common_vi.h 中的错误码描述。

【需求】

头文件：mpi_vi.h

库文件：libmpp_vi.so

【注意】

该函数作用是申请 buffer，将所有申请到的 buffer 放入队列，然后开启数据流，创建数据捕获线程。

【举例】

无。

AW_MPI_VI_DisableVipp

【目的】

停止 VIPP 物理设备

【语法】

AW_S32 AW_MPI_VI_DisableVipp(VI_DEV ViDev);

【参数】

参数	描述
ViDev	禁止的VIPP设备号

【返回值】

返回值	描述
SUCCESS	成功

错误码	参考 mm_common_vi.h 中的错误码描述。

【需求】

头文件：mpi_vi.h

库文件：libmpp_vi.so

【注意】

该函数会停止数据流，释放掉所有申请到的 buffer。

【举例】

无。

AW_MPI_VI_SetVippFlip

【目的】

设置 vipp 翻转。

【语法】

AW_S32 AW_MPI_VI_SetVippFlip(VI_DEV ViDev, int Value);

【参数】

参数	描述
ViDev	VIPP 设备号
Value	翻转标志（0：正常；1：翻转） 动态属性

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_common_vi.h 中的错误码描述。

【需求】

头文件：mpi_vi.h

库文件：libmpp_vi.so

【注意】

该函数与 AW_MPI_ISP_SetFlip 函数作用一致。

【举例】

无。

AW_MPI_VI_GetVippFlip

【目的】

获取 vipp 的翻转标志。

【语法】

AW_S32 AW_MPI_VI_GetVippFlip(VI_DEV ViDev, int *Value);

【参数】

参数	描述
ViDev	VIPP设备号
Value	翻转标志 动态属性

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_common_vi.h 中的错误码描述。

【需求】

- 头文件：mpi_vi.h
- 库文件：libmpp_vi.so

【注意】

- 获取到的Value值（0：未翻转；1：翻转），这里说的翻转指的是垂直方向的翻转。

该函数与AW_MPI_ISP_GetFlip函数的作用是一样的。

【举例】

- 无。

AW_MPI_VI_SetVippMirror

【目的】

设置 vipp 的图像水平镜像。

【语法】

AW_S32 AW_MPI_VI_SetVippMirror(VI_DEV ViDev, int Value);

【参数】

参数	描述
ViDev	VIPP 设备号
Value	镜像标志（0：不翻转，1：翻转） 动态属性

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_common_vi.h 中的错误码描述。

【需求】

头文件：mpi_vi.h

库文件：libmpp_vi.so

【注意】

设置的 Value 值 （0：不镜像；1：镜像），这里说的镜像指的是水平方向的翻转。该函数与 AW_MPI_ISP_SetMirror 函数的作用是一样的。

【举例】

无。

AW_MPI_VI_GetVippMirror

【目的】

获取 vipp 的水平镜像标志。

【语法】

AW_S32 AW_MPI_VI_GetVippMirror(VI_DEV ViDev, int *Value);

【参数】

参数	描述
ViDev	VIPP 设备号
Value	镜像标志 动态属性

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_common_vi.h 中的错误码描述。

【需求】

头文件：mpi_vi.h

库文件：libmpp_vi.so

【注意】

获取到的 Value 值 （0：未镜像；1：镜像），这里说的镜像指的是水平方向的翻转。该函数与 AW_MPI_ISP_GetMirror 函数的作用是一样的。

【举例】

无。

AW_MPI_VI_CreateVirChn

【目的】

基于某个VIPP，创建虚拟通道。

【语法】

AW_S32 AW_MPI_VI_CreateVirChn(VI_DEV ViDev, VI_CHN ViCh, void *pAttr);

【参数】

参数	描述
AW_DEV ViDev	已经创建的 VIPP 设备通道 取值范围：[0, 3]
VI_CHN ViCh	需要创建的 VIR 虚拟通道 取值范围：[0, 3]
void *pAttr	NULL 静态属性

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_common_vi.h 中的错误码描述。

【需求】

- 头文件：mpi_vi.h

- 库文件：libmpp_vi.so

【注意】

- 保证VIPP创建后在进行 AW_MPI_VI_CreateVirChn 操作。

【举例】

```
/*declaration*/
int ret = 0;
AW_CHN ViCh;
/* init VI device*/
ret = AW_MPI_VI_InitCh(Videv, ViCh, NULL);
if (SUCCESS != ret)
{
return -1;
}
```

AW_MPI_VI_DestoryVirChn

【目的】

销毁虚拟通道

【语法】

AW_S32 AW_MPI_VI_DestoryVirChn(VI_DEV ViDev, VI_CHN ViCh);

【参数】

参数	描述
AW_DEV ViDev	需要销毁的 VIPP 设备通道 取值范围：[0, 3]
VI_CHN ViCh	需要销毁的 VIR 虚拟通道 取值范围：[0, 3]

P

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_common_vi.h 中的错误码描述。

【需求】

- 头文件：mpi_vi.h
- 库文件：libmpp_vi.so

【注意】

- 无。

【举例】

- 无。

AW_MPI_VI_SetVirChnAttr

【目的】

设置虚拟通道属性

【语法】

AW_S32 AW_MPI_VI_SetVirChnAttr(VI_DEV ViDev, VI_CHN ViCh, void *pAttr);

【参数】

参数	描述
AW_DEV ViDev	VIPP 设备通道 取值范围：[0, 3]
VI_CHN ViCh	VIR 虚拟通道 取值范围：[0, 3]
void *pAttr	Default = NULL 静态属性

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_common_vi.h 中的错误码描述。

【需求】

- 头文件：mpi_vi.h
- 库文件：libmpp_vi.so

【注意】

- 无。

【举例】

- 无。

AW_MPI_VI_GetVirChnAttr

【目的】

获取虚拟通道属性

【语法】

AW_S32 AW_MPI_VI_GetVirChnAttr(VI_DEV ViDev, VI_CHN ViCh, void *pAttr);

【参数】

参数	描述
AW_DEV ViDev	VIPP 设备通道 取值范围：[0, 3]
VI_CHN ViCh	VIR 虚拟通道 取值范围：[0, 3]
void *pAttr	Default=NULL

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_common_vi.h 中的错误码描述。

【需求】

- 头文件：mpi_vi.h
- 库文件：libmpp_vi.so

【注意】

- 无。

【举例】

- 无。

AW_MPI_VI_EnableVirChn

【目的】

启动虚拟通道

【语法】

AW_S32 AW_MPI_VI_EnableVirChn(VI_DEV ViDev, VI_CHN ViCh);

【参数】

参数	描述
ViDev	要使能的 VIPP 设备通道
ViCh	要使能的 VIR 虚拟通道

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_common_vi.h 中的错误码描述。

【需求】

- 头文件：mpi_vi.h
- 库文件：libmpp_vi.so

【注意】

- 无。

【举例】

- 无。

AW_MPI_VI_DisableVirChn

【目的】

停止虚拟通道

【语法】

AW_S32 AW_MPI_VI_DisableVirChn(VI_DEV ViDev, VI_CHN ViCh);

【参数】

参数	描述
ViDev	要禁止的 VIPP 设备通道
ViCh	要禁止的 VIR 虚拟通道

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_common_vi.h 中的错误码描述。

【需求】

- 头文件：mpi_vi.h
- 库文件：libmpp_vi.so

【注意】

- 无。

【举例】

- 无。

AW_MPI_VI_GetFrame

【目的】

获取VI设备一帧图像，属性包括width、height、field、pixelformat、timestamp、index、VirAddr、mem_phy、size等。

【语法】

AW_S32 AW_MPI_VI_GetFrame(VI_DEV ViDev, VI_CHN ViCh, VIDEO_FRAME_INFO_S *pstFrameInfo, AW_S32 s32MilliSec);

【参数】

参数	描述
ViDev	VIPP 设备通道
ViCh	VIR 虚通道
pstFrameInfo	帧信息
s32MilliSec	Timeout超时时间设置 动态属性

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_common_vi.h 中的错误码描述。

【需求】

- 头文件：mpi_vi.h
- 库文件：libmpp_vi.so

【注意】

- 超过s32MilliSec设置的时间值并且还没有获取到帧数据时函数就会返回。

【举例】

- 无。

AW_MPI_VI_ReleaseFrame

【目的】

释放VI设备图像内存资源。

【语法】

AW_S32 AW_MPI_VI_ReleaseFrame(VI_DEV ViDev, VI_CHN ViCh, VIDEO_FRAME_INFO_S *pstFrameInfo);

【参数】

参数	描述
ViDev	VIPP 设备通道
ViCh	VIR 虚通道
VI_FRAME_BUF_INFO_S *pstFrameInfo	帧信息

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_common_vi.h 中的错误码描述。

【需求】

- 头文件：mpi_vi.h
- 库文件：libmpp_vi.so

【注意】

- 无。

【举例】

- 无。

2.6. 数据结构

VI_ATTR_S

【说明】

定义 Vi 输入设备接口属性

【定义】

```
typedef struct awVI_ATTR_S {
enum v4l2_buf_type type;
enum v4l2_memory memtype;
struct v4l2_pix_format_mplane format;
unsigned int nbufs;
unsigned int nplanes;
unsigned int fps;
unsigned int capturemode;
unsigned int use_current_win;
unsigned int wdr_mode;
} VI_ATTR_S;
```

【成员】

成员名称	描述
type	默认值：V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE。 采集数据方式，不能修改
memtype	默认值：V4L2_MEMORY_MMAP。 采集数据内存使用方式，不建议修改
format	struct v4l2_pix_format_mplane { __u32 width; __u32 height; __u32 pixelformat; /* 默认 V4L2_PIX_FMT_NV21M */ __u32 field; __u32 colorspace; struct v4l2_plane_pix_format plane_fmt[VIDEO_MAX_PLANES]; __u8 num_planes; __u8 reserved[11]; } __attribute__((packed));

<table>
<tr><td></td><td>
<table>
<tr><th>pixelformat 的参数数据格式：</th><th>field 的取值</th></tr>
<tr><td>V4L2_PIX_FMT_NV12
V4L2_PIX_FMT_NV21
V4L2_PIX_FMT_NV21M
V4L2_PIX_FMT_NV12M
V4L2_PIX_FMT_YUV420M
V4L2_PIX_FMT_YVU420M
V4L2_PIX_FMT_SBGGR8
V4L2_PIX_FMT_SGBRG8
V4L2_PIX_FMT_SGRBG8
V4L2_PIX_FMT_SRGGB8
V4L2_PIX_FMT_SBGGR10
V4L2_PIX_FMT_SGBRG10
V4L2_PIX_FMT_SGRBG10
V4L2_PIX_FMT_SRGGB10
V4L2_PIX_FMT_SBGGR12
V4L2_PIX_FMT_SGBRG12
V4L2_PIX_FMT_SGRBG12
V4L2_PIX_FMT_SRGGB12
V4L2_PIX_FMT_FBC</td><td>V4L2_FIELD_ANY
V4L2_FIELD_NONE
V4L2_FIELD_TOP
V4L2_FIELD_BOTTOM
V4L2_FIELD_INTERLACED
V4L2_FIELD_SEQ_TB
V4L2_FIELD_SEQ_BT
V4L2_FIELD_ALTERNATE
V4L2_FIELD_INTERLACED_TB
V4L2_FIELD_INTERLACED_BT</td></tr>
</table>
</td></tr>
<tr><td>nbufs</td><td>默认值：5
YUV/RAW 内存节点缓冲个数。</td></tr>
<tr><td>nplanes</td><td>plane 个数，属于返回值，不设置。</td></tr>
<tr><td>fps</td><td>默认值：25。
设置 Sensor 的帧率</td></tr>
<tr><td>capturemode</td><td>默认值：V4L2_MODE_VIDEO
V4L2_MODE_VIDEO
V4L2_MODE_IMAGE
V4L2_MODE_PREVIEW</td></tr>
<tr><td>use_current_win</td><td>0：表示不管之前有没有设置过分辨率，都重新找当前设置分辨率最近的分辨率
1：表示使用之前设置过的分辨率
注意：0 相当于从新设置 sensor 输出的分辨率，1 相当于 sensor 在有输出的情况下，使用后端 VIPP 做视频缩小处理，出不同规格的分辨率。</td></tr>
</table>

wdr_mode	默认值：0 0：normal 1：DOL 2：Sensor Commanding

【注意事项】

无。

【相关数据类型及接口】

```
enum v4l2_buf_type {
        V4L2_BUF_TYPE_VIDEO_CAPTURE        = 1,
        V4L2_BUF_TYPE_VIDEO_OUTPUT         = 2,
        V4L2_BUF_TYPE_VIDEO_OVERLAY        = 3,
        V4L2_BUF_TYPE_VBI_CAPTURE          = 4,
        V4L2_BUF_TYPE_VBI_OUTPUT           = 5,
        V4L2_BUF_TYPE_SLICED_VBI_CAPTURE   = 6,
        V4L2_BUF_TYPE_SLICED_VBI_OUTPUT    = 7,
        V4L2_BUF_TYPE_VIDEO_OUTPUT_OVERLAY = 8,
        V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE = 9,
        V4L2_BUF_TYPE_VIDEO_OUTPUT_MPLANE  = 10,
        /* Deprecated, do not use */
        V4L2_BUF_TYPE_PRIVATE              = 0x80,
};

enum v4l2_memory {
        V4L2_MEMORY_MMAP             = 1,
        V4L2_MEMORY_USERPTR          = 2,
        V4L2_MEMORY_OVERLAY          = 3,
        V4L2_MEMORY_DMABUF           = 4,
};

enum v4l2_field {
        V4L2_FIELD_ANY           = 0, /* driver can choose from none, top, bottom, interlaced
                                         depending on whatever it thinks is approximate ... */
        V4L2_FIELD_NONE          = 1, /* this device has no fields ... */
        V4L2_FIELD_TOP           = 2, /* top field only */
```

```
    V4L2_FIELD_BOTTOM          = 3, /* bottom field only */
    V4L2_FIELD_INTERLACED      = 4, /* both fields interlaced */
    V4L2_FIELD_SEQ_TB          = 5, /* both fields sequential into one buffer, top-bottom order
*/
    V4L2_FIELD_SEQ_BT          = 6, /* same as above + bottom-top order */
    V4L2_FIELD_ALTERNATE       = 7, /* both fields alternating into separate buffers */
    V4L2_FIELD_INTERLACED_TB = 8, /* both fields interlaced, top field first and the top field is
                    transmitted first */
    V4L2_FIELD_INTERLACED_BT = 9, /* both fields interlaced, top field first and the bottom
field is
                    transmitted first */
};
```

RGN_ATTR_S

【说明】

定义 region 的属性。

【定义】

```
typedef struct RGN_ATTR_S
{
    RGN_TYPE_E enType;    /* region type */
    RGN_ATTR_U unAttr;    /* region attribute */
} RGN_ATTR_S;
```

【成员】

成员名称	描述
enType	region 的类型。取值有以下几种 OVERLAY_RGN COVER_RGN COVEREX_RGN OVERLAYEX_RGN
unAttr	region 属性（共用体类型）。 typedef union RGN_ATTR_U { OVERLAY_ATTR_S stOverlay; OVERLAYEX_ATTR_S stOverlayEx; } RGN_ATTR_U;

【注意事项】

无。

【相关数据类型及接口】

```
typedef struct OVERLAY_ATTR_S
{
    /* bitmap pixel format,now only support ARGB1555 or ARGB4444 */
    PIXEL_FORMAT_E mPixelFmt;

    /* background color, pixel format depends on "enPixelFmt" */
    unsigned int mBgColor;

    /* region size,W:[4,4096],align:2,H:[4,4096],align:2 */
    SIZE_S mSize;
}OVERLAY_ATTR_S;
```

RGN_CHN_ATTR_S

【说明】

定义 region 的属性。

【定义】

```
typedef struct RGN_CHN_ATTR_S
{
    BOOL                bShow;
    RGN_TYPE_E          enType;       /* region type */
    RGN_CHN_ATTR_U      unChnAttr;    /* region attribute */
} RGN_CHN_ATTR_S;
```

【成员】

成员名称	描述
bShow	bool 类型，表示是否隐藏。
enType	参见上面一条的描述
unChnAttr	通道 region 属性（共用体类型）。 typedef union RGN_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;

【注意事项】

无。

【相关数据类型及接口】

```
typedef struct OVERLAY_CHN_ATTR_S
{
    /* X:[0,4096],align:4,Y:[0,4096],align:4 */
    POINT_S stPoint;

    /* background an foreground transparence when pixel format is ARGB1555
     * the pixel format is ARGB1555,when the alpha bit is 1 this alpha is value!
     * range:[0,128]
     */
    unsigned int mFgAlpha;

    /* background an foreground transparence when pixel format is ARGB1555
     * the pixel format is ARGB1555,when the alpha bit is 0 this alpha is value!
     * range:[0,128]
     */
    unsigned int mBgAlpha;

    unsigned int mLayer;    /* OVERLAY region layer range:[0,7]*/

    OVERLAY_QP_INFO_S stQpInfo;

    OVERLAY_INVERT_COLOR_S stInvertColor;
}OVERLAY_CHN_ATTR_S;
```

BITMAP_S

【说明】

位图数据描述结构体

【定义】

```
typedef struct BITMAP_S
{
    PIXEL_FORMAT_E mPixelFormat;    /* Bitmap's pixel format */
    unsigned int mWidth;            /* Bitmap's width */
    unsigned int mHeight;           /* Bitmap's height */
    void* mpData;                   /* Address of Bitmap's data */
} BITMAP_S;
```

【成员】

成员名称	描述
mPixelFormat	位图的像素格式，参见该项的【相关数据类型及接口】
mWidth	位图的宽
mHeight	位图的高
mpData	位图的像素数据

【注意事项】

无。

【相关数据类型及接口】

```
typedef enum PIXEL_FORMAT_E
{
    MM_PIXEL_FORMAT_RGB_1BPP = 0,
    MM_PIXEL_FORMAT_RGB_2BPP,
    MM_PIXEL_FORMAT_RGB_4BPP,
    MM_PIXEL_FORMAT_RGB_8BPP,
    MM_PIXEL_FORMAT_RGB_444,

    MM_PIXEL_FORMAT_RGB_4444,
    MM_PIXEL_FORMAT_RGB_555,
    MM_PIXEL_FORMAT_RGB_565,
    MM_PIXEL_FORMAT_RGB_1555,

    /* 9 reserved */
    MM_PIXEL_FORMAT_RGB_888,
```

```
    MM_PIXEL_FORMAT_RGB_8888,

    MM_PIXEL_FORMAT_RGB_PLANAR_888,
    MM_PIXEL_FORMAT_RGB_BAYER_8BPP,
    MM_PIXEL_FORMAT_RGB_BAYER_10BPP,
    MM_PIXEL_FORMAT_RGB_BAYER_12BPP,
    MM_PIXEL_FORMAT_RGB_BAYER_14BPP,

    MM_PIXEL_FORMAT_RGB_BAYER,          /* 16 bpp */

    MM_PIXEL_FORMAT_YUV_A422,
    MM_PIXEL_FORMAT_YUV_A444,

    MM_PIXEL_FORMAT_YUV_PLANAR_422,
    MM_PIXEL_FORMAT_YUV_PLANAR_420,     //YU12
    MM_PIXEL_FORMAT_YUV_PLANAR_444,
    MM_PIXEL_FORMAT_YUV_SEMIPLANAR_422,     //NV16
    MM_PIXEL_FORMAT_YUV_SEMIPLANAR_420,     //NV12
    MM_PIXEL_FORMAT_YUV_SEMIPLANAR_444,

    MM_PIXEL_FORMAT_UYVY_PACKAGE_422,
    MM_PIXEL_FORMAT_YUYV_PACKAGE_422,
    MM_PIXEL_FORMAT_VYUY_PACKAGE_422,
    MM_PIXEL_FORMAT_YCbCr_PLANAR,

    MM_PIXEL_FORMAT_SINGLE,

    MM_PIXEL_FORMAT_YVU_PLANAR_420,     //YV12
    MM_PIXEL_FORMAT_YVU_SEMIPLANAR_422,     //NV61
    MM_PIXEL_FORMAT_YVU_SEMIPLANAR_420,     //NV21

    MM_PIXEL_FORMAT_YUV_AW_AFBC, //by andy

    MM_PIXEL_FORMAT_BUTT
} PIXEL_FORMAT_E;
```

VI_OsdMaskRegion

【说明】

定义 OSD 和视频遮挡属性

【定义】

```
typedef struct awVI_OsdMaskRegion {
    int clipcount; /* number of clips */
    int chromakey;   //V4L2_PIX_FMT_RGB32
    int global_alpha;
    void *bitmap[64];
    struct rect region[64]; /* overlay or cover win */
} VI_OsdMaskRegion;
```

【成员】

成员名称	描述
clipcount	OSD&视频遮挡数目
chromakey	只支持以下图片格式： V4L2_PIX_FMT_RGB555 V4L2_PIX_FMT_RGB444 V4L2_PIX_FMT_RGB32
global_alpha	全局 Alpa 值
*bitmap[64]	1. OSD：图片地址 2. Mask：必须设置为 Null
region[64]	图片位置大小

【注意事项】

无。

【相关数据类型及接口】

```
typedef enum VIDEO_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;

typedef enum PIXEL_FORMAT_E
{
    MM_PIXEL_FORMAT_RGB_1BPP = 0,
    MM_PIXEL_FORMAT_RGB_2BPP,
    MM_PIXEL_FORMAT_RGB_4BPP,
    MM_PIXEL_FORMAT_RGB_8BPP,
    MM_PIXEL_FORMAT_RGB_444,

    MM_PIXEL_FORMAT_RGB_4444,
    MM_PIXEL_FORMAT_RGB_555,
    MM_PIXEL_FORMAT_RGB_565,
    MM_PIXEL_FORMAT_RGB_1555,

    /*   9 reserved */
    MM_PIXEL_FORMAT_RGB_888,
    MM_PIXEL_FORMAT_RGB_8888,

    MM_PIXEL_FORMAT_RGB_PLANAR_888,
    MM_PIXEL_FORMAT_RGB_BAYER_8BPP,
    MM_PIXEL_FORMAT_RGB_BAYER_10BPP,
    MM_PIXEL_FORMAT_RGB_BAYER_12BPP,
    MM_PIXEL_FORMAT_RGB_BAYER_14BPP,

    MM_PIXEL_FORMAT_RGB_BAYER,          /* 16 bpp */

    MM_PIXEL_FORMAT_YUV_A422,
    MM_PIXEL_FORMAT_YUV_A444,

    MM_PIXEL_FORMAT_YUV_PLANAR_422,
    MM_PIXEL_FORMAT_YUV_PLANAR_420,     //YU12

    MM_PIXEL_FORMAT_YUV_PLANAR_444,
```

```
        MM_PIXEL_FORMAT_YUV_SEMIPLANAR_422,     //NV16
        MM_PIXEL_FORMAT_YUV_SEMIPLANAR_420,     //NV12
        MM_PIXEL_FORMAT_YUV_SEMIPLANAR_444,

        MM_PIXEL_FORMAT_UYVY_PACKAGE_422,
        MM_PIXEL_FORMAT_YUYV_PACKAGE_422,
        MM_PIXEL_FORMAT_VYUY_PACKAGE_422,
        MM_PIXEL_FORMAT_YCbCr_PLANAR,

        MM_PIXEL_FORMAT_SINGLE,

        MM_PIXEL_FORMAT_YVU_PLANAR_420,     //YV12
        MM_PIXEL_FORMAT_YVU_SEMIPLANAR_422,     //NV61
        MM_PIXEL_FORMAT_YVU_SEMIPLANAR_420,     //NV21

        MM_PIXEL_FORMAT_BUTT
    } PIXEL_FORMAT_E;
    typedef enum VIDEO_FORMAT_E
    {
        VIDEO_FORMAT_LINEAR     = 0x0,      /* nature video line */
        VIDEO_FORMAT_TILE       = 0x1,      /* tile cell: 256pixel x 16line, default tile mode
*/
        VIDEO_FORMAT_TILE64     = 0x2,      /* tile cell: 64pixel x 16line */

        VIDEO_FORMAT_BUTT
    } VIDEO_FORMAT_E;
    typedef enum COMPRESS_MODE_E
    {
        COMPRESS_MODE_NONE      = 0x0,      /* no compress */
        COMPRESS_MODE_SEG       = 0x1,      /* compress unit is 256 bytes as a segment,
default seg mode */
        COMPRESS_MODE_SEG128    = 0x2,      /* compress unit is 128 bytes as a segment */
        COMPRESS_MODE_LINE      = 0x3,      /* compress unit is the whole line */
        COMPRESS_MODE_FRAME     = 0x4,      /* compress unit is the whole frame */
```

```
    COMPRESS_MODE_BUTT
} COMPRESS_MODE_E;

typedef struct VIDEO_FRAME_S
{
    unsigned int        mWidth;
    unsigned int        mHeight;
    VIDEO_FIELD_E   mField;
    PIXEL_FORMAT_E  mPixelFormat;

    VIDEO_FORMAT_E  mVideoFormat;
    COMPRESS_MODE_E mCompressMode;

    unsigned int        mPhyAddr[3];// Y, U, V; Y, UV; Y, VU
    void*           mpVirAddr[3];
    unsigned int        mStride[3];

    short           mOffsetTop;         /* top offset of show area */
    short           mOffsetBottom;  /* bottom offset of show area */
    short           mOffsetLeft;        /* left offset of show area */
    short           mOffsetRight;       /* right offset of show area */

    uint64_t            mpts; //unit:us
    unsigned int        mTimeRef;

    unsigned int        mPrivateData;
} VIDEO_FRAME_S;
```

VIDEO_FRAME_INFO_S

【说明】

VI 视频输出属性

【定义】

```
typedef struct VIDEO_FRAME_INFO_S
{
    VIDEO_FRAME_S VFrame;
```

```
    unsigned int mId;
} VIDEO_FRAME_INFO_S;
```

【成员】

成员名称	描述
VFrame	Buf 数据信息结构属性
mId	Buf 唯一 ID 号

【注意事项】

无。

【相关数据类型及接口】

```
typedef struct VIDEO_FRAME_S
{
    unsigned int        mWidth;
    unsigned int        mHeight;
    VIDEO_FIELD_E       mField;
    PIXEL_FORMAT_E      mPixelFormat;

    VIDEO_FORMAT_E      mVideoFormat;
    COMPRESS_MODE_E mCompressMode;

    unsigned int        mPhyAddr[3]; // Y, U, V 或 Y, UV 或  Y, VU
    void*               mpVirAddr[3];           //虚拟地址，对应于物理地址
    unsigned int        mStride[3];

    unsigned int        mHeaderPhyAddr[3];
    void*               mpHeaderVirAddr[3];
    unsigned int        mHeaderStride[3];

    short               mOffsetTop;         /* top offset of show area */
    short               mOffsetBottom;  /* bottom offset of show area */
    short               mOffsetLeft;        /* left offset of show area */
    short               mOffsetRight;       /* right offset of show area */

    uint64_t            mpts; //unit:us
```

```
        unsigned int            mTimeRef;

        unsigned int            mPrivateData;
        VIDEO_SUPPLEMENT_S mSupplement;
        int mEnvLV; //environment luminance value
    } VIDEO_FRAME_S;
```

2.7. 错误码

错误码	宏定义	描述
0xA0108002	ERR_VI_INVALID_CHNID	无效的 VI 通道号
0xA0108003	ERR_VI_INVALID_PARA	无效的参数
0xA0108006	ERR_VI_INVALID_NULL_PTR	空指针
0xA0108007	ERR_VI_FAILED_NOTCONFIG	模块未配置
0xA0108008	ERR_VI_NOT_SUPPORT	设备不支持
0xA0108009	ERR_VI_NOT_PERM	不允许
0xA0108001	ERR_VI_INVALID_DEVID	无效的 VI 设备号
0xA010800C	ERR_VI_NOMEM	无可用的内存
0xA010800E	ERR_VI_BUF_EMPTY	数据缓冲区为空
0xA010800F	ERR_VI_BUF_FULL	数据缓冲区为满
0xA0108010	ERR_VI_SYS_NOTREADY	系统还未准备好
0xA0108012	ERR_VI_BUSY	VI 设备正忙
0xA0108041	ERR_VI_FAILED_NOTENABLE	设备未使能
0xA0108042	ERR_VI_FAILED_NOTDISABLE	设备未禁止（处于使能状态）
0xA0108040	ERR_VI_CFG_TIMEOUT	配置超时
0xA0108043	ERR_VI_NORM_UNMATCH	不匹配
0xA0108044	ERR_VI_INVALID_PHYCHNID	无效的物理通道
0xA0108045	ERR_VI_FAILED_NOTBIND	设备未绑定
0xA0108046	ERR_VI_FAILED_BINDED	设备已经绑定
0xA0108047	ERR_VI_UNEXIST	VI 设备不存在
0xA0108048	ERR_VI_EXIST	VI 设备已经存在
0xA0108014	ERR_VI_SAMESTATE	状态相同（常见于状态转换）
0xA0108015	ERR_VI_INVALIDSTATE	无效的状态
0xA0108016	ERR_VI_INCORRECT_STATE_TRANSITION	不正确的状态转换
0xA0108017	ERR_VI_INCORRECT_STATE_OPERATION	不正确的状态操作

3. 视频输出

3.1. 概述

3.1.1. 文档目的

介绍 VO 模块的使用方式，以供开发人员可以快速根据本文档进行基于 VO 模块的开发。

3.1.2. VO 简介

VO 模块主要处理与视频输出显示相关的功能，主要功能如下：

- 支持 linux 标准的 framebuffer 接口
- 支持 lcd(hv/lvds/cpu/dsi)输出
- 支持多图层叠加混合处理
- 支持多种显示效果处理（alpha, colorkey, 图像增强，亮度/对比度/饱和度/色度调整）
- 支持智能背光调节
- 支持多种图像数据格式输入(argb,yuv)
- 支持图像缩放处理
- 支持截屏
- 支持图像转换

3.1.3. 术语解释

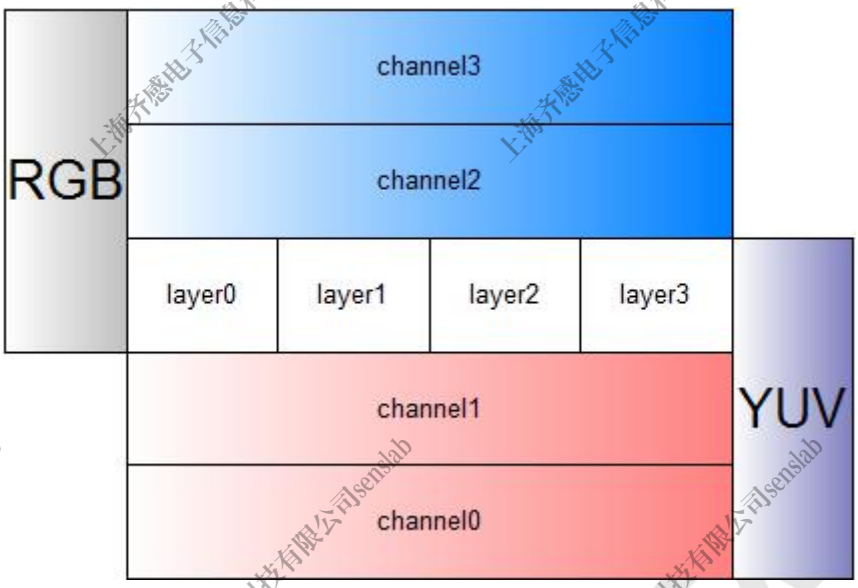
说明文中涉及的专业术语。

3.2. 图层

3.2.1. 图层操作说明

显示中最重要的资源是图层，VO 中支持 2 路显示通道，第 0 路支持 16 个图层（其中视频图层 4 个，3 个 Blending 通道）；第 1 路一般支持 8 个图层（其中视频图层 4 个，1 个 Blending 通道），所有图层都支持缩放。图层由 disp、channel、layer_id 三个索引唯一确定（disp:0/1,channel:0/1/2/3，layer_id:0/1/2/3)。

需要注意的是 channel 0,1 通道下对应 layer_id 为 0~3 时，索引到的图层是支持 YUV 格式图像数据的；在 channel 的 2~3 通道下对应 layer_id 为 0~3 时，索引到的图层不支持 YUV 格式，而是 RGB 格式，示意图如下：



正常情况下，使用 0 路显示（本系统默认使用第 0 路显示）就可以满足用户需求。用户可以选择 disp：0 的某个通道对应的图层来播放视频，或者选择另外一个图层显示 UI，不同图层之间可以设置优先级、alpha 等参数，进行叠加显示。

对于内核来说，16 个 Layer 可以看作从 0~15 线性排布的（由 channel*4+layer_id 计算得到，默认 disp 为 0），其中第 0~7 个是视频图层，第 8~15 个是 UI 图层，视频图层的 0~3、4~7 图层的属性需要分别保持一致。

- 设置图层参数并使能：接口为 AW_MPI_VO_SetVideoLayerAttr 、AW_MPI_VO_EnableVideoLayer。
- 释放图层：接口 AW_MPI_VO_DisableVideoLayer，参数为需要释放的图层号。
- 打开/关闭图层：接口为 AW_MPI_VO_OpenVideoLayer/AW_MPI_VO_CloseVideoLayer。

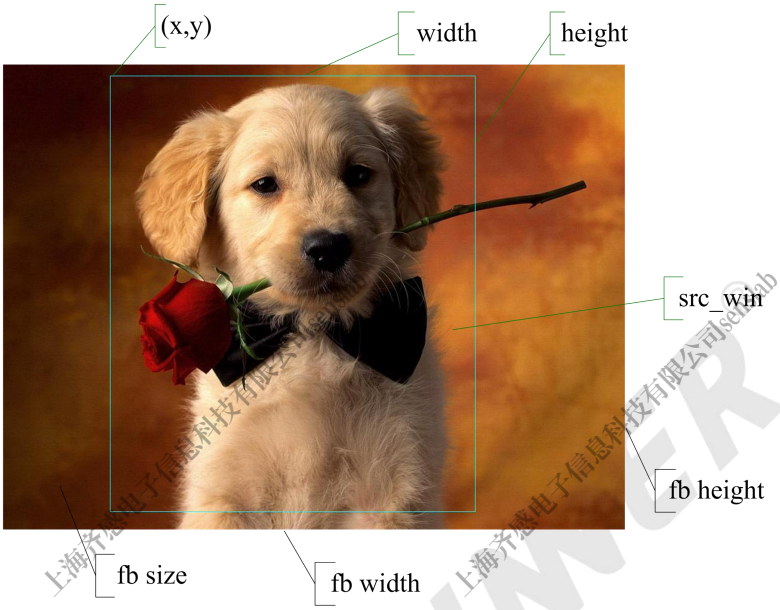
3.2.2. 显示输出设备操作说明

VO 支持多种显示输出设备，比如 LCD、TV、HDMI。开启显示输出设备有几种方式，第一种是在 sys_config.fex 中配置[disp]的初始化参数，显示模块在加载时将会根据配置来初始化选择的显示输出设备；第二种是在 kernel 启动后，调用 VO 模块的 API 接口去开启或关闭指定的输出设备，以下是操作的说明：

- 切换到某个具体的显示输出设备：接口是 AW_MPI_VO_SetPubAttr，参数是一个 VO_PUB_ATTR_S 类型的结构体，其中第二个参数 enIntfType 参数用来指定显示设备。

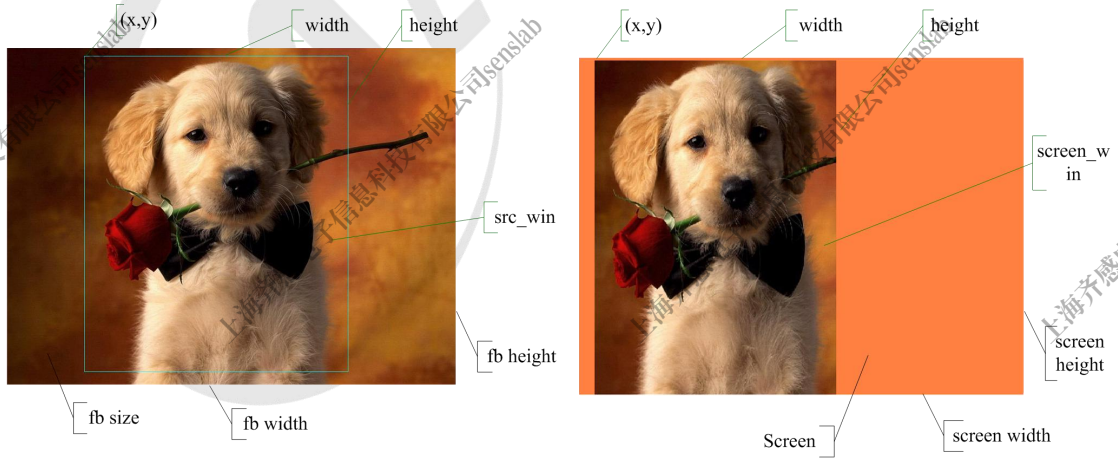
3.2.3. 图层 size 与 crop

图层 Frame Buffer 有两个与 size 有关的参数，分别是 size 与 crop。Size 表示 buffer 的完整尺寸，crop 则表示 buffer 中需要显示裁减区。如下图所示，完整的图像以 size 标识，而矩形框住的部分为裁减区，以 crop 标识，在屏幕上只能看到 crop 标识的部分，其余部分是隐藏的，不能在屏幕上显示出来。



3.2.4. 图层 crop 和 screen_win

Screen_win 为 crop 部分 buffer 在屏幕上显示的位置。如果不需要进行缩放的话，crop 和 screen_win 的 width,height 是相等的，如果需要缩放，需要用 scaler_mode 的图层来显示，crop 和 screen_win 的 width,height 可以不等。

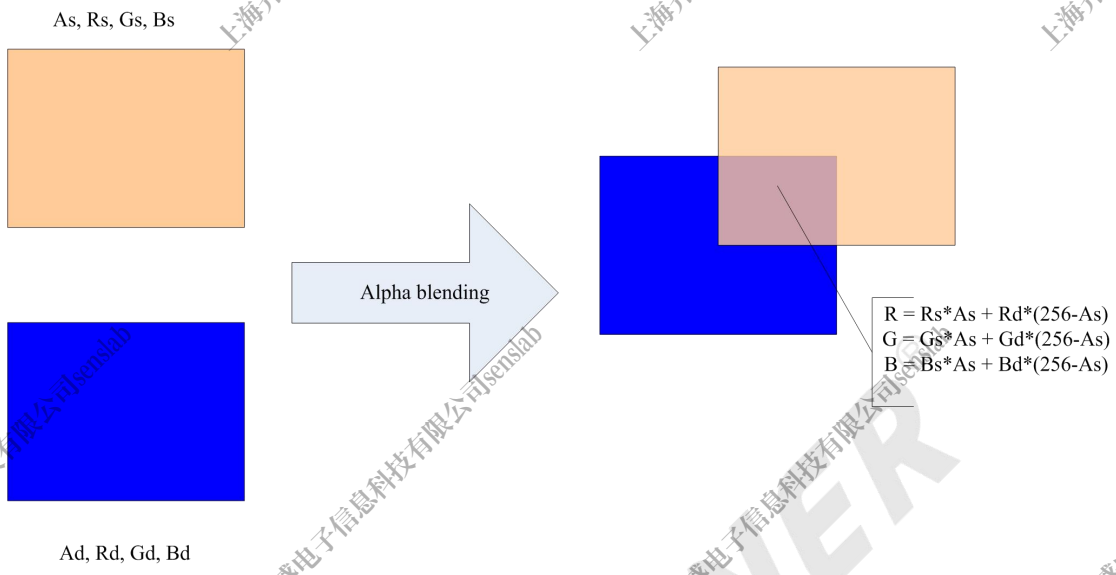


3.2.5. alpha

Alpha 模式有三种:

- Gloabal alpha：全局 alpha,也叫面 alpha，即整个图层共用一个 alpha，统一的透明度
- Pixel alpha：点 alpha，即每个像素都有自己单独的 alpha，可以实现部分区域全透，部分区域半透，部分区域不透的效果
- Global_pixel alpha：可以是说以上两种效果的叠加，在实现 pixel alpha 的效果的同时，

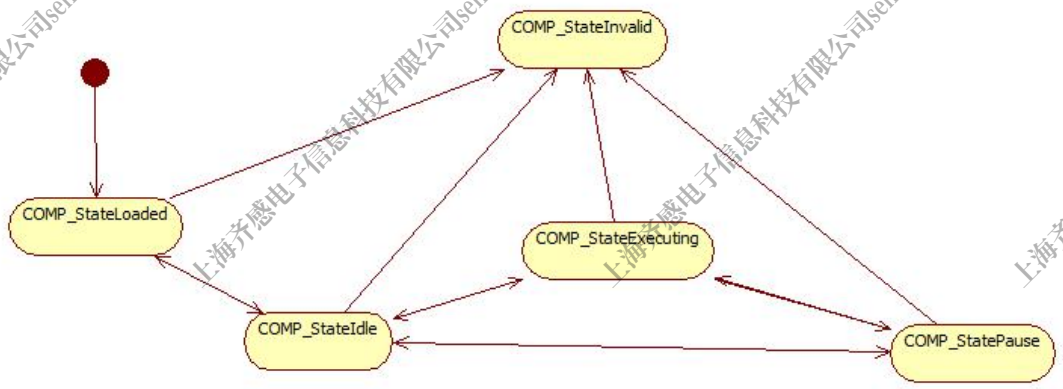
还可以做到淡入淡出的效果。



3.3. 输出设备介绍

- VO 支持屏、HDMI 以及 cvbs 等输出
- 屏的接口有很多类型，该平台支持 RGB/CPU/LVDS/DSI 接口。

3.4. 模块状态转换



各种状态的定义如下所示：

- COMP_StateIdle ：有资源，不传输数据。
- COMP_StateExecuting ：有资源，传输数据，处理数据。
- COMP_StatePause ：有资源，传输数据，不处理数据。
- COMP_StateLoaded ：无资源。
- COMP_StateInvalid ：非法状态。

3.5. API 接口

AW_MPI_VO_Enable

【目的】

使能指定的 VO 设备，构造 VODevInfo 结构体并加入全局链表。

【语法】

ERRORTYPE AW_MPI_VO_Enable(VO_DEV VoDev);

【参数】

参数	描述	输入输出
VoDev	需要使能的 VO 设备编号	输入

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_comm_vo.h 中的错误码描述。

【需求】

- 头文件：mpi_vo.h
- 库文件：libmpp_vo.so

【注意】

- 在进行 VO 相关操作之前应该首先调用此接口，保证相应的 VO 设备被使能。

【举例】

无。

AW_MPI_VO_Disable

【目的】

禁用指定的 VO 设备，同时释放相关的资源。

【语法】

ERRORTYPE AW_MPI_VO_Disable(VO_DEV VoDev);

【参数】

参数	描述	输入输出
VoDev	需要禁用的 VO 设备编号	输入

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_comm_vo.h 中的错误码描述。

【需求】

- 头文件：mpi_vo.h
- 库文件：libmpp_vo.so

【注意】

- 该操作会将指定 VO 设备结构体从全局链表中移除，该函数需要与 AW_MPI_VO_Enable 配对使用，在整个 VO 模块使用完毕之后调用该函数进行相关资源的释放。

【举例】

无。

AW_MPI_VO_SetPubAttr

【目的】

设置 VO 显示设备的背景色，指定 VO 设备的类型（HDMI、LCD 等），设置显示尺寸（720p、1080p 等）、频率。

【语法】

ERRORTYPE AW_MPI_VO_SetPubAttr(VO_DEV VoDev, const VO_PUB_ATTR_S *pstPubAttr);

【参数】

参数	描述	输入输出
VoDev	需要设置的 VO 设备编号	输入

pstPubAttr	VO 设备属性结构体	输入（动态）

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_comm_vo.h 中的错误码描述。

【需求】

- 头文件：mpi_vo.h
- 库文件：libmpp_vo.so

【注意】

- 需要初始化 pstPubAttr 的 enIntfType（显示设备）与 enIntfSync（分辨率以及刷新频率）成员。

enIntfType：VO_INTF_LCD、VO_INTF_TV、VO_INTF_HDMI、VO_INTF_VGA。

enIntfSync：

VO_OUTPUT_PAL、VO_OUTPUT_NTSC

VO_OUTPUT_720P50、VO_OUTPUT_720P_60

VO_OUTPUT_1080P24 、 VO_OUTPUT_1080P_25 、 VO_OUTPUT_1080P_30 、 VO_OUTPUT_1080P_50、VO_OUTPUT_1080P_60 等。

【举例】

```
/* 显示设备切换为 HDMI 设备，输出格式是 1080P/60fps */
VO_PUB_ATTR_S stPubAttr;
AW_MPI_VO_GetPubAttr(0, &stPubAttr);
stPubAttr.enIntfType = VO_INTF_HDMI;
stPubAttr.enIntfSync = VO_OUTPUT_1080P60;
AW_MPI_VO_SetPubAttr(0, &stPubAttr);
```

AW_MPI_VO_GetPubAttr

【目的】

获取 VO 设备的背景色、显示设备类型、显示数据格式等信息。

【语法】

ERRORTYPE AW_MPI_VO_GetPubAttr(VO_DEV VoDev, const VO_PUB_ATTR_S *pstPubAttr);

【参数】

参数	描述	输入输出
VoDev	需要获取信息的 VO 设备编号	输入
pstPubAttr	VO 设备属性结构体	输出（动态）

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_comm_vo.h 中的错误码描述。

【需求】

- 头文件：mpi_vo.h
- 库文件：libmpp_vo.so

【注意】

无。

【举例】

参见上面一条。

AW_MPI_VO_GetHdmiHwMode

【目的】

获取接入的 Hdmi 设备显示分辨率信息

【语法】

ERRORTYPE AW_MPI_VO_GetHdmiHwMode(VO_DEV VoDev, VO_INTF_SYNC_E *mode);

【参数】

返回值	描述	输入输出
VoDev	VO 设备	输入
mode	设备信息	输出（动态属性）

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_comm_vo.h 中的错误码描述。

【需求】

- 头文件：mpi_vo.h
- 库文件：libmpp_vo.so

【注意】

该函数在没有HDMI设备接入的时候会返回ERR_VO_DEV_NOT_CONFIG错误码，如果Hdmi设备的显示分辨率不被支持则会返回ERR_VO_NOT_SUPPORT错误码。

【举例】

```
/* declaration */
int ret = 0;
VO_INTF_SYNC_E mode;
/* enable one layer, chn = 0, lyl = 1 */
ret = AW_MPI_VO_GetHdmiHwMode(0, &mode);
If (SUCCESS != ret) {
    return -1;
}
```

AW_MPI_VO_EnableVideoLayer

【目的】

申请并使能一个图层，将该图层信息结构体加入到全局链表里面。

【语法】

```
ERRORTYPE AW_MPI_VO_EnableVideoLayer(VO_LAYER VoLayer);
```

【参数】

返回值	描述	输入输出
VoLayer	图层索引	输入

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_comm_vo.h 中的错误码描述。

【需求】

- 头文件：mpi_vo.h
- 库文件：libmpp_vo.so

【注意】

- VoLayer 要依靠一个宏定义来得到，该宏定义为 #define HLAY(chn, lyl) (chn*4+lyl)，其中 chn 是 VO 设备通道号（channel），lyl 是 layer 的编号（layer_id）。chn(0-3)，lyl(0-3)。

【举例】

```
/* declaration */
int ret = 0;
/* enable one layer, chn = 0, lyl = 1 */
ret = AW_MPI_VO_EnableVideoLayer(HALY(0, 1));
If (SUCCESS != ret) {
    return -1;
}
```

AW_MPI_VO_DisableVideoLayer

【目的】

释放指定的图层，从全局链表里面移除该图层信息结构体。

【语法】

```
ERRORTYPE AW_MPI_VO_DisableVideoLayer(VO_LAYER VoLayer);
```

【参数】

返回值	描述	输入输出
VoLayer	图层索引	输入

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_comm_vo.h 中的错误码描述。

【需求】

- 头文件：mpi_vo.h
- 库文件：libmpp_vo.so

【注意】

- 参见上一条。

【举例】

```
/* declaration */
int ret = 0;
/* enable one layer, chn = 0, lyl = 1 */
ret = AW_MPI_VO_DisableVideoLayer(HALY(0, 1));
If (SUCCESS != ret) {
    return -1;
}
```

AW_MPI_VO_AddOutsideVideoLayer

【目的】

添加一个外部图层（专门用于GUI）。

【语法】

```
ERRORTYPE AW_MPI_VO_AddOutsideVideoLayer(VO_LAYER VoLayer);
```

【参数】

返回值	描述	输入输出
VoLayer	图层索引	输入

【返回值】

返回值	描述
SUCCESS	成功

错误码	参考 mm_comm_vo.h 中的错误码描述。

【需求】

- 头文件：mpi_vo.h
- 库文件：libmpp_vo.so

【注意】

- 该函数专门用于添加 GUI 的界面图层（因为 GUI 图层的申请不经过 MPP_VO 这个模块，所以如果用 AW_MPI_VO_EnableVideoLayer 的话可能会出现因图层重复被申请而导致申请失败的情况，AW_MPI_VO_AddOutsideVideoLayer 函数没有图层申请这一步骤），特别注意：如果不是用于 GUI 图层的话请使用函数 AW_MPI_VO_EnableVideoLayer 来完成图层的新建。VoLayer 需要 HLAY 宏定义来配合生成。

【举例】

```
/* declaration */
int ret = 0;
/* add one UI layer, chn = 1, lyl = 0 */
ret = AW_MPI_VO_AddOutsideVideoLayer(HALY(1, 0));
If (SUCCESS != ret) {
    return -1;
}
```

AW_MPI_VO_RemoveOutsideVideoLayer

【目的】

移除外部图层（专门用于 GUI）。

【语法】

```
ERRORTYPE AW_MPI_VO_RemoveOutsideVideoLayer(VO_LAYER VoLayer);
```

【参数】

返回值	描述	输入输出
VoLayer	图层索引	输入

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_comm_vo.h 中的错误码描述。

【需求】

- 头文件：mpi_vo.h
- 库文件：libmpp_vo.so

【注意】

- 该函数专门用于移除GUI的界面图层（因为GUI图层的释放不经过MPP_VO这个模块，如果用AW_MPI_VO_DisableVideoLayer的话可能会出现别的程序正在使用该图层，而该图层的资源被释放的情况），特别注意：如果不是用于GUI图层的话请尽量使用函数AW_MPI_VO_DisableVideoLayer来完成图层的移除。VoLayer需要HLAY宏定义来配合生成。

【举例】

无。

AW_MPI_VO_OpenVideoLayer

【目的】

打开一个指定的图层。

【语法】

ERRORTYPE AW_MPI_VO_OpenVideoLayer(VO_LAYER VoLayer);

【参数】

返回值	描述	输入输出
VoLayer	图层索引	输入

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_comm_vo.h 中的错误码描述。

【需求】

- 头文件：mpi_vo.h

- 库文件：libmpp_vo.so

【注意】

- 在调用完 AW_MPI_VO_EnableVideoLayer 函数之后图层默认是被关闭的，如果需要在屏幕上显示出该图层，需要先调用该函数打开图层。

【举例】

```
/* declaration */
int ret = 0;
/* open one layer, chn = 1, lyl = 0 */
ret = AW_MPI_VO_OpenVideoLayer(HALY(1, 0));
If (SUCCESS != ret) {
    return -1;
}
```

AW_MPI_VO_CloseVideoLayer

【目的】

关闭一个指定的图层。

【语法】

ERRORTYPE AW_MPI_VO_CloseVideoLayer(VO_LAYER VoLayer);

【参数】

返回值	描述	输入输出
VoLayer	图层索引	输入

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_comm_vo.h 中的错误码描述。

【需求】

- 头文件：mpi_vo.h
- 库文件：libmpp_vo.so

【注意】

- 如果先调用函数 AW_MPI_VO_DisableVideoLayer，再调用该函数可能会造成花屏现象，花屏持续的时间取决于两个函数的调用间隔，间隔时间越长，花屏时间就越长。最好还是先调用该函数关闭图层，然后再 Disable。

【举例】

无。

AW_MPI_VO_SetVideoLayerAttr

【目的】

设置指定图层的属性。

【语法】

ERRORTYPE AW_MPI_VO_SetVideoLayerAttr(VO_LAYER VoLayer, const VO_VIDEO_LAYER_ATTR_S *pstLayerAttr);

【参数】

返回值	描述	输入输出
VoLayer	图层索引	输入
pstLayerAttr	图层属性结构体	输入（动态）

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_comm_vo.h 中的错误码描述。

【需求】

- 头文件：mpi_vo.h
- 库文件：libmpp_vo.so

【注意】

- pstLayerAttr 的结构体类型是 VO_VIDEO_LAYER_ATTR_S，里面包含显示区域、图片大小、图层像素格式等元素，但是本函数只对 VO_VIDEO_LAYER_ATTR_S 的

stDispRect 成员生效，也就是说本函数只是用于设置图像显示的 X、Y 坐标以及宽和高。

【举例】

```
/* declaration */
int ret = 0;
VO_VIDEO_LAYER_ATTR_S stLayerAttr;
/* set layer attribute, chn = 1, lyl = 0 */
stLayerAttr.stDispRect.X = 0;
stLayerAttr.stDispRect.Y = 0;
stLayerAttr.stDispRect.Width = 1280;
stLayerAttr.stDispRect.Height = 720;
ret = AW_MPI_VO_SetVideoLayerAttr(HALY(1, 0), &stLayerAttr);
If (SUCCESS != ret) {
    return -1;
}
```

AW_MPI_VO_GetVideoLayerAttr

【目的】

获取指定图层的属性。

【语法】

ERRORTYPE AW_MPI_VO_GetVideoLayerAttr(VO_LAYER VoLayer, const VO_VIDEO_LAYER_ATTR_S *pstLayerAttr);

【参数】

返回值	描述	输入输出
VoLayer	图层索引	输入
pstLayerAttr	图层属性结构体	输出（动态）

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_comm_vo.h 中的错误码描述。

【需求】

- 头文件：mpi_vo.h
- 库文件：libmpp_vo.so

【注意】

无。

【举例】

```
/* declaration */
int ret = 0;
VO_VIDEO_LAYER_ATTR_S stLayerAttr;
/* set layer attribute, chn = 1, lyl = 0 */
ret = AW_MPI_VO_GetVideoLayerAttr(HALY(1, 0), &stLayerAttr);
If (SUCCESS != ret) {
    return -1;
}
```

AW_MPI_VO_SetVideoLayerPriority

【目的】

设置指定图层的显示优先级。

【语法】

```
ERRORTYPE AW_MPI_VO_SetVideoLayerPriority(VO_LAYER VoLayer, unsigned int uPriority);
```

【参数】

返回值	描述	输入输出
VoLayer	图层索引	输入
uPriority	优先级	输入（动态）

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_comm_vo.h 中的错误码描述。

【需求】

- 头文件：mpi_vo.h
- 库文件：libmpp_vo.so

【注意】

- 显示的时候越靠近上层的图层其 uPriority 值越大，所以优先级越高，相应的图层就越靠近顶层。uPriority 的取值范围是 0~15，使用的时候注意不要超过此范围。

【举例】

```
/* declaration */
int ret = 0;
unsigned int uPriority = 10;
/* set layer priority, chn = 1, lyl = 0 */
ret = AW_MPI_VO_SetVideoLayerAttr(HALY(1, 0), uPriority);
If (SUCCESS != ret) {
    return -1;
}
```

AW_MPI_VO_GetVideoLayerPriority

【目的】

获取指定图层的显示优先级。

【语法】

ERRORTYPE AW_MPI_VO_GetVideoLayerPriority(VO_LAYER VoLayer, unsigned int *puPriority);

【参数】

返回值	描述	输入输出
VoLayer	图层索引	输入
uPriority	优先级	输出（动态）

【返回值】

返回值	描述
SUCCESS	成功

错误码	参考 mm_comm_vo.h 中的错误码描述。

【需求】

- 头文件：mpi_vo.h
- 库文件：libmpp_vo.so

【注意】

无。

【举例】

```
/* declaration */
int ret = 0;
unsigned int uPriority;
/* get layer priority, chn = 1, lyl = 0 */
ret = AW_MPI_VO_GetVideoLayerAttr(HALY(1, 0), &uPriority);
If (SUCCESS != ret) {
    return -1;
}
```

AW_MPI_VO_SetVideoLayerAlpha

【目的】

设置指定图层的 alpha，可以理解为透明度。

【语法】

ERRORTYPE AW_MPI_VO_SetVideoLayerAlpha(VO_LAYER, VO_VIDEO_LAYER_ALPHA_S *pAlpha);

【参数】

返回值	描述	输入输出
VoLayer	图层索引	输入
pAlpha	alpha 信息结构体	输入（动态）

【返回值】

返回值	描述

SUCCESS	成功
错误码	参考 mm_comm_vo.h 中的错误码描述。

【需求】

- 头文件：mpi_vo.h
- 库文件：libmpp_vo.so

【注意】

- pAlpha 结构体有结构体成员 mAlphaMode，为 0 时是点 alpha 模式，此时每个像素都有自己单独的 alpha，可以实现部分区域全透，部分区域半透，部分区域不透的效果；为 1 时是面 alpha 模式，整个图层共用一个 alpha，统一的透明度。需要注意的是 YUV 格式下并不支持使用 alpha，所以该格式下函数调用会失败，只有 ARGB 等带有 alpha 模式的的图像才可以使用。

【举例】

```
/* declaration */
int ret = 0;
VO_VIDEO_LAYER_ALPHA_S stAlpha;
stAlpha.mAlphaMode = 1;
stAlpha.mAlphaValue = 128;
/* set layer's alpha, chn = 1, lyl = 0 */
ret = AW_MPI_VO_SetVideoLayerAlpha(HALY(1, 0), &stAlpha);
If (SUCCESS != ret) {
    return -1;
}
```

AW_MPI_VO_GetVideoLayerAlpha

【目的】

获取指定图层的 alpha，包括 alpha 模式以及其 value

【语法】

ERRORTYPE AW_MPI_VO_GetVideoLayerAlpha(VO_LAYER, VO_VIDEO_LAYER_ALPHA_S *pAlpha);

【参数】

返回值	描述	输入输出
VoLayer	图层索引	输入
pAlpha	alpha 信息结构体	输出（动态）

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_comm_vo.h 中的错误码描述。

【需求】

- 头文件：mpi_vo.h
- 库文件：libmpp_vo.so

【注意】

无。

【举例】

```
/* declaration */
int ret = 0;
VO_VIDEO_LAYER_ALPHA_S stAlpha;
/* set layer's alpha, chn = 1, lyl = 0 */
ret = AW_MPI_VO_GetVideoLayerAlpha(HALY(1, 0), &stAlpha);
If (SUCCESS != ret) {
    return -1;
}
```

AW_MPI_VO_EnableChn

【目的】

为指定的 VO 图层创建一个指定编号的 VO 组件通道。

【语法】

```
ERRORTYPE AW_MPI_VO_EnableChn(VO_LAYER VoLayer, VO_CHN VoChn);
```

【参数】

返回值	描述	输入输出
VoLayer	图层索引	输入
VoChn	通道号	输入

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_comm_vo.h 中的错误码描述。

【需求】

- 头文件：mpi_vo.h
- 库文件：libmpp_vo.so

【注意】

【举例】

```
/* declaration */
int ret = 0;
/* create channel */
ret = AW_MPI_VO_EnableChn(HLAY(0, 1), 0);
if (SUCCESS == ret) {
    printf("create vo channel succeed!\n");
} else if (ERR_VO_CHN_NOT_DISABLE == ret) {
    printf("this vo channel has exist, find another one. \n");
} else {
    return -1;
}
```

AW_MPI_VO_DisableChn

【目的】

销毁指定 VO 图层指定编号的 VO 通道。

【语法】

ERRORTYPE AW_MPI_VO_DisableChn(VO_LAYER VoLayer, VO_CHN VoChn);

【参数】

返回值	描述	输入输出
VoLayer	图层索引	输入
VoChn	通道号	输入

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_comm_vo.h 中的错误码描述。

【需求】

- 头文件：mpi_vo.h
- 库文件：libmpp_vo.so

【注意】

- 该函数会销毁 AW_MPI_VO_EnableChn 的所有资源。

【举例】

无。

AW_MPI_VO_RegisterCallback

【目的】

为创建的 VO 通道组件实例注册一个回调函数。

【语法】

ERRORTYPE AW_MPI_VO_RegisterCallback(VO_LAYER VoLayer, VO_CHN VoChn, MPPCallbackInfo *pCallback);

【参数】

返回值	描述	输入输出
VoLayer	图层索引	输入

VoChn	VO 通道号	输入
pCallback	回调函数信息结构体	输入（静态）

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_comm_vo.h 中的错误码描述。

【需求】

- 头文件：mpi_vo.h
- 库文件：libmpp_vo.so

【注意】

- 该函数用于 MPP_VO 模块向应用程序发送事件通知，如果应用程序不需要接收来自 MPP_VO 模块的事件，那么也可以不注册该函数（但不建议这么做）。pCallback 的 cookie 成员存储应用程序自定义的结构体数据，callback 成员则需要指向应用程序的回调函数实体。回调函数的参数为 Func(void *cookie, MPP_CHN_S *pChn, MPP_EVENT_TYPE event, void *pEventData)。
- MPP_VO 模块定义的事件有以下类型：

MPP_EVENT_RELEASE_VIDEO_BUFFER, //VIDEO_FRAME_INFO_S for recorder/VIChannel::DoVdaThread, ISE, VO, VENC

MPP_EVENT_VENC_TIMEOUT, //uint64_t*

MPP_EVENT_RELEASE_ISE_VIDEO_BUFFER0, //VIDEO_FRAME_INFO_S for recorder/VIChannel::DoVdaThread, ISE, VO, VENC

MPP_EVENT_RELEASE_ISE_VIDEO_BUFFER1, //VIDEO_FRAME_INFO_S for recorder/VIChannel::DoVdaThread, ISE, VO, VENC

MPP_EVENT_RELEASE_AUDIO_BUFFER, //AUDIO_FRAME_S

MPP_EVENT_BSFRAME_AVAILABLE, //CDXRecorderBsInfo

MPP_EVENT_ERROR_ENCBUFFER_OVERFLOW,

MPP_EVENT_NEED_NEXT_FD, // int muxerId

MPP_EVENT_RECORD_DONE, // int muxerId

MPP_EVENT_CAPTURE_AUDIO_DATA, // unsigned int size;

MPP_EVENT_NOTIFY_EOF = 0x100,

MPP_EVENT_SET_VIDEO_SIZE, //SIZE_S

```
    MPP_EVENT_RENDERING_START,
```

【举例】

```
/* declaration */
int ret = 0;
MPPCallbackInfo cbInfo;
struct custom_st_name stCustom;
cbInfo.cookie   = (void *)&stCustom;
cbInfo.callback = (MPPCallbackFuncType)&CallbackFuncName;
/* register it */
ret = AW_MPI_VO_RegisterCallback(HLAY(0, 1), 0, &cbInfo);
if (SUCCESS == ret) {
    return -1;
}
```

AW_MPI_VO_SetChnDispBufNum

【目的】

设置 VO 组件通道实例的缓存 buf 数量。

【语法】

ERRORTYPE AW_MPI_VO_SetChnDispBufNum(VO_LAYER VoLayer, VO_CHN VoChn, unsigned int uBufNum);

【参数】

返回值	描述	输入输出
VoLayer	图层索引	输入
VoChn	VO 通道号	输入
uBufNum	缓存 buf 的数量	输入（动态）

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_comm_vo.h 中的错误码描述。

【需求】

- 头文件：mpi_vo.h
- 库文件：libmpp_vo.so

【注意】

- 该函数必须被调用，并且 uBufNum 的数量要大于等于 1，同时 uBufNum 的数量不能超过 16。

【举例】

无。

AW_MPI_VO_GetChnDispBufNum

【目的】

获取 VO 组件通道实例的缓存 buf 数量。

【语法】

ERRORTYPE AW_MPI_VO_GetChnDispBufNum(VO_LAYER VoLayer, VO_CHN VoChn, unsigned int uBufNum);

【参数】

返回值	描述	输入输出
VoLayer	图层索引	输入
VoChn	VO 通道号	输入
uBufNum	缓存 buf 的数量	输出（动态）

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_comm_vo.h 中的错误码描述。

【需求】

- 头文件：mpi_vo.h
- 库文件：libmpp_vo.so

【注意】

无。

【举例】

无。

AW_MPI_VO_GetDisplaySize

【目的】

获取 VO 组件通道实例图层的显示数据（图层的显示宽、高）。

【语法】

ERRORTYPE AW_MPI_VO_GetDisplaySize(VO_LAYER VoLayer, VO_CHN VoChn, SIZE_S *pDisplaySize);

【参数】

返回值	描述	输入输出
VoLayer	图层索引	输入
VoChn	VO 通道号	输入
pDisplaySize	size 信息结构体	输出（动态）

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_comm_vo.h 中的错误码描述。

【需求】

- 头文件：mpi_vo.h
- 库文件：libmpp_vo.so

【注意】

- pDisplaySize 结构体成员的 Width 代表宽度，Height 代表高度。

【举例】

无。

AW_MPI_VO_StartChn

【目的】

通道开始工作（VO 组件状态设为 StateExecuting），接收视频数据并送去显示。

【语法】

ERRORTYPE AW_MPI_VO_StartChn(VO_LAYER VoLayer, VO_CHN VoChn);

【参数】

返回值	描述	输入输出
VoLayer	图层索引	输入
VoChn	VO 通道号	输入

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_comm_vo.h 中的错误码描述。

【需求】

- 头文件：mpi_vo.h
- 库文件：libmpp_vo.so

【注意】

- 要在通道使能后，注册完回调函数之后再调用该函数。

【举例】

无。

AW_MPI_VO_StopChn

【目的】

通道停止（VO 组件状态设为 StateIdle），停止数据传输。

【语法】

ERRORTYPE AW_MPI_VO_StopChn(VO_LAYER VoLayer, VO_CHN VoChn);

【参数】

返回值	描述	输入输出
VoLayer	图层索引	输入
VoChn	VO 通道号	输入

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_comm_vo.h 中的错误码描述。

【需求】

- 头文件：mpi_vo.h
- 库文件：libmpp_vo.so

【注意】

- 停止数据传输之后并不会释放相关的资源，可以再次从停止状态恢复。

【举例】

无。

AW_MPI_VO_PauseChn

【目的】

通道暂停（设置组件状态为 StatePause），停止显示。

【语法】

ERRORTYPE AW_MPI_VO_PauseChn(VO_LAYER VoLayer, VO_CHN VoChn);

【参数】

返回值	描述	输入输出
VoLayer	图层索引	输入
VoChn	VO 通道号	输入

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_comm_vo.h 中的错误码描述。

【需求】

- 头文件：mpi_vo.h
- 库文件：libmpp_vo.so

【注意】

无。

【举例】

无。

AW_MPI_VO_ResumeChn

【目的】

通道恢复（设置 VO 组件状态为 StateExecuting），开始数据传输。

【语法】

ERRORTYPE AW_MPI_VO_ResumeChn(VO_LAYER VoLayer, VO_CHN VoChn);

【参数】

返回值	描述	输入输出
VoLayer	图层索引	输入
VoChn	VO 通道号	输入

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_comm_vo.h 中的错误码描述。

【需求】

- 头文件：mpi_vo.h

- 库文件：libmpp_vo.so

【注意】

- 根据 openMAX 标准的说法，各种状态的定义如下所示：

StateIdle：有资源，不传输数据。

StateExecuting：有资源，传输数据，处理数据。

StatePause：有资源，传输数据，不处理数据。

三个状态两两可以相互转换。

【举例】

无。

AW_MPI_VO_Seek

【目的】

跳转播放。

【语法】

ERRORTYPE AW_MPI_VO_Seek(VO_LAYER VoLayer, VO_CHN VoChn);

【参数】

返回值	描述	输入输出
VoLayer	图层索引	输入
VoChn	VO 通道号	输入

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_comm_vo.h 中的错误码描述。

【需求】

头文件：mpi_vo.h

库文件：libmpp_vo.so

【注意】

- MPP_VO 模块的跳转播放不是真正的“跳转”，真正的跳转在视频格式解析模块里面（可以跳转到指定的时间播放视频），该模块的跳转函数只是负责在设置跳转之后将跳转之前缓存到的视频帧丢弃，否则会出现跳转之后播放到跳转前视频帧的情况。最好在跳播之后调用该函数。举例：现在播放到视频的第 100ms，此时 MPP_VO 模块可能缓存了 100ms 之后的几帧视频，然后同时发生了跳转事件，视频直接从 100ms 跳转到了 2000ms 处播放，那么需要调用该函数清除 100ms 之后的几帧缓存视频数据，从而保证跳转播放的连贯性。

【举例】

无。

AW_MPI_VO_SetStreamEof

【目的】

根据参数标记视频流的结束（此时会停止视频的显示），状态转为 StateIdle；或者取消视频流结束标记。

【语法】

ERRORTYPE AW_MPI_VO_SetStreamEof(VO_LAYER VoLayer, VO_CHN VoChn, BOOL bEofFlag);

【参数】

返回值	描述	输入输出
VoLayer	图层索引	输入
VoChn	VO 通道号	输入
bEofFlag	视频流结束标志	输入

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_comm_vo.h 中的错误码描述。

【需求】

- 头文件：mpi_vo.h
- 库文件：libmpp_vo.so

【注意】

- bEofFlag 为 1 时表示视频流结束，为 0 时取消视频结束标志。当设置为 1 时 VO 组件的状态会转为 StateIdle，此时视频流会停止传输。

【举例】

无。

AW_MPI_VO_ShowChn

【目的】

通道图层显示（去隐藏化）。

【语法】

ERRORTYPE AW_MPI_VO_ShowChn(VO_LAYER VoLayer, VO_CHN VoChn);

【参数】

返回值	描述	输入输出
VoLayer	图层索引	输入
VoChn	VO 通道号	输入

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_comm_vo.h 中的错误码描述。

【需求】

- 头文件：mpi_vo.h
- 库文件：libmpp_vo.so

【注意】

- 如果通道本来就是处于 show 状态，则不会有任何的变化，如果通道本来处于 hide 状态，该函数会使得指定通道对应的图层变得可见。

【举例】

无。

AW_MPI_VO_HideChn

【目的】

通道图层隐藏（不可见）。

【语法】

ERRORTYPE AW_MPI_VO_HideChn(VO_LAYER VoLayer, VO_CHN VoChn);

【参数】

返回值	描述	输入输出
VoLayer	图层索引	输入
VoChn	VO 通道号	输入

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_comm_vo.h 中的错误码描述。

【需求】

- 头文件：mpi_vo.h
- 库文件：libmpp_vo.so

【注意】

- 通道隐藏只是在显示屏幕上面变得不可见，并不会销毁通道相关的任何资源，调用 AW_MPI_VO_ShowChn 之后通道对应的图层在显示屏幕上面会重新变为可见状态。

【举例】

无。

AW_MPI_VO_GetChnPts

【目的】

获取 VO 组件通道实例视频播放的时间戳，单位 us

【语法】

ERRORTYPE AW_MPI_VO_GetChnPts(VO_LAYER VoLayer, VO_CHN VoChn, uint64_t *pChnPts);

【参数】

返回值	描述	输入输出
VoLayer	图层索引	输入
VoChn	VO 通道号	输入
pChnPts	时间戳（指针类型）	输出（动态）

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_comm_vo.h 中的错误码描述。

【需求】

- 头文件：mpi_vo.h
- 库文件：libmpp_vo.so

【注意】

无。

【举例】

无。

AW_MPI_VO_SendFrame

【目的】

发送一帧视频数据送去显示。

【语法】

ERRORTYPE AW_MPI_VO_SendFrame(VO_LAYER VoLayer, VO_CHN VoChn, VIDEO_FRAME_INDO_S *pstFrame, int nMilliSec);

【参数】

返回值	描述	输入输出
VoLayer	图层索引	输入
VoChn	VO 通道号	输入
pstFrame	视频帧信息结构体	输入（动态）
nMilliSec	等待时间，单位 ms	输入

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_comm_vo.h 中的错误码描述。

【需求】

- 头文件：mpi_vo.h
- 库文件：libmpp_vo.so

【注意】

- pstFrame 结构体里面包含视频帧 id，视频帧大小、像素格式、物理地址、虚拟地址、时间戳等等。该函数一般用在非绑定的 VO 组件中，此时视频数据是从外部文件中读取到的，必须设置视频帧的 id、宽度、高度、像素格式、物理地址、虚拟地址、时间戳。
- nMilliSec 参数代表等待时间，如果超过该时间，视频还没有被成功发送，那么该帧视频数据就会被丢弃。

【举例】

无。

AW_MPI_VO_Debug_StoreFrame

【目的】

保存一帧视频数据，可用于截图。

【语法】

ERRORTYPE AW_MPI_VO_Debug_StoreFrame(VO_LAYER VoLayer, VO_CHN VoChn, unit64_t framePts);

【参数】

返回值	描述	输入输出
VoLayer	图层索引	输入
VoChn	VO 通道号	输入
framePts	时间戳	输入

【返回值】

返回值	描述
SUCCESS	成功
错误码	参考 mm_comm_vo.h 中的错误码描述。

【需求】

- 头文件：mpi_vo.h
- 库文件：libmpp_vo.so

【注意】

- 该函数可用于保存一帧视频数据到文件里面，framePts 参数指定了需要保存的视频数据所在的位置（以时间为坐标，单位 ms）。如果找不到指定时间的帧，那么程序就会选择最接近的帧进行保存。

【举例】

无。

3.6. 数据结构

VO_CHN_ATTR_S

●PROTOTYPE

```
typedef struct VO_CHN_ATTR_S
{
    unsigned int mPriority;                 /* video out overlay pri */
    RECT_S    mRect;                        /* rect of video out chn */
    BOOL mbDeflicker;                       /* deflicker or not */
}VO_CHN_ATTR_S;
```

●MEMBERS

mPriority：通道优先级

mRect ：通道视频显示的范围（X、Y 坐标，宽高）

mbDeflicker ：

●DESCRIPTION

VO_CHN_ATTR_S：用于设置通道的属性

VO_PUB_ATTR_S

●PROTOTYPE

```
typedef struct VO_PUB_ATTR_S
{
    unsigned int            mBgColor;       /* 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;
```

●MEMBERS

mBgColor：设备背景色，RGB 格式

enIntfType：VO 显示设备结口类型

enIntfSync：显示分辨率以及频率

stSyncInfo：VO 接口的时间信息

/* VO 显示设备接口类型（部分） */

VO_INTF_CVBS ：CVBS 接口

VO_INTF_YPBPR：YPBPR 接口

VO_INTF_VGA ：VGA 接口

VO_INTF_BT656 ：BT656 接口

VO_INTF_HDMI ：HDMI 接口

VO_INTF_LCD ：LCD 接口

/* 显示分辨率以及频率（部分） */

VO_OUTPUT_PAL

VO_OUTPUT_NTSC

VO_OUTPUT_1080P24,

```
    VO_OUTPUT_1080P25,
    VO_OUTPUT_1080P30,

    VO_OUTPUT_720P50,
    VO_OUTPUT_720P60,
    VO_OUTPUT_1080P50,
    VO_OUTPUT_1080P60,

    VO_OUTPUT_800x600_60,          /* VESA 800 x 600 at 60 Hz (non-interlaced) */
    VO_OUTPUT_1024x768_60,         /* VESA 1024 x 768 at 60 Hz (non-interlaced) */
    VO_OUTPUT_1280x1024_60,        /* VESA 1280 x 1024 at 60 Hz (non-interlaced) */
    VO_OUTPUT_1366x768_60,         /* VESA 1366 x 768 at 60 Hz (non-interlaced) */
    VO_OUTPUT_1440x900_60,         /* VESA 1440 x 900 at 60 Hz (non-interlaced) CVT
Compliant */
    VO_OUTPUT_1920x1200_60,        /* VESA 1920 x 1600 at 60 Hz (non-interlaced) CVT
(Reduced Blanking)*/
```

●DESCRIPTION

VO_PUB_ATTR_S：设置显示通道的设备接口类型、分辨率

VO_VIDEO_LAYER_ATTR_S

●PROTOTYPE

```
typedef struct VO_VIDEO_LAYER_ATTR_S
{
    RECT_S stDispRect;                  /* Display resolution */
    SIZE_S stImageSize;                 /* Canvas size of the video layer */
    unsigned int mDispFrmRt;            /* Display frame rate */
    PIXEL_FORMAT_E enPixFormat;         /* Pixel format of the video layer */
    BOOL bDoubleFrame;                  /* Whether to double frames */
    BOOL bClusterMode;                  /* Whether to take Cluster way to use memory*/
} VO_VIDEO_LAYER_ATTR_S;
```

●MEMBERS

stDispRect	：	图层的范围（X、Y 坐标，宽高）
stImageSize	：	图层画布大小
mDispFrmRt	：	显示频率

enPixFormat ：图层像素格式

bDoubleFrame：视频层倍帧开关。

bClusterMode ：视频层内存聚集使能开关。

●DESCRIPTION

VO_VIDEO_LAYER_ATTR_S：描述显示图层的属性

VO_VIDEO_LAYER_ALPHA_S

●PROTOTYPE

```
typedef struct VO_VIDEO_LAYER_ALPHA_S
{
    unsigned char mAlphaMode;    /* 0: Pixel Mode, 1: Global Mode */
    unsigned char mAlphaValue;
}VO_VIDEO_LAYER_ALPHA_S;
```

●MEMBERS

mAlphaMode：图层的 alpha 模式，0：点 alpha；1：面 alpha。参见 *alpha* 解释

mAlphaValue：图层 alpha 的值

●DESCRIPTION

VO_VIDEO_LAYER_ALPHA_S：描述图层的 alpha 属性

VIDEO_FRAME_INFO_S

●PROTOTYPE

```
typedef struct VIDEO_FRAME_INFO_S
{
    VIDEO_FRAME_S VFrame;
    unsigned int mId;    //id identify frame uniquely
} VIDEO_FRAME_INFO_S;
```

●MEMBERS

VFrame ：帧结构体描述

mId ：帧 ID

●DESCRIPTION

VIDEO_FRAME_INFO_S：用于描述一帧视频数据

VIDEO_FRAME_S

●PROTOTYPE

```
typedef struct VIDEO_FRAME_S
{
    unsigned int        mWidth;
    unsigned int        mHeight;
    VIDEO_FIELD_E       mField;
    PIXEL_FORMAT_E      mPixelFormat;

    VIDEO_FORMAT_E      mVideoFormat;
    COMPRESS_MODE_E     mCompressMode;

    unsigned int        mPhyAddr[3];// Y, U, V; Y, UV; Y, VU
    void*               mpVirAddr[3];
    unsigned int        mStride[3];

    unsigned int        mHeaderPhyAddr[3];
    void*               mpHeaderVirAddr[3];
    unsigned int        mHeaderStride[3];

    short               mOffsetTop;         /* top offset of show area */
    short               mOffsetBottom;      /* bottom offset of show area */
    short               mOffsetLeft;        /* left offset of show area */
    short               mOffsetRight;       /* right offset of show area */

    uint64_t            mpts; //unit:us
    unsigned int        mTimeRef;

    unsigned int        mPrivateData;
    VIDEO_SUPPLEMENT_S mSupplement;
    int mEnvLV; //environment luminance value
} VIDEO_FRAME_S;
```

●MEMBERS

mWidth ： 宽

mHeight ：高

mField ：视频所在的域

mPixelFormat ：像素格式

mVideoFormat ：视频格式

mCompressMode ：压缩模式

mPhyAddr ：物理地址，按照 Y、U、V 或者 Y、UV 或者 Y、VU 的顺序排列

mpVirAddr ：虚拟地址，对应物理地址

●DESCRIPTION

VIDEO_FRAME_S：描述视频帧的大小，位置等信息。

VIDEO_FIELD_E

●PROTOTYPE

```
typedef enum VIDEO_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;
```

●MEMBERS

VIDEO_FIELD_TOP ：偶数行

VIDEO_FIELD_BOTTOM ：奇数行

VIDEO_FIELD_INTERLACED ：隔行扫描

VIDEO_FIELD_FRAME ：帧

●DESCRIPTION

VIDEO_FIELD_E：描述视频帧在一副画面中所在的位置

VIDEO_FORMAT_E

●PROTOTYPE

```
typedef enum VIDEO_FORMAT_E
{
```

```
    VIDEO_FORMAT_LINEAR     = 0x0,      /* nature video line */
    VIDEO_FORMAT_TILE       = 0x1,      /* tile cell: 256pixel x 16line, default tile mode
*/
    VIDEO_FORMAT_TILE64     = 0x2,      /* tile cell: 64pixel x 16line */

    VIDEO_FORMAT_BUTT
} VIDEO_FORMAT_E;
```

●MEMBERS

VIDEO_FORMAT_LINEAR：正常的视频行

VIDEO_FORMAT_TILE ：单位：256 像素*16 行

VIDEO_FORMAT_TILE64 ：单位：64 像素*16 行

●DESCRIPTION

VIDEO_FORMAT_E：描述视频的格式。

COMPRESS_MODE_E

●PROTOTYPE

```
typedef enum COMPRESS_MODE_E
{
    COMPRESS_MODE_NONE      = 0x0,      /* no compress */
    COMPRESS_MODE_SEG       = 0x1,      /* compress unit is 256 bytes as a segment,
default seg mode */
    COMPRESS_MODE_SEG128    = 0x2,      /* compress unit is 128 bytes as a segment */
    COMPRESS_MODE_LINE      = 0x3,      /* compress unit is the whole line */
    COMPRESS_MODE_FRAME     = 0x4,      /* compress unit is the whole frame */

    COMPRESS_MODE_BUTT
} COMPRESS_MODE_E;
```

●MEMBERS

COMPRESS_MODE_NONE：不压缩

COMPRESS_MODE_SEG ：压缩单位：256 个字节为一段，默认的段压缩模式

COMPRESS_MODE_SEG128 ：压缩单位：128 个字节为一段

COMPRESS_MODE_LINE ：压缩单位：一整行压缩

COMPRESS_MODE_FRAME ：压缩单位：完整的一帧

●DESCRIPTION

COMPRESS_MODE_E：表明视频帧的压缩方法。

3.7. 错误码

错误码	宏定义	描述
0xA00F8012	ERR_VO_BUSY	VO 正忙
0xA00F800C	ERR_VO_NO_MEM	没有足够的内存
0xA00F8006	ERR_VO_NULL_PTR	空指针
0xA00F8010	ERR_VO_SYS_NOTREADY	VO 没有准备好
0xA00F8001	ERR_VO_INVALID_DEVID	无效的 VO 设备 ID
0xA00F8002	ERR_VO_INVALID_CHNID	无效的通道 ID
0xA00F8003	ERR_VO_ILLEGAL_PARAM	非法参数
0xA00F8008	ERR_VO_NOT_SUPPORT	不支持
0xA00F8009	ERR_VO_NOT_PERMIT	没有权限，不允许
0xA00F806C	ERR_VO_INVALID_WBCID	
0xA00F806D	ERR_VO_INVALID_LAYERID	非法的 Layer ID
0xA00F8040	ERR_VO_DEV_NOT_CONFIG	VO 设备没有被配置
0xA00F8041	ERR_VO_DEV_NOT_ENABLE	VO 设备没有使能
0xA00F8042	ERR_VO_DEV_HAS_ENABLED	VO 设备已经使能
0xA00F8043	ERR_VO_DEV_HAS_BINDED	VO 设备已经被绑定
0xA00F8044	ERR_VO_DEV_NOT_BINDED	VO 设备没有被绑定
0xA00F8045	ERR_VO_VIDEO_NOT_ENABLE	视频没有使能
0xA00F8046	ERR_VO_VIDEO_NOT_DISABLE	视频处于使能状态
0xA00F8047	ERR_VO_VIDEO_NOT_CONFIG	视频没有配置
0xA00F806E	ERR_VO_VIDEO_HAS_BINDED	视频已经绑定
0xA00F806F	ERR_VO_VIDEO_NOT_BINDED	视频没有绑定
0xA00F8048	ERR_VO_CHN_NOT_DISABLE	VO 通道处于使能状态
0xA00F8049	ERR_VO_CHN_NOT_ENABLE	VO 通道没有使能
0xA00F804A	ERR_VO_CHN_NOT_CONFIG	VO 通道没有被配置
0xA00F804B	ERR_VO_CHN_NOT_ALLOC	VO 通道没有分配
0xA00F806B	ERR_VO_CHN_AREA_OVERLAP	VO 通道区域重叠
0xA00F8014	ERR_VO_CHN_SAMESTATE	同样的状态，错误常见于状态转换
0xA00F8015	ERR_VO_CHN_INVALIDSTATE	非法的状态

0xA00F8016	ERR_VO_CHN_INCORRECT_STATE_TRANSITION	错误的状态转换
0xA00F8017	ERR_VO_CHN_INCORRECT_STATE_OPERATION	错误的状态行为
0xA00F804C	ERR_VO_INVALID_PATTERN	无效的样式
0xA00F804D	ERR_VO_INVALID_POSITION	无效级联位置（例如：组件通道绑定端口属性不一致）
0xA00F804E	ERR_VO_WAIT_TIMEOUT	等待超时
0xA00F804F	ERR_VO_INVALID_VFRAME	非法的视频帧
0xA00F8050	ERR_VO_INVALID_RECT_PARA	非法的矩形参数（rect）
0xA00F8051	ERR_VO_SETBEGIN_ALREADY	已经设置为开始
0xA00F8052	ERR_VO_SETBEGIN_NOTYET	还没有设置开始
0xA00F8053	ERR_VO_SETEND_ALREADY	已经设置为结束
0xA00F8054	ERR_VO_SETEND_NOTYET	还没有设置结束
	ERR_VO_GRP_INVALID_ID	
	ERR_VO_GRP_NOT_CREATE	
	ERR_VO_GRP_HAS_CREATED	
	ERR_VO_GRP_NOT_DESTROY	
	ERR_VO_GRP_CHN_FULL	
	ERR_VO_GRP_CHN_EMPTY	
	ERR_VO_GRP_CHN_NOT_EMPTY	
	ERR_VO_GRP_INVALID_SYN_MODE	
	ERR_VO_GRP_INVALID_BASE_PTS	
	ERR_VO_GRP_NOT_START	
	ERR_VO_GRP_NOT_STOP	
	ERR_VO_GRP_INVALID_FRMRATE	
	ERR_VO_GRP_CHN_HAS_REG	
	ERR_VO_GRP_CHN_NOT_REG	
	ERR_VO_GRP_CHN_NOT_UNREG	
	ERR_VO_GRP_BASE_NOT_CFG	
0xA00F8065	ERR_VO_GFX_NOT_DISABLE	图形层处于使能状态

0xA00F8066	ERR_VO_GFX_NOT_BIND	图形层没有绑定
0xA00F8067	ERR_VO_GFX_NOT_UNBIND	图形层没有解绑
0xA00F8068	ERR_VO_GFX_INVALID_ID	非法的图形层 ID

4. 图像拼接

4.1. 概述

4.1.1. ISE 简介

ISE模块，该模块主要包括三个模式，单目鱼眼模式(ISEMODE_ONE_FISHEYE)、双目鱼眼模

式(ISEMODE_TWO_FISHEYE)、双目拼接模式(ISEMODE_TWO_ISH)。

单目鱼眼模式是对单路鱼眼镜头图像进行校正，支持包括360全景左右展开（WARP_PANO360）、360全景上下展开（WARP_180WITH2）、180度展开（WARP_PANO180）和 Normal（WARP_NORMAL）四种校正模式，同时支持对单路普通镜头图像进行畸变校正（WARP_UNDISTORT）；

双目鱼眼模式主要作用是将两路鱼眼镜头图像拼接成一路 360 度全景图像输出；

双目广角拼接模式主要作用是将两路普通镜头的图片拼接成一路广角图像输出。

ISE模块的输入源包括以下二类：

- 用户态读取图像文件向 ISE 组件发送数据；
- 视频输入模块（VI模块）采集的图像直接发送到ISE组件。

4.1.2. 功能框图

ISE 组件功能框图如图 4-1 所示，ISE 组件接收外部原始图像数据，而不关心图像数据是来自哪个外部模块。由于硬件设计的原因，一个 ISE 组件对应一个 Group，一个 Group 最多可以创建 4 个 Port，其中 Port 0 为 ISE 硬件模块处理后输出的图像，Port 1 ~ Port 3 是对 Port 0 输出的图像进行缩放而得到的。Group 创建完后必须创建 Port 0，Port 1 ~ Port 3 可以根据实际需要依次创建，Port 1 ~ Port 3 支持无极缩放，图像缩放的范围为 Port 0 宽高的 1/8 ~ 1，各个 Port 之间相互独立，互不影响。

ISE 组件输入端的数据流来源有三种，分别是 VI 组件与 ISE 组件通过绑定方式传递 YUV 数据，VI 组件与 ISE 组件通过非绑定方式传递 YUV 数据，以及用户态读取图像文件（Image File）向 ISE 组件传递 YUV 数据。

ISE 组件输出端的数据流获取有三种，当 ISE 组件输入端（如 VI 组件）通过绑定方式向 ISE 组件传递数据时，输出端（如 VENC 组件或 VO 组件）可以通过绑定方式或者非绑定式获取 ISE 组件输出的图像，当使用绑定方式将 Port 0 与 VO/VENC 组件绑定后，同一个 Group 号的 ISE 组件的 Port 1 ~ Port 3 必须使用绑定方式获取 ISE 输出的图像，同理使用非绑定方式的时候也要遵循上述的规则；用户也可以通过调用 ISE 组件 API 获取 ISE 组件输出端的数据流。

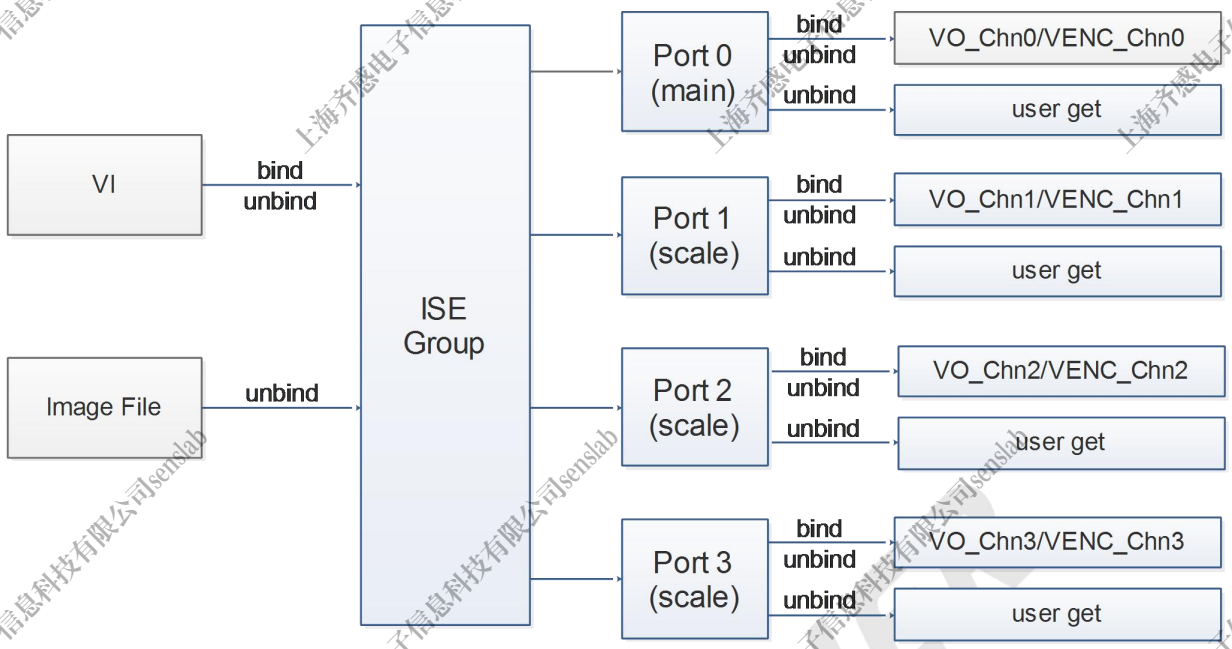


图 4-1 ISE 功能框图

4.1.3. ISE 组件 buffer 管理和使用

ISE组件包含输入和输出buffer，使用/处理情况如下：

（1）对于输入buffer(待处理YUV图像)：ISE组件本身不对外提供输入的buffer，在非绑定方式下，需要输入端根据输入图像大小开辟buffer供ISE组件使用，通过调用AW_MPI_ISE_SendPic()向ISE组件传递输入buffer, ISE组件内部会对输入的buffer进行管理，在使用完输入buffer后，ISE组件会通过事件回调函数AW_MPI_ISE_RegisterCallback返回事件MPP_EVENT_RELEASE_ISE_VIDEO_BUFFER，通知ISE组件输入端数据已经被处理，调用者收到此事件后可以自行对frame buffer进行释放等操作。在绑定方式下，由VI组件跟ISE组件内部自行完成buffer的操作交互，不需要调用者再额外处理。

（2）对于输出buffer(处理后YUV图像)：该buffer由ISE组件内部根据输出图像的大小提供，默认为5个buffer。在非绑定方式下，调用者通过调用AW_MPI_ISE_GetData()接口获取ISE组件输出buffer，在使用完输出buffer后必须及时调用AW_MPI_ISE_ReleaseData()将buffer归还给ISE组件。在绑定方式下，由ISE组件跟VENC组件或VO组件内部自行完成buffer的操作交互，不需要调用者再额外处理。

4.1.4. ISE 组件典型应用场景

下面根据典型的应用场景对 ISE 组件的使用进行描述

如图 4-2 所示，该场景是单目鱼眼 360 全景左右展开，创建一个物理设备 VIPP 0 和一个虚通道 Virvi0，将 VI 组件与 ISE 组件输入端进行绑定，ISE 组件输出端分别与编码通道和视频输出通道进行绑定。

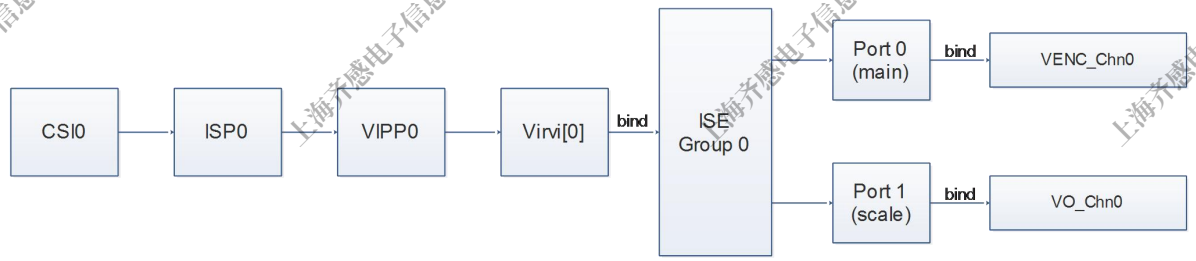


图 4-2 单目鱼眼 360 全景左右展开功能框图

如图 4-3 所示，该场景是单目鱼眼 PTZ，创建一个物理设备 VIPP 0 和三个虚通道 Virvi 0 ~ Virvi 2，将 VI 组件与三个 ISE 组件输入端进行绑定，每个 ISE 组件输出端分别与编码通道和视频输出通道进行绑定。

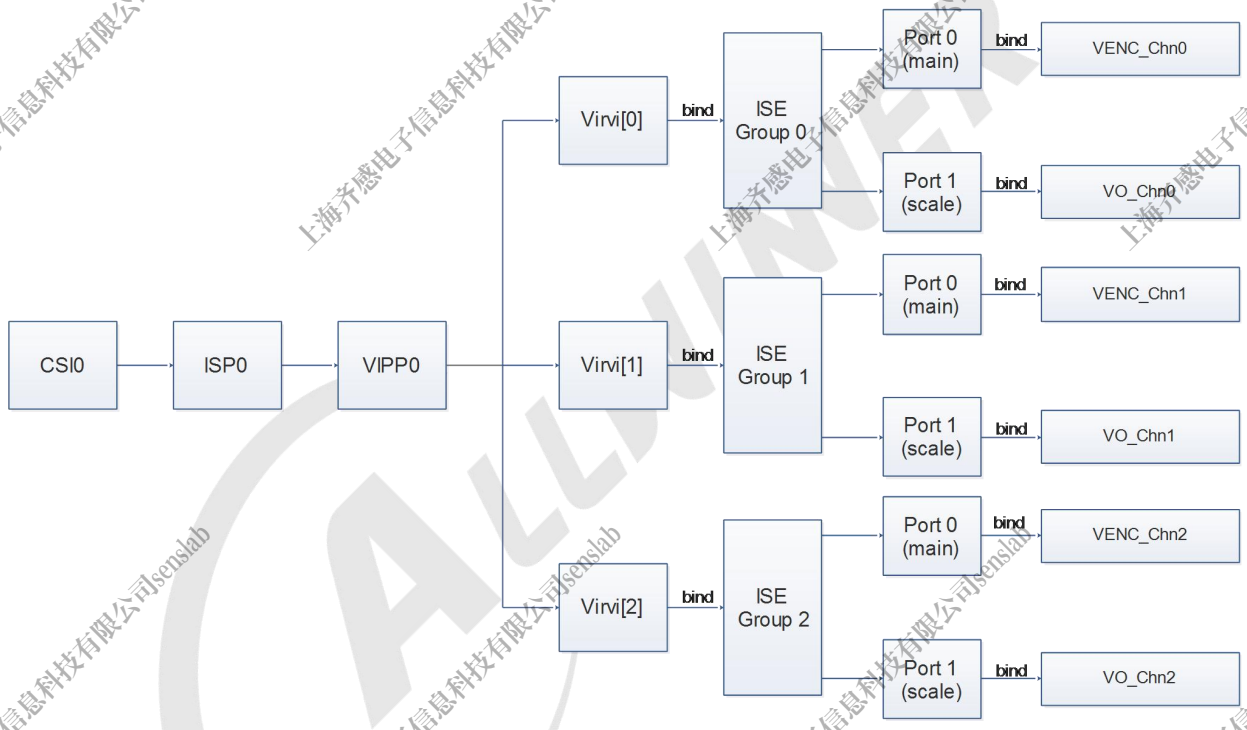


图 4-3 单目鱼眼 PTZ 功能框图

如图 4-4 所示，该场景是双目鱼眼拼接，创建两个物理设备 VIPP 0 和 VIPP 1，每个物理设备下创建一个虚通道 Virvi 0，ISE 组件输出端分别与编码通道和视频输出通道进行绑定。

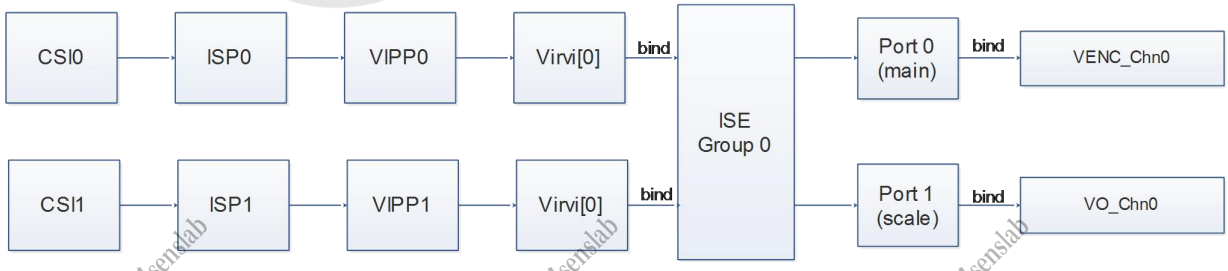


图 4-4 双目鱼眼拼接功能框图

如图 4-5 所示，该场景是双目广角拼接，创建两个物理设备 VIPP 0 和 VIPP 1，每个物理设备下

创建一个虚通道 Virvi 0，ISE 组件输出端分别与编码通道和视频输出通道进行绑定。

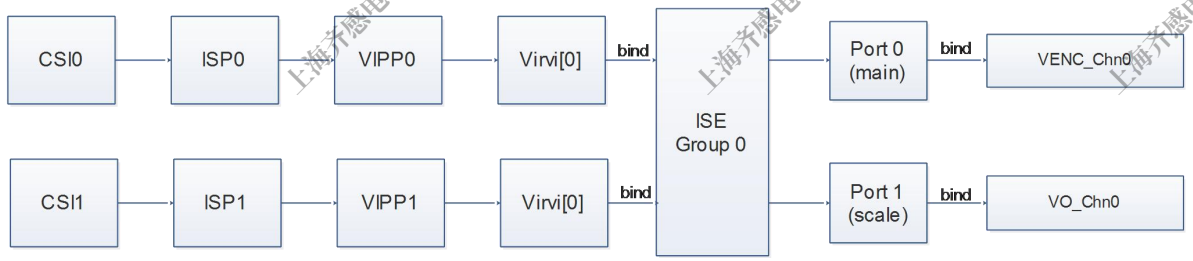


图 4-5 双目广角拼接功能框图

4.2. 状态转换与 API 接口

4.2.1. 状态图

每个 ISE 组件内部都有一个线程，其内部按状态机的方式工作，状态转化如图 4-6 所示：

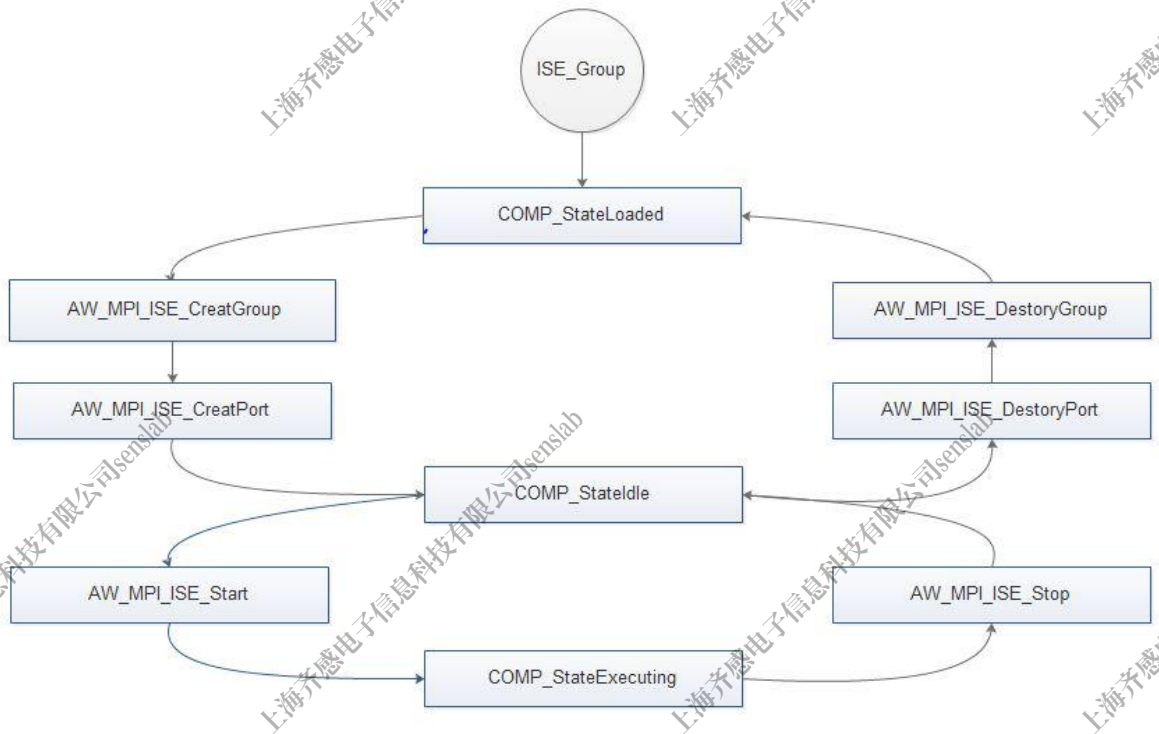


图 4-6 ISE 组件状态图

4.2.2. API 接口

AW_MPI_ISE_CreateGroup

【目的】

创建ISE组，选择ISE工作模式

【语法】

ERRORTYPE AW_MPI_ISE_CreateGroup(ISE_GRP IseGrp, ISE_GROUP_ATTR_S *pGrpAttr);

【参数】

参数	描述	输入/输出
IseGrp	组 ID 号	输入
pGrpAttr	组属性，选择 ISE 组件工作模式	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码

【需求】

头文件：mpi_ise.h

库文件：libmpp_ise.so

【注意】

目前只支持单目鱼眼、双目鱼眼两种模式下创建多个Group,双目拼接模式下只允许创建一个Group

【举例】

无

AW_MPI_ISE_DestroyGroup

【目的】

销毁ISE组

【语法】

ERRORTYPE AW_MPI_ISE_DestroyGroup(ISE_GRP IseGrp);

【参数】

参数	描述	输入/输出
IseGrp	组 ID 号	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码

【需求】

头文件：mpi_ise.h

库文件：libmpp_ise.so

【注意】

无

【举例】

无

AW_MPI_ISE_SetGrpAttr

【目的】

设置ISE 组的属性

【语法】

ERRORTYPE AW_MPI_ISE_SetGrpAttr(ISE_GRP IseGrp, ISE_GROUP_ATTR_S *pGrpAttr);

【参数】

参数	描述	输入/输出
IseGrp	组 ID 号	输入
pGrpAttr	组属性，选择 ISE 组件工作模式	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码

【需求】

头文件：mpi_ise.h

库文件：libmpp_ise.so

【注意】

无

【举例】

无

AW_MPI_ISE_GetGrpAttr

【目的】

获取ISE组的属性

【语法】

ERRORTYPE AW_MPI_ISE_GetGrpAttr(ISE_GRP IseGrp, ISE_GROUP_ATTR_S *pGrpAttr);

【参数】

参数	描述	输入/输出
IseGrp	组 ID 号	输入
pGrpAttr	组属性，选择 ISE 组件工作模式	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码

【需求】

头文件：mpi_ise.h

库文件：libmpp_ise.so

【注意】

无

【举例】

无

AW_MPI_ISE_CreatePort

【目的】

创建ISE输出端口

【语法】

ERRORTYPE AW_MPI_ISE_CreatePort(ISE_GRP IseGrp, ISE_CHN IsePort, ISE_CHN_ATTR_S *pChnAttr);

【参数】

参数	描述	输入/输出
IseGrp	组 ID 号	输入
IsePort	端口 ID 号	输入
pChnAttr	端口属性，设置输出端口属性	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码

【需求】

头文件：mpi_ise.h

库文件：libmpp_ise.so

【注意】

每个 ISE Group 都有 4 个端口可以输出，每个端口相互独立。

【举例】

无

AW_MPI_ISE_DestroyPort

【目的】

销毁ISE输出端口

【语法】

ERRORTYPE AW_MPI_ISE_DestroyPort(ISE_GRP IseGrp, ISE_CHN IsePort);

【参数】

参数	描述	输入/输出
IseGrp	组 ID 号	输入
IsePort	端口 ID 号	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码

【需求】

头文件：mpi_ise.h

库文件：libmpp_ise.so

【注意】

无

【举例】

无

AW_MPI_ISE_GetPortAttr

【目的】

获取ISE输出端口属性

【语法】

ERRORTYPE AW_MPI_ISE_GetPortAttr(ISE_GRP IseGrp, ISE_CHN IsePort, ISE_CHN_ATTR_S *pChnAttr);

【参数】

参数	描述	输入/输出
IseGrp	组 ID 号	输入
IsePort	端口 ID 号	输入

pChnAttr	端口属性，设置输出端口属性	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码

【需求】

头文件：mpi_ise.h

库文件：libmpp_ise.so

【注意】

无

【举例】

无

AW_MPI_ISE_SetPortAttr

【目的】

设置ISE输出端口属性

【语法】

ERRORTYPE AW_MPI_ISE_SetPortAttr(ISE_GRP IseGrp, ISE_CHN IsePort, ISE_CHN_ATTR_S *pChnAttr);

【参数】

参数	描述	输入/输出
IseGrp	组 ID 号	输入
IsePort	端口 ID 号	输入
pChnAttr	端口属性，设置输出端口属性	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码

【需求】

头文件：mpi_ise.h

库文件：libmpp_ise.so

【注意】

无

【举例】

无

AW_MPI_ISE_Start

【目的】

启动ISE组件，所有创建的port会同时启动工作。

【语法】

ERRORTYPE AW_MPI_ISE_Start(ISE_GRP IseGrp);

【参数】

参数	描述	输入/输出
IseGrp	组 ID 号	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码

【需求】

头文件：mpi_ise.h

库文件：libmpp_ise.so

【注意】

无

【举例】

无

AW_MPI_ISE_Stop

【目的】

停止ISE组件，所有创建的port会同时停止工作。

【语法】

ERRORTYPE AW_MPI_ISE_Stop(ISE_GRP IseGrp);

【参数】

参数	描述	输入/输出
IseGrp	组 ID 号	输入

【返回值】

返回值	描述
0	成功

非 0	失败，其值见错误码

【需求】

头文件：mpi_ise.h

库文件：libmpp_ise.so

【注意】

无

【举例】

无

AW_MPI_ISE_GetData

【目的】

在非绑定的情况下，从端口获取ISE处理后的数据

【语法】

ERRORTYPE AW_MPI_ISE_GetData(ISE_GRP IseGrp, ISE_CHN IsePort, VIDEO_FRAME_INFO_S *pstIseData, AW_S32 nMilliSec);

【参数】

参数	描述	输入/输出
IseGrp	组 ID 号	输入
IsePort	端口 ID 号	输入
pstIseData	ISE 组件处理后的数据流	输出
nMilliSec	获取数据的超时时间 -1 表示阻塞模式； 0 表示非阻塞模式； >0 表示阻塞 nMilliSec 毫秒，超时则报错返回。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码

【需求】

头文件：mpi_ise.h

库文件：libmpp_ise.so

【注意】

绑定模式下，此函数不起作用

【举例】

AW_MPI_ISE_ReleaseData

【目的】

在非绑定的情况下，从端口释放ISE处理后的数据

【语法】

ERRORTYPE AW_MPI_ISE_ReleaseData(ISE_GRP IseGrp, ISE_CHN IsePort, VIDEO_FRAME_INFO_S * pstIseData);

【参数】

参数	描述	输入/输出
IseGrp	组 ID 号	输入
IsePort	端口 ID 号	输入
pstIseData	ISE 组件处理后的数据流	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码

【需求】

头文件：mpi_ise.h

库文件：libmpp_ise.so

【注意】

绑定模式下，此函数不起作用

【举例】

无

AW_MPI_ISE_SendPic

【目的】

在非绑定的情况下，用户向ISE组件发送需要处理的数据

【语法】

ERRORTYPE AW_MPI_ISE_SendPic(ISE_GRP IseGrp, VIDEO_FRAME_INFO_S *pstUserFrame0, VIDEO_FRAME_INFO_S *pstUserFrame1, AW_S32 nMilliSec);

【参数】

参数	描述	输入/输出
IseGrp	组 ID 号	输入
pstUserFrame0	第一路源图像数据	输入

pstUserFrame1	第二路源图像数据	输入
nMilliSec	发送数据的超时时间， -1 表示阻塞模式； 0 表示非阻塞模式； >0 表示阻塞 nMilliSec 毫秒，超时则报错返回。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码

【需求】

头文件：mpi_ise.h

库文件：libmpp_ise.so

【注意】

绑定模式下，此函数不起作用

【举例】

AW_MPI_ISE_RegisterCallback

【目的】

在非绑定的情况下，ISE通知用户帧已经处理完

【语法】

ERRORTYPE AW_MPI_ISE_RegisterCallback(ISE_GRP IseGrp, MPPCallbackInfo *pCallback);

【参数】

参数	描述	输入/输出
IseGrp	组 ID 号	输入
pCallback	回调参数	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码

【需求】

头文件：mpi_ise.h

库文件：libmpp_ise.so

【注意】

绑定模式下，此函数不起作用

【举例】

无

AW_MPI_ISE_SetISEFreq

【目的】

设置 ISE 模块时钟频率。

【语法】

ERRORTYPE AW_MPI_ISE_SetISEFreq(ISE_GRP IseGrp, int nFreq)

【参数】

参数	描述	输入/输出
IseGrp	组 ID 号	输入
nFreq	引擎时钟频率，单位 MHz。范围：[432-648]，且为 12 的倍数 单目鱼眼和双目拼接模式默认 432 MHz，双目鱼眼模式默认为 576MHz。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

头文件：mpi_ise.h

库文件：libmpp_ise.so

【注意】

无。

【举例】

无。

4.3. 数据结构

ISE_GROUP_ATTR_S

【说明】

定义 ISE 组属性

【定义】

```
typedef  struct
{
```

```
    AW_U32    iseMode;
} ISE_GROUP_ATTR_S;
```

【成员】

成员名称	描述
iseMode	设置 ISE 组属性，选择 ISE 组件工作模式

【注意事项】

无

【相关数据类型及接口】

无

WARP_Type_MO

【说明】

定义单目鱼眼矫正方式

【定义】

```
typedef    enum    _WARP_Type_MO_
{
    WARP_PANO180 = 0x0001,
    WARP_PANO360 = 0x0002,
    WARP_NORMAL = 0x0003,
    WARP_UNDISTORT = 0x0004,
    WARP_180WITH2 = 0x0005
}WARP_Type_MO;
```

【成员】

成员名称	描述
WARP_PANO180	180 全景校正模式
WARP_PANO360	360 全景校正模式（左右展开）
WARP_NORMAL	Normal 校正模式
WARP_UNDISTORT	镜头畸变校正模式
WARP_180WITH2	360 全景校正模式（上下展开）

【注意事项】

无

【相关数据类型及接口】

无

MOUNT_Type_MO

【说明】

定义单目鱼眼镜头安装方式

【定义】

```
typedef    enum _MOUNT_Type_MO_{
```

```
    MOUNT_TOP = 0x0001,
    MOUNT_WALL = 0x0002,
    MOUNT_BOTTOM = 0x0003
}MOUNT_Type_MO;
```

【成员】

成员名称	描述
MOUNT_TOP	顶装模式
MOUNT_WALL	壁装模式
MOUNT_BOTTOM	地装模式

【注意事项】

无

【相关数据类型及接口】

无

ISE_CFG_PARA_MO

【说明】

定义单目鱼眼模式相关配置

【定义】

```
typedef   struct   _ISE_CFG_PARA_MO_
{
    WARP_Type_MO        dewarp_mode;
    MOUNT_Type_MO       mount_mode;
    int                 in_h;
    int                 in_w;
    int                 in_luma_pitch;
    int                 in_chroma_pitch;
    int                 in_yuv_type;
    float               pan;
    float               tilt;
    float               zoom;
    int                 out_en[4];
    int                 out_h[4];
    in                  out_w[4];
    int                 out_luma_pitch[4];
    int                 out_chroma_pitch[4];
    int                 out_flip[4];
    int                 out_mirror[4];
    int                 out_yuv_type;
    float               p;
    float               cx;
    float               cy;
    float               fx;
```

```
    float           fy;
    float           cxd;
    float           cyd;
    float           fxd;
    float           fyd;
    float           k[6];
    float           p_undis[2];
    double          calib_matr[3][3];
    double          calib_matr_cv[3][3];
    double          distort[8];
    char            reserved[32];
}ISE_CFG_PARA_MO;
```

【成员】

成员名称	描述
dewarp_mode	校正模式，静态属性
mount_mode	安装方式，静态属性
in_h	源图像高度，静态属性
in_w	源图像宽度，静态属性
in_luma_pitch	源图像亮度（Y 分量）pitch 值，32 的倍数最大值为 4096，静态属性
in_chroma_pitch	源图像色度（UV 分量）pitch 值，32 的倍数最大值为 4096，静态属性
in_yuv_type	源图像 YUV 格式，0 表示 NV21 格式，静态属性
pan	Normal 模式下视场角的水平转动，动态属性
tilt	Normal 模式下视场角的垂直转动，动态属性
zoom	Normal 模式下视场角大小，动态属性
out_en[4]	是否使能单目鱼眼硬件通道，硬件通道 0～3 分别对应 Port 0～3 0：不使能 1：使能 out_en[0]：静态属性 out_en[1～3]：动态属性，若不使能则需要把对应的 Port 销毁掉，在需要使能时重新创建 Port
out_h[4]	目标图像的高度，硬件通道 0～3 分别对应 Port 0～3 out_h[1～3]须满足[0.125,1] * out_h[0]，4 的倍数 out_h[0]：静态属性 out_h[1～3]：动态属性
out_w[4]	目标图像的宽度，硬件通道 0～3 分别对应 Port 0～3 out_w[1～3]须满足[0.125,1] * out_w[0]，8 的倍数 out_w[0]：静态属性 out_w[1～3]：动态属性
out_luma_pitch[4]	目标图像亮度（Y 分量）pitch 值，16 或 32 的倍数 硬件通道 0～3 分别对应 Port 0～3 out_luma_pitch[0]：静态属性 out_luma_pitch[1～3]：动态属性
out_chroma_pitch[4]	目标图像色度（UV 分量）pitch 值，16 或 32 的倍数

	硬件通道 0～3 分别对应 Port 0～3 out_chroma_pitch[0]：静态属性 out_chroma_pitch[1～3]：动态属性
out_flip[4]	是否使能通道图像水平翻转，硬件通道 0～3 分别对应 Port 0～3 0：不使能 1：使能 out_flip[0]：静态属性 out_flip[1～3]：动态属性
out_mirror[4]	是否使能通道图像镜像，硬件通道 0～3 分别对应 Port 0～3 0：不使能 1：使能 out_ mirror[0]：静态属性 out_ mirror[1～3]：动态属性
out_yuv_type	目标图像 YUV 格式，0 表示 NV21 格式，静态属性
p	鱼眼镜头校正模式参数，源图像的镜头参数，镜头焦距 需要使用标定工具进行标定，静态属性
cx	源图像的镜头参数，光心水平坐标，需要使用标定工具进行标定 静态属性
cy	源图像的镜头参数，光心垂直坐标，需要使用标定工具进行标定 静态属性
fx	普通镜头畸变校正模式参数，畸变图像的水平方向焦距 需要使用标定工具进行标定，静态属性
fy	普通镜头畸变校正模式参数，畸变图像的垂直方向焦距 需要使用标定工具进行标定，静态属性
cxd	普通镜头畸变校正模式参数，畸变校正后图像光心水平坐标 需要使用标定工具进行标定，静态属性
cyd	普通镜头畸变校正模式参数，畸变校正后图像光心垂直坐标 需要使用标定工具进行标定，静态属性
fxd	普通镜头畸变校正模式参数，畸变校正后图像水平方向焦距 需要使用标定工具进行标定，静态属性
fyd	普通镜头畸变校正模式参数，畸变校正后图像垂直方向焦距 需要使用标定工具进行标定，静态属性
k[6]	普通镜头畸变校正模式参数，径向畸变系数 需要使用标定工具进行标定，静态属性
p_undis[2]	普通镜头畸变校正模式参数，法向畸变系数 需要使用标定工具进行标定，静态属性
calib_matr[3][3]	扩展类型，暂不支持
calib_matr_cv[3][3]	扩展类型，暂不支持
distort[8]	扩展类型，暂不支持
reserved[32]	保留，暂不涉及

【注意事项】

1、由于单目鱼眼模式下不同的校正模式，其输入图像的宽高设置、安装方式也有所不同，如图 2-2 所示。

校正模式 dewarp_mode	输入图像大小 in_h in_w		安装方式 mount_mode	
WARP_PANO180 （180模式）	宽高为8的倍数	输入宽高1:1 最大支持3000*3000 最小为768*768	无	
WARP_PANO360 （360模式）		输入宽高1:1 最大支持2048*2048 最小为768*768	MOUNT_TOP（顶装）	无
			MOUNT_BOTTON（地装）	
WARP_NORMAL （PTZ模式）		输入宽高1:1 最大支持3000*3000 最小为768*768	MOUNT_TOP（顶装）	P:[0,360]T:[0,90]Z:[1,4]
			MOUNT_WALL（壁装）	P:[25,155]T:[25,155]Z:[1,4]
			MOUNT_BOTTON（地装）	P:[0,360]T:[0,90]Z:[1,4]
WARP_UNDISTORT （镜头畸变矫正模式）		输入宽高16:9/4:3/1:1 最大支持4096*2304/ 4096*3072/4096*4096 最小为 1152x864/1366x768/768*768	无	
WARP_180WITH2 （第二种360模式）		输入宽高1:1 最大支持2048*2048 最小为768*768	MOUNT_TOP（顶装）	无
			MOUNT_BOTTON（地装）	

图 2-2

2、图像水平翻转和镜像每个 Port 只能使能一个。

3、Normal 模式下 PTZ 参数在 ISE 组件运行过程中支持动态设置，动态设置间隔时间不得小于 60ms 且必须在 ISE Group 处于 COMP_StateExecuting 之后再进行设置。

【相关数据类型及接口】

无

ISE_CFG_PARA_BI

【说明】

定义双目鱼眼模式相关配置

【定义】

```
typedef   struct   _ISE_CFG_PARA_BI_{
    int                 in_h;
    int                 in_w;
    int                 in_luma_pitch;
    int                 in_chroma_pitch;
    int                 in_yuv_type;
    int                 out_en[4];
```

```
    int                 out_h[4];
    int                 out_w[4];
    int                 out_flip[4];
    int                 out_mirror[4];
    int                 out_luma_pitch[4];
    int                 out_chroma_pitch[4];
    int                 out_yuv_type;
    float               p0;
    int                 cx0;
    int                 cy0;
    float               p1;
    int                 cx1;
    int                 cy1;
    float               calib_matr[3][3];
    double              calib_matr_cv[3][3];
    double              distort[8];
    float               change_focus;
    int                 feather;
    int                 trans_width;
    int                 findpath_width;
    int                 pyr_level;
    int                 findpath_enable;
    float               inv_scale;
    char                reserved[32];
}ISE_CFG_PARA_BI;
```

【成员】

成员名称	描述
in_h	源图像高度，4 的倍数取值范围[320,4032]，静态属性
in_w	源图像宽度，4 的倍数取值范围[320,4032]，静态属性
in_luma_pitch	源图像亮度（Y 分量）pitch 值，32 倍数，静态属性
in_chroma_pitch	源图像色度（UV 分量）pitch 值，32 倍数，静态属性
in_yuv_type	源图像 YUV 格式，0 表示 NV21 格式，静态属性
out_en[4]	是否使能双目鱼眼硬件通道，硬件通道 0～3 分别对应 Port 0～3 0：不使能 1：使能 out_en[0]：静态属性 out_en[1～3]：动态属性，若不使能则需要把对应的 Port 销毁掉，在需要使能时重新创建 Port
out_h[4]	目标图像的高度，硬件通道 0～3 分别对应 Port 0～3 out_h[0]为 4 的倍数取值范围[320,4032] out_h[1～3]须满足[0.125,1] * out_h[0]，4 的倍数 out_h[0]：静态属性 out_h[1～3]：动态属性

out_w[4]	目标图像的宽度，硬件通道 0 ~ 3 分别对应 Port 0 ~ 3 out_w[0]为 8 的倍数取值范围[640, 8064] out_w[1 ~ 3]须满足[0.125,1] * out_w[0]，8 的倍数 out_w[0]：静态属性 out_w[1～3]：动态属性
out_flip[4]	是否使能通道图像水平翻转，硬件通道 0 ~ 3 分别对应 Port 0 ~ 3 0：不使能 1：使能 out_flip[0]：静态属性 out_flip[1～3]：动态属性
out_mirror[4]	是否使能通道图像镜像，硬件通道 0 ~ 3 分别对应 Port 0 ~ 3 0：不使能 1：使能 out_ mirror[0]：静态属性 out_ mirror[1～3]：动态属性
out_luma_pitch[4]	目标图像亮度（Y 分量）pitch 值，16 或 32 的倍数 硬件通道 0 ~ 3 分别对应 Port 0 ~ 3 out_luma_pitch[0]：静态属性 out_luma_pitch[1～3]：动态属性
out_chroma_pitch[4]	目标图像色度（UV 分量）pitch 值，16 或 32 的倍数 硬件通道 0 ~ 3 分别对应 Port 0 ~ 3 out_chroma_pitch [0]：静态属性 out_chroma_pitch [1～3]：动态属性
out_yuv_type	目标图像 YUV 格式，0 表示 NV21 格式，静态属性
p0	第一组源图像的镜头参数，镜头焦距，需要使用标定工具进行标定 静态属性
cx0	第一组源图像的镜头参数，光心水平坐标，需要使用标定工具进行标定 静态属性
cy0	第一组源图像的镜头参数，光心垂直坐标，需要使用标定工具进行标定 静态属性
p1	第二组源图像的镜头参数，镜头焦距，需要使用标定工具进行标定 静态属性
cx1	第二组源图像的镜头参数，光心水平坐标，需要使用标定工具进行标定 静态属性
cy1	第二组源图像的镜头参数，光心垂直坐标，需要使用标定工具进行标定 静态属性
calib_matr[3][3]	源图像的镜头参数，镜头旋转矩阵，需要使用标定工具进行标定 静态属性
change_focus	软件拼接算法参数，视差校正系数，静态属性
feather	软件拼接算法参数，重叠区域半径，静态属性
trans_width	软件拼接算法参数，过渡区大小，静态属性
findpath_width	软件拼接算法参数，最佳路径查找区域大小，静态属性
pyr_level	软件拼接算法参数，金字塔级数设定，静态属性

findpath_enable	软件拼接算法参数，最佳路径使能控制，静态属性
inv_scale	软件拼接算法参数，最佳路径查找缩放比例，静态属性
calib_matr_cv[3][3]	扩展类型，暂不支持
distort[8]	扩展类型，暂不支持
reserved[32]	保留，暂不涉及

【注意事项】

1、 输入图像宽高要求 1:1。

2、 图像水平翻转和镜像每个 Port 只能使能一个。

【相关数据类型及接口】

无

ISE_CFG_PARA_ STI

【说明】

定义双目拼接模式相关配置

【定义】

```
typedef   struct   _ISE_CFG_PARA_STI_{
    int                 ncam;
    int                 in_h;
    int                 in_w;
    int                 pano_h;
    int                 pano_w;
    int                 ov;
    int                 yuv_type;
    float               p0;
    float               p1;
    int                 offset_r2l;
    float               pano_fov;
    float               t_angle;
    int                 stre_en;
    float               stre_coeff;
    float               hfov;
    float               wfov;
    float               wfov_rev;
    double              calib_matr[3][3];
    double              calib_matr_cv[3][3];
    double              distort[8];
    int                 in_luma_pitch;
    int                 in_chroma_pitch;
    int                 pano_luma_pitch;
    int                 pano_chroma_pitch;
    char                reserved[32];
}ISE_CFG_PARA_STI;
```

【成员】

成员名称	描述
ncam	相机数量，静态属性
in_h	源图像高度，2 的倍数取值范围[120,1728]，静态属性
in_w	源图像宽度，取值范围[160,2560] 宽高比为 4:3 时，为 8 的倍数；宽高为 16:9 时，为 32 的倍数，静态属性
pano_h	目标图像 Port 0 的高度，4 的倍数取值范围[64,2048]，静态属性
pano_w	目标图像 Port 0 的宽度 与输入图像宽度的关系为：pano_w = [0.8,1] * 2 * in_w 16 的倍数取值范围[320,4096] 静态属性
ov	重叠区总宽度，与输出宽度的关系为：ov < 1/10 * pano_w 小于等于 320 且为 16 的倍数，静态属性
yuv_type	图像 YUV 格式，0 表示 NV21 格式，静态属性
p0	镜头 0 光轴与水平线的夹角，取值范围[0,180]，静态属性
p1	镜头 1 光轴与水平线的夹角，取值范围[0,180]，静态属性
offset_r2l	右图相对左图高度方向偏移像素数，暂不开放，默认为 0
pano_fov	输出图像视场角，为大约值，需要使用标定工具进行标定，静态属性
t_angle	俯仰角，T 值，单位为度，暂不开放，默认为 0
stre_en	视场拉伸开关，暂不开放，默认为 0
stre_coeff	视场拉伸系数，>1.0，暂不开放，默认为 0
hfov	当前分辨率下水平视场角，单位为度，需要使用标定工具进行标定 静态属性
wfov	当前分辨率下垂直视场角，单位为度（只配置 wfov 即可） 需要使用标定工具进行标定，静态属性
wfov_rev	水平视场角修正值，单位为度，需要使用标定工具进行标定，静态属性
calib_matr[3][3]	未校正相机矩阵，需要使用标定工具进行标定，静态属性
calib_matr_cv[3][3]	畸变校正后相机矩阵，需要使用标定工具进行标定，静态属性
distort[8]	畸变系数，需要使用标定工具进行标定，静态属性
in_luma_pitch	源图像亮度（Y 分量）pitch 值，32 的倍数，静态属性
in_chroma_pitch	源图像色度（UV 分量）pitch 值，32 的倍数，静态属性
pano_luma_pitch	目标图像 Port0 亮度（Y 分量）pitch 值，16 或 32 的倍数，静态属性
pano_chroma_pitch	目标图像 Port0 图像色度（UV 分量）pitch 值，16 或 32 的倍数 静态属性
reserved[32]	保留，暂不涉及

【注意事项】

图像水平翻转和镜像只能使能一个。

【相关数据类型及接口】

无

ISE_PROCCFG_PARA_STI

【说明】

定义双目拼接模式相关配置

【定义】

```
typedef   struct   _ISE_PROCCFG_PARA_STI_{
    int                     bgfgmode_en;
    int                     pano_flip;
    int                     pano_mirr;
    int                     scalar_h[3];
    int                     scalar_w[3];
    int                     scalar_luma_pitch[3];
    int                     scalar_chroma_pitch[3];
    int                     scalar_flip[3];
    int                     scalar_mirr[3];
    int                     scalar_en[3];
    char                    reserved[32];
}ISE_PROCCFG_PARA_STI;
```

【成员】

成员名称	描述
bgfgmode_en	是否使能背景建模 0：不使能 1：使能 动态参数
pano_flip	是否使能目标图像 Port0 水平翻转 0：不使能 1：使能 静态属性
pano_mirr	是否使能目标图像 Port0 镜像 0：不使能 1：使能 静态属性
scalar_h[3]	scale 通道目标图像高度，scalar_h[0～3]分别对应 Port 1～3 scalar_h [1～3]须满足[0.125,1] * pano_h，4 的倍数，动态属性
scalar_w[3]	scale 通道目标图像宽度，scalar_w[0～3]分别对应 Port 1～3 scalar_w[1～3]须满足[0.125,1] * pano_w，8 的倍数，动态属性
scalar_luma_pitch[3]	目标图像亮度（Y 分量）pitch 值，16 或 32 的倍数 scalar_luma_pitch[0～3]分别对应 Port 1～3，动态属性
scalar_chroma_pitch[3]	目标图像色度（UV 分量）pitch 值，16 或 32 的倍数 scalar_ chroma _pitch[0～3]分别对应 Port 1～3，动态属性
scalar_flip[3]	是否使能通道图像水平翻转，scalar_flip[0～3]分别对应 Port 1～3 0：不使能 1：使能 动态属性
scalar_mirr[3]	是否使能通道图像镜像，scalar_mirr[0～3]分别对应 Port 1～3

	0：不使能 1：使能 动态属性
scalar_en[3]	是否使能双目拼接 scale 硬件通道，scalar_en[0～3]分别对应 Port 1～3 0：不使能 1：使能 scalar_en [1～3]：动态属性，若不使能则需要把对应的 Port 销毁掉，在需要使能时重新创建 Port
reserved[32]	保留，暂不涉及

【注意事项】

1、背景建模功能使能在 ISE 组件运行过程中支持动态设置。

2、图像水平翻转和镜像每个 Port 只能使能一个。

【相关数据类型及接口】

无

ISE_BGFG_PARA_STI

【说明】

定义双目拼接模式背景建模相关配置

【定义】

```
typedef   struct {
    int             bgfg_intvl;
    int             getbgd_intvl;
    int             bgfg_sleep_ms;
}ISE_BGFG_PARA_STI;
```

【成员】

成员名称	描述
bgfg_intvl	每隔多少个 SLEEP_MS 进行一次 model
getbgd_intvl	每隔多少帧获取背景图片
bgfg_sleep_ms	SC16 队列非空时，sleep 多少 MS

【注意事项】

背景建模参数在 ISE 组件运行过程中支持动态设置。

【相关数据类型及接口】

无

MODE_ATTR

【说明】

定义 ISE 组件模式属性

【定义】

```
typedef    union {
    FISH            mFish;
```

```
    DFISH       mDFish;
    ISE         mIse;
} MODE_ATTR;
```

【成员】

成员名称	描述
mode_attr	定义 ISE 组件模式属性，选择 ISE 组件工作模式

【注意事项】

无

【相关数据类型及接口】

无

ISE_CHN_ATTR_S

【说明】

定义 ISE 输出端口属性

【定义】

```
typedef struct
{
    MODE_ATTR       mode_attr;
    int             buffer_num;
} ISE_CHN_ATTR_S;
```

【成员】

成员名称	描述
mode_attr	设置相应模式下的 ISE 端口属性
buffer_num	设置 ISE 端口 buffer 个数，每个端口 buffer 个数可以单独设置，默认为 5 个

【注意事项】

无

【相关数据类型及接口】

无

4.4. 错误码

错误码	宏定义	描述
0xa0e58002	AW_ERR_ISE_INVALID_CHNID	无效 ISE 端口号
0xa0e58003	AW_ERR_ISE_INVALID_PARAM	无效参数
0xa0e58004	AW_ERR_ISE_EXIST	试图申请或创建已经存在的端口或资源
0xa0e58005	AW_ERR_ISE_UNEXIST	试图申请或创建不存在的端口或资源
0xa0e58006	AW_ERR_ISE_NULL_PTR	函数参数中有空指针

0xa0e58007	AW_ERR_ISE_NOT_CONFIG	使用前未配置
0xa0e58008	AW_ERR_ISE_NOT_SUPPORT	不支持的参数或功能
0xa0e58009	AW_ERR_ISE_NOT_PERM	该操作不允许执行，如试图修改静态参数
0xa0e5800c	AW_ERR_ISE_NOMEM	无可用内存
0xa0e5800d	AW_ERR_ISE_NOBUF	无可用缓存
0xa0e5800e	AW_ERR_ISE_BUF_EMPTY	缓存为空
0xa0e58010	AW_ERR_ISE_SYS_NOTREADY	系统没有初始化或没有加载相应模块
0xa0e58012	AW_ERR_ISE_BUSY	设备忙
0xa0e58013	AW_ERR_ISE_TIMEOUT	设备超时
0xa0e58014	AW_ERR_ISE_SAMESTATE	状态相同（常见于状态转换）
0xa0e58015	AW_ERR_ISE_INVALIDSTATE	无效的状态
0xa0e58016	AW_ERR_ISE_INCORRECT_STATE_TRANSITION	不正确的状态转换
0xa0e58017	AW_ERR_ISE_INCORRECT_STATE_OPERATION	不正确的状态操作

表 4-1 错误码

5. 视频编码

5.1. 概述

VENC模块，即视频编码模块。本模块支持多路实时编码，且每路编码独立，编码协议和编码profile 可以不同。

VENC模块的输入源包括以下二类：

(1)用户态读取图像文件向编码模块发送数据；

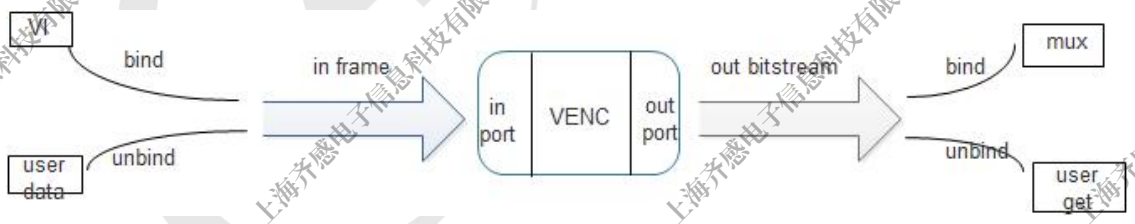
(2)视频输入（VIU）模块采集的图像直接发送到编码模块；

芯片支持的编码规格如下表所示。

H.264			JPEG	MOTION JPEG	H.265	MPEG-4
BP	MP	HP			MP	
支持	支持	支持	支持	支持	支持	支持

5.2. 功能描述

典型的编码流程包括了输入图像的接收、图像的编码、以及码流的输出等过程。通道支持接收YUV格式图像输入，支持格式为semi-planar YUV4:2:0、 semi-planar YVU4:2:0、planar YUV4:2:0以及全志自定义aw-afbc格式(yuv)。通道模块接收外部原始图像数据，而不关心图像数据是来自哪个外部模块。



每个venc通道最多能绑定一个输入端口，一个输出端口。

模块buffer使用情况：venc模块包含输入frame manager与输出编码bit stream manager，二者的buffer使用/处理情况如下。

- 对于输入待编码frame(yuv)：venc模块本身不对外提供frame的buffer，由venc输入端自己解决输入frame的buffer，venc模块内部frame buffer manager在获取in frame后，会通知venc输入端frame数据已经被处理，可以释放该frame的buffer。对于venc模块输入端绑定情况，由venc模块跟venc模块输入端口组件内部自行完成frame buffer的操作交互，不需要调用者再额外处理；对于venc输入端口非绑定的情况，此时组件会发出
- MPP_EVENT_RELEASE_VIDEO_BUFFER消息，调用者收到此消息后可以自行对

frame buffer进行释放等操作。

- 对于编码输出bit stream：由venc模块本身提供输出bit stream的buffer，调用者在获取完已编码数据后必须及时归还buffer给venc模块。对于venc输出端口绑定方式，由venc模块跟venc输出端口组件内部自行完成bit stream buffer操作交互，不需要再额外处理；对于venc输出端非绑定方式，调用者在调用AW_MPI_VENC_GetStream()接口成功获取已编码bit stream数据(处理)后，需调用AW_MPI_VENC_ReleaseStream()将bit stream buffer还给venc模块。

具体使用可参考sample_venc(输入输出端口都非绑定)、sample_venc2muxer（输入端口非绑定输出绑定)、sample_vi2venc(输入端口绑定输出端口非绑定)、sample_vi2venc2muxer（输入端口与输出端口都绑定)。

注意事项：模块（通道）注销时，会等待自己所有对外提供的数据buffer回收，因此对于绑定方式来讲，要注意各相关模块的注销顺序。如：venc -> mux方式时，注销时先注销mux，再注销venc。

5.2.1. 缩放功能

编码器支持对输入源进行放大或者缩小编码，宽度和高度可分别最大放大 8 倍，最小缩小 4 倍；实际缩放时的比例根据用户在 VencBaseConfig 设定的输入源的分辨率和编码的分辨率自动计算。输入源的宽度和编码输出的宽度要求 8 对齐，输入源的高度可不比满足 8 对齐，如果其不是 8 对齐，建议对输入源的非 8 对齐部分用最后一行的数据填充。

5.2.2. 旋转功能

编码器支持对输入源进行旋转编码，旋转角度为顺时针 90 度、180 度、360 度.

5.2.3. 码率控制

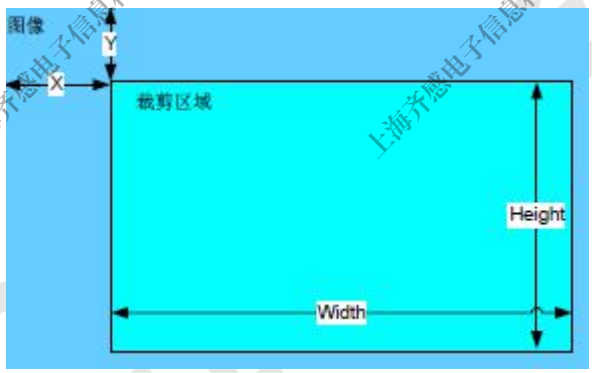
H264 和 H265 编码支持 CBR，VBR，FIXQP，QPMAP 的码率控制方式，MJPEG 编码只支持 CBR 和 FIX_QUALITY 的码率控制方式。

- CBR：固定比特率，即在码率统计时间内保证码率平稳，当前默认的码率统计时间是 1s；如果用户设定的帧率与实际的帧率不一致，则实际的码率与设定的码率与帧率成线性比例；
- VBR：可变比特率，即在码率统计时间内编码码率波动，从而保证编码图像质量稳定；当前通过编码驱动内部统计已编码帧的 mv 信息，对整体运动状况作出估计，为静态场景帧少分配目标比特量，为动态场景帧多分配目标比特量。用户可通过 MotionParam 数据结构设定运动和静止场景的等级，以及运动帧和静止帧占用码率的比例，具体请参看 MotionParam 定义；
- FIXQP：使用固定 qp 值，在整个编码过程中，所有帧的所有宏块都使用固定的 qp 值，I 帧和 P 帧可以使用不同的固定值；
- QPMAP：在该模式下，用户可以获取到上一帧的编每个宏块的 qp 信息，通过该信息可控制下一帧编码的每个宏块的 qp 信息。用户可通过 VideoEncSetParameter 的接口

VENC_IndexParamMBInfoOutput，开启每一帧的编码信息输出功能，包括每一个宏块的 qp 值，mad 值，sse 值，调用该接口时传递数据结构 VencMBInfo 的指针；通过 VideoEncGetParameter 获取上一帧的整帧的 qp、mad、sse 值，接口为 VENC_IndexParamMBSumInfoOutput 数据结构为 VencMBSumInfo ；用户可通过 VideoEncSetParameter 的接口 VENC_IndexParamMBModeCtrl 设置下一帧的编码细节，传递的数据结构为 VencMBModeCtrl，在 VencMBModeCtrl 数据结构中有指针指向 VencMBModeCtrlInfo 数组，该数组成员包括宏块 qp 值，是否优先使用 skip 预测模式，是否打开该宏块的用户控制使能位，即用户可通过该接口控制每一个宏块的编码模式和 qp 值；

5.2.4. 裁剪编码

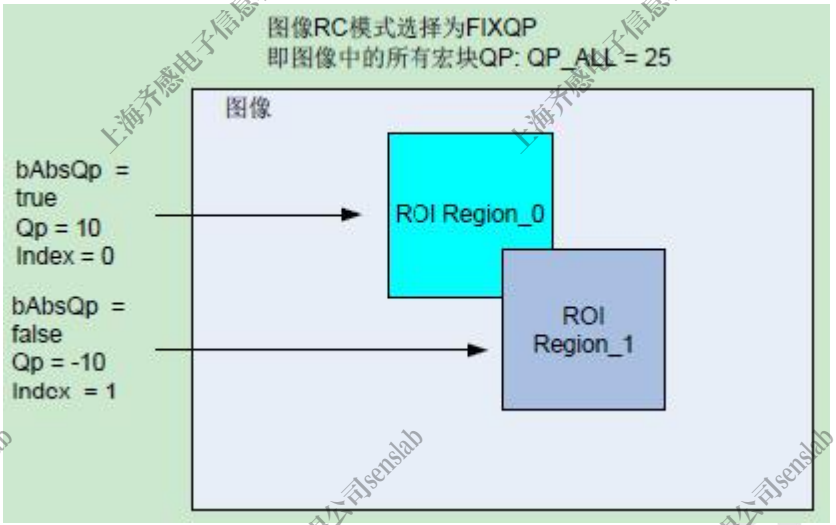
编码器支持从输入源中裁剪一部分进行编码。用户需要设定裁剪区域的起始坐标（x, y），以及裁剪区域的大小（width, height），起始坐标要求 16 对齐，裁剪区域的宽度和高度需要 16 对齐。



5.2.5. ROI 编码

ROI 是（region of interrest）的缩写，即感兴趣区域编码。编码器可以对用户设置的 roi 区域进行特殊编码，roi 的设置包括起始坐标（x,y），区域大小（width，height），以及 QP 设置（QP 设置分为绝对 QP 和相对 QP，可通过 VencROIConfig 数据结构的 roi_abs_flag 使能位进行控制，roi_abs_flag 为 1 表示绝对 QP 模式，直接使用该设定的 QP 值作为整个区域的 QP 值，roi_abs_flag 为 0 表示相对 QP 模式，使用帧级 QP 值加上该设定的 QP 值作为区域的 QP 值，），编码器共支持 8 个该区域的设置，如果其中有区域重叠的话，则重叠区域使用 index 值大的区域的 QP 值，优先级从低到高依次为 0-7。ROI 的起始坐标和宽度高度均需要 16 对齐。

下图示例编码图像采用FixQp模式，设置图像Qp为25，即图像中所有宏块Qp值为25。ROI区域0设置为绝对Qp模式，Qp值为10，索引为0；ROI区域1设置为相对Qp模式，Qp为-10，索引为1。区域0的index小于区域1的index，所以在发生互相重叠的图像区域按高优先级的区域（区域1）Qp设置。区域0除了发生重叠的部分的Qp值等于10。区域1的Qp值为25-10=15。



5.2.6. 非 ROI 区域低帧率

非 ROI 区域低帧率编码，即 ROI 区域按正常帧率编码，非 ROI 区域低帧率编码；用户可根据需要设置非 ROI 区域的编码帧率。

5.2.7. P 帧帧内刷新

P帧刷新ISlice/Intra宏块行，可以为客户提供码流非常平滑的编码方式，每个I帧和P帧的大小可以非常接近。在网络带宽有限（如无线网络）的情况下，降低I帧过大带来的网络冲击，降低网传延时，降低网络传输出错的概率。

5.2.8. osd 叠加

venc 支持 osd 叠加功能，osd 叠加最多可设置 64 个区域，区域之间可以互相重合；osd 叠加包括三种类型，普通叠加功能，将用户指定的 argb 数据叠加到指定的位置上，将用户指定的 argb 数据叠加到指定的位置上并且根据背景的明暗程度反色叠加的图层亮度（即如果背景偏黑色则将图层叠加为白色，如果背景偏白色则将图层叠加为黑色），将用户指定的 yuv 数据填充到用户指定的区域中。

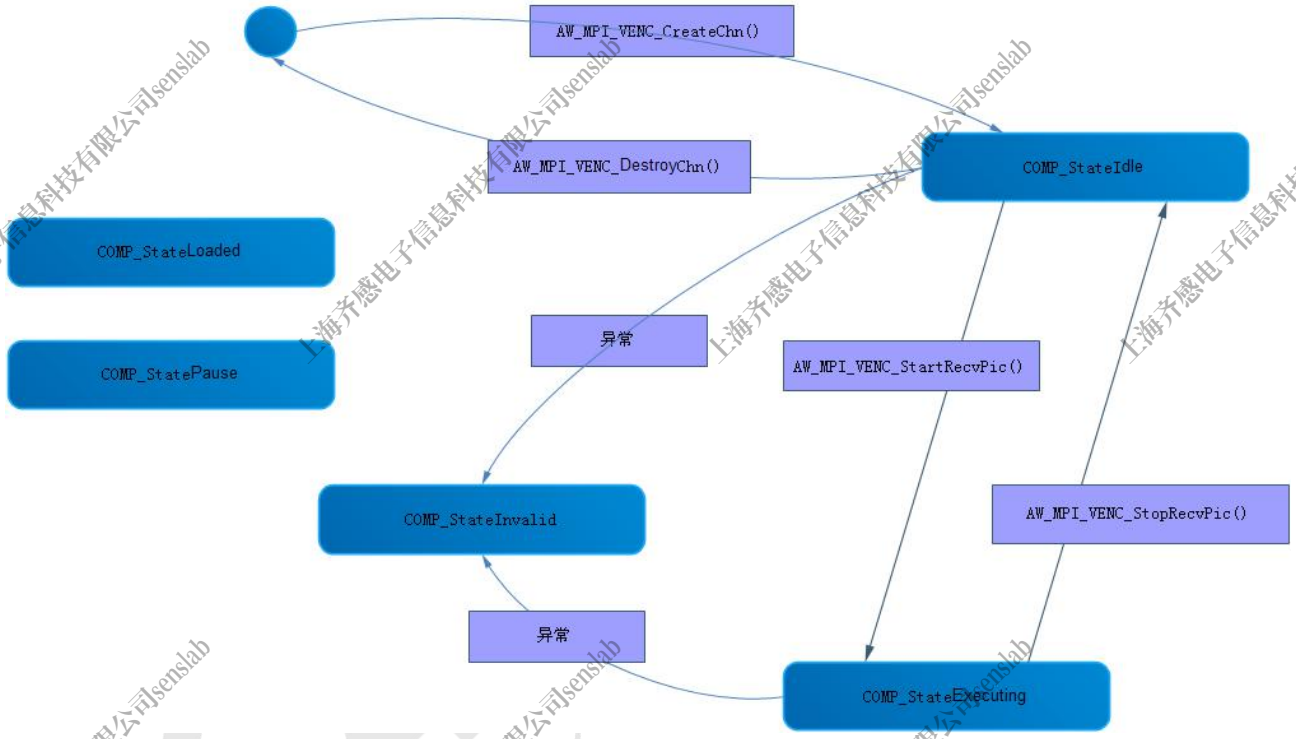
各个区域显示由优先级决定，优先级越大显示越在上层。

5.2.9. 输入数据压缩模式省带宽

venc 支持输入数据为全志自定义的 afbc 的压缩数据格式。afbc 数据可有效节省 dram 带宽，提高编码速度。

5.3. 状态转换与 API 接口

5.3.1. 状态图



Venc 组件内部状态设定为：

- COMP_StateLoaded：组件初始创建状态。
- COMP_StateIdle：组件完成初始化，参数设置、资源配置完毕，随时可以运行的状态。
- COMP_StateExecuting：运行状态。
- COMP_StateInvalid：异常状态。

API AW_MPI_VENC_CreateChn() 的实现过程会经过 COMP_StateLoaded 状态，到达 COMP_StateIdle。

组件内部状态转换的函数是：SendCommand(..., COMP_CommandStateSet, 目标 COMP_State, ...)；

5.3.2. API 和状态

每个 API 只能在允许的状态下调用，如果不在允许的状态下调用 API，则无效。

API 列表如下：(允许被调用的状态栏填写 Y)

	COMP_StateIdle	COMP_StateExecuting	COMP_StateInvalid	说明
AW_MPI_VENC_CreateChn				引起状态转换。创建组件，完成后状态为 COMP_StateIdle
AW_MPI_VENC_DestroyChn	Y			销毁组件
AW_MPI_VENC_SetChnlPriority	Y	Y		
AW_MPI_VENC_GetChnlPriority	Y	Y		
AW_MPI_VENC_ResetChn	Y			重置组件到初始化状态。
AW_MPI_VENC_StartRecvPic	Y			引起状态转换。到 Executing。
AW_MPI_VENC_StartRecvPicEx	Y			引起状态转换。到 Executing，再恢复到 Idle。
AW_VENC_StopRecvPic		Y		引起状态转换。到 Idle。
AW_MPI_VENC_Query		Y		
AW_MPI_VENC_SetChnAttr	Y	Y		设置编码通道属性，只能设置动态属性。
AW_MPI_VENC_GetChnAttr	Y	Y		获取编码通道属性。
AW_MPI_VENC_GetStream	Y	Y		获取编码后的码流。只能用于非绑定模式。
AW_MPI_VEN	Y	Y		归还码流。只能用于非绑定模

C_ReleaseStream				式。
AW_MPI_VENC_InsertUserData	Y	Y		插入用户数据，编码为 SEI 单元。
AW_MPI_VENC_SendFrame	Y	Y		发送待编码的图像。只能用于非绑定模式。
AW_MPI_VENC_SetMaxStreamDuration	Y	Y		设定编码 buffer 的最大缓冲时间。
AW_MPI_VENC_GetMaxStreamDuration	Y	Y		获取编码 buffer 的最大缓冲时间。
AW_MPI_VENC_RequestIDR		Y		立即编码 I 帧。
AW_MPI_VENC_SetRoiCfg	Y	Y		设定 ROI 区域
AW_MPI_VENC_GetRoiCfg	Y	Y		获取 ROI 区域
AW_MPI_VENC_SetH264SliceSplit	Y	Y		设置 slice 分割属性
AW_MPI_VENC_GetH264SliceSplit	Y	Y		获取 slice 分割属性
AW_MPI_VENC_SetH264InterPred	Y	Y		
AW_MPI_VENC_GetH264InterPred	Y	Y		
AW_MPI_VENC_SetH264IntraPred	Y	Y		

AW_MPI_VENC_GetH264IntraPred	Y	Y		
AW_MPI_VENC_SetH264Trans	Y	Y		
AW_MPI_VENC_GetH264Trans	Y	Y		
AW_MPI_VENC_SetH264Entropy	Y	Y		设置熵编码模式
AW_MPI_VENC_GetH264Entropy	Y	Y		获取熵编码模式
AW_MPI_VENC_SetH264Poc	Y	Y		设置 H264 编码 POC 类型。
AW_MPI_VENC_GetH264Poc	Y	Y		获取 H264 编码 POC 类型。
AW_MPI_VENC_SetH264Dblk	Y	Y		设置 H264 编码的 Deblocking 类型
AW_MPI_VENC_GetH264Dblk	Y	Y		获取 H264 编码的 Deblocking 类型
AW_MPI_VENC_SetH264Vui	Y	Y		设置 H264 的 VUI 参数。
AW_MPI_VENC_GetH264Vui	Y	Y		获取 H264 的 VUI 参数。
AW_MPI_VENC_SetJpegParam	Y	Y		设置 JPEG 编码参数。
AW_MPI_VENC_GetJpegParam	Y	Y		获取 JPEG 编码参数。

AW_MPI_VENC_SetMJpegParam	Y	Y		设置 MJPEG 编码参数。
AW_MPI_VENC_GetMJpegParam	Y	Y		获取 MJPEG 编码参数。
AW_MPI_VENC_SetFrameRate	Y			设置编码通道的帧率控制属性。
AW_MPI_VENC_GetFrameRate	Y			获取编码通道的帧率控制属性。
AW_MPI_VENC_SetRcParam	Y	Y		设置编码通道的码率控制属性
AW_MPI_VENC_GetRcParam	Y	Y		获取编码通道的码率控制属性
AW_MPI_VENC_SetRefParam	Y	Y		设置 h264/h265 编码通道高级跳帧参考参数
AW_MPI_VENC_GetRefParam	Y	Y		获取 h264/h265 编码通道高级跳帧参考参数
AW_MPI_VENC_SetColor2Grey	Y	Y		开启或关闭一个通道的彩转灰功能
AW_MPI_VENC_GetColor2Grey	Y	Y		获取一个通道是否开启彩转灰功能
AW_MPI_VENC_SetCrop	Y	Y		
AW_MPI_VENC_GetCrop	Y	Y		
AW_MPI_VENC_SetJpegSnapMode	Y	Y		
AW_MPI_VEN	Y	Y		

C_GetJpegSnapMode				
AW_MPI_VENC_EnableIDR	Y	Y		
AW_MPI_VENC_GetStreamBufInfo	Y	Y		获取码流 buffer 的物理地址和大小。即 VBVBuffer 的整体大小。
AW_MPI_VENC_SetRcPriority	Y	Y		设置码率控制的优先级类型。
AW_MPI_VENC_GetRcPriority	Y	Y		获取码率控制的优先级类型。
AW_MPI_VENC_SetH265SliceSplit	Y	Y		设置 h265 分割属性。如果是按 LCU line 分割，LCU line 最大值：(图像高+63)/64。
AW_MPI_VENC_GetH265SliceSplit	Y	Y		
AW_MPI_VENC_SetH265PredUnit	Y	Y		
AW_MPI_VENC_GetH265PredUnit	Y	Y		
AW_MPI_VENC_SetH265Trans	Y	Y		
AW_MPI_VENC_GetH265Trans	Y	Y		
AW_MPI_VENC_SetH265Entropy	Y	Y		
AW_MPI_VENC_GetH265Entr	Y	Y		

opy				
AW_MPI_VENC_SetH265Dblk	Y	Y		
AW_MPI_VENC_GetH265Dblk	Y	Y		
AW_MPI_VENC_SetH265Sao	Y	Y		
AW_MPI_VENC_GetH265Sao	Y	Y		
AW_MPI_VENC_SetH265Timing	Y	Y		
AW_MPI_VENC_GetH265Timing	Y	Y		
AW_MPI_VENC_SetFrameLostStrategy	Y	Y		
AW_MPI_VENC_GetFrameLostStrategy	Y	Y		
AW_MPI_VENC_SetSuperFrameCfg	Y	Y		
AW_MPI_VENC_GetSuperFrameCfg	Y	Y		
AW_MPI_VENC_SetIntraRefresh	Y	Y		
AW_MPI_VENC_GetIntraRefresh	Y	Y		

5.3.3. API 参考

视频编码模块主要提供视频编码通道的创建和销毁、视频编码通道的复位、开启和停止接收图像、设置和获取编码通道属性、获取和释放码流等功能。

AW_MPI_VENC_CreateChn

【目的】

创建一个编码通道

【语法】

ERRORTYPE AW_MPI_VENC_CreateChn(VENC_CHN VeChn, const VENC_CHN_ATTR_S *pAttr)

【参数】

参数	描述	输入/输出
VeChn	通道 ID 号，范围[0, VENC_MAX_CHN_NUM)	输入
pAttr	编码通道属性	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_venc.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

- 创建后状态为 COMP_StateIdle。

【举例】

无。

AW_MPI_VENC_DestroyChn

【目的】

销毁一个编码通道

【语法】

ERRORTYPE AW_MPI_VENC_DestroyChn(VENC_CHN VeChn)

【参数】

参数	描述	
VeChn	通道 ID 号，范围：[0, VENC_MAX_CHN_NUM)	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_venc.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

AW_MPI_VENC_ResetChn

【目的】

重置编码通道到初始化状态

【语法】

ERRORTYPE AW_MPI_VENC_ResetChn(VENC_CHN VeChn)

【参数】

参数	描述	
VeChn	通道 ID 号。范围：[0, VENC_MAX_CHN_NUM)	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_venc.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

AW_MPI_VENC_StartRecvPic

【目的】

开始编码

【语法】

ERRORTYPE AW_MPI_VENC_StartRecvPic(VENC_CHN VeChn)

【参数】

参数	描述	
VeChn	通道 ID 号。范围：[0, VENC_MAX_CHN_NUM)	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_venc.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

- 引起状态转换，切换到 Executin。

【举例】

无。

AW_MPI_VENC_StartRecvPicEx

【目的】

启动编码，并且编码指定的帧数。

【语法】

ERRORTYPE AW_MPI_VENC_StartRecvPicEx(VENC_CHN VeChn, VENC_RECV_PIC_PARAM_S *pRecvParam);

【参数】

参数	描述	
VeChn	通道 ID 号。范围：[0, VENC_MAX_CHN_NUM)	输入
pRecvParam	指定接收帧数。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_venc.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

- 引起状态转换，切换到 Executing。

【举例】

无。

AW_MPI_VENC_StopRecvPic

【目的】

停止编码

【语法】

ERRORTYPE AW_MPI_VENC_StopRecvPic(VENC_CHN VeChn)

【参数】

参数	描述	
VeChn	编码通道号。范围：[0, VENC_MAX_CHN_NUM)	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_venc.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

- 引起状态转换，切换到 Idle 状态。

【举例】

无。

AW_MPI_VENC_Query

【目的】

查询编码通道状态

【语法】

ERRORTYPE AW_MPI_VENC_Query(VENC_CHN VeChn, VENC_CHN_STAT_S *pStat)

【参数】

参数	描述	
VeChn	编码通道号。范围：[0, VENC_MAX_CHN_NUM)	输入
pStat	状态	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_venc.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

AW_MPI_VENC_RegisterCallback

【目的】

设置编码通道回调函数。

【语法】

ERRORTYPE AW_MPI_VENC_RegisterCallback(VENC_CHN VeChn, MPPCallbackInfo *pCallback)

【参数】

参数	描述	

VeChn	编码通道号。范围：[0, VENC_MAX_CHN_NUM)	输入
pCallback	回调参数	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_venc.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

AW_MPI_VENC_SetChnAttr

【目的】

设置编码通道属性

【语法】

ERRORTYPE AW_MPI_VENC_SetChnAttr(VENC_CHN VeChn, const VENC_CHN_ATTR_S *pAttr)

【参数】

参数	描述	
VeChn	编码通道号。范围：[0, VENC_MAX_CHN_NUM)	输入
pAttr	通道属性	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_venc.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

- 只能设置动态属性。

【举例】

无。

AW_MPI_VENC_GetChnAttr

【目的】

获取编码通道属性

【语法】

ERRORTYPE AW_MPI_VENC_GetChnAttr(VENC_CHN VeChn, VENC_CHN_ATTR_S *pAttr)

【参数】

参数	描述	
VeChn	编码通道号。范围：[0, VENC_MAX_CHN_NUM)	输入
pAttr	通道属性	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_venc.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

AW_MPI_VENC_GetStream

【目的】

获取编码器编码后的码流

【语法】

ERRORTYPE AW_MPI_VENC_GetStream(VENC_CHN VeChn, VENC_STREAM_S *pStream, int nMilliSec)

【参数】

参数	描述	输入 / 输出
VeChn	编码通道号。范围：[0, VENC_MAX_CHN_NUM)	输入
pStream	编码后的码流	输出
nMilliSec	获取数据的超时时间。 -1 表示阻塞模式； 0 表示非阻塞模式； >0 表示阻塞 s32MilliSec 毫秒，超时则报错返回。 取值范围：(0, +∞)	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_venc.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

- 仅限于 venc 组件非绑定模式。与 AW_MPI_VENC_ReleaseStream 必须成对使用，否则编码器 bit stream 内存不会被释放。

【举例】

无。

AW_MPI_VENC_ReleaseStream

【目的】

释放已获取的编码器编码码流（内存）

【语法】

ERRORTYPE AW_MPI_VENC_ReleaseStream(VENC_CHN VeChn, VENC_STREAM_S *pStream)

【参数】

参数	描述	
VeChn	编码通道号。范围：[0, VENC_MAX_CHN_NUM)	输入

pStream	通过 getstream 获取的编码码流	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_venc.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

- 仅限于 venc 组件非绑定模式。与 AW_MPI_VENC_GetStream 必须成对使用，否则编码器 bit stream 内存不会被释放。

【举例】

无。

AW_MPI_VENC_SendFrame

【目的】

向编码器传送待编码图像数据帧

【语法】

ERRORTYPE AW_MPI_VENC_SendFrame(VENC_CHN VeChn, VIDEO_FRAME_INFO_S *pFrame ,int nMilliSec)

【参数】

参数	描述	
VeChn	编码通道号。范围：[0, VENC_MAX_CHN_NUM)	输入
pFrame	图像数据帧	输入
nMilliSec	发送数据的超时时间。 -1 表示阻塞模式； 0 表示非阻塞模式； >0 表示阻塞 s32MilliSec 毫秒，超时则报错返回。	输入

【返回值】

返回值	描述
0	成功

非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_venc.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

- 仅限 venc 组件非绑定模式使用。

【举例】

无。

AW_MPI_VENC_RequestIDR

【目的】

立即编码 I 帧

【语法】

ERRORTYPE AW_MPI_VENC_RequestIDR(VENC_CHN VeChn, BOOL bInstant)

【参数】

参数	描述	
VeChn	编码通道号。范围：[0, VENC_MAX_CHN_NUM)	输入
bInstant	未使用。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_venc.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

AW_MPI_VENC_GetHandle

【目的】

获取编码器编码管道文件句柄

【语法】

int AW_MPI_VENC_GetHandle (VENC_CHN VeChn)

【参数】

参数	描述	
VeChn	编码通道号，范围：[0, VENC_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
>= 0	句柄编号
< 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_venc.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

AW_MPI_VENC_SetRoiCfg

【目的】

设置编码器感兴趣区域

【语法】

ERRORTYPE AW_MPI_VENC_SetRoiCfg(VENC_CHN VeChn, VENC_ROI_CFG_S *pVencRoiCfg)

【参数】

参数	描述	输入/输出
VeChn	编码通道号。范围：[0, VENC_MAX_CHN_NUM)。	输入
pVencRoiCfg	感兴趣区域	输入

【返回值】

返回值	描述

0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_venc.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

AW_MPI_VENC_GetRoiCfg

【目的】

获取编码器感兴趣区域

【语法】

ERRORTYPE AW_MPI_VENC_GetRoiCfg(VENC_CHN VeChn, unsigned int nIndex, VENC_ROI_CFG_S *pVencRoiCfg)

【参数】

参数	描述	
VeChn	编码通道号，范围：[0, VENC_MAX_CHN_NUM)。	输入
pVencRoiCfg	感兴趣区域	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_venc.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

AW_MPI_VENC_GetH264SpsPpsInfo

【目的】

获取编码器 H264 编码的 spspps 头信息

【语法】

ERRORTYPE AW_MPI_VENC_GetH264SpsPpsInfo(VENC_CHN VeChn, VencHeaderData*pH264SpsPpsInfo)

【参数】

参数	描述	
VeChn	编码通道号，范围：[0, VENC_MAX_CHN_NUM)。	输入
pH264SpsPpsInfo	spspps 信息	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_venc.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

AW_MPI_VENC_GetH265SpsPpsInfo

【目的】

获取编码器 H265 编码的 spspps 头信息

【语法】

ERRORTYPE AW_MPI_VENC_GetH265SpsPpsInfo(VENC_CHN VeChn, VencHeaderData *pH264SpsPpsInfo)

【参数】

参数	描述	输入/输出

VeChn	编码通道号，范围：[0, VENC_MAX_CHN_NUM)。	输入
pH265SpsPpsInfo	spspps 信息	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_venc.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

AW_MPI_VENC_SetJpegParam

【目的】

设置 JPEG 协议编码通道的高级参数。

【语法】

ERRORTYPE AW_MPI_VENC_SetJpegParam(VENC_CHN VeChn, const VENC_PARAM_JPEG_S *pJpegParam)

【参数】

参数	描述	
VeChn	编码通道号，范围：[0, VENC_MAX_CHN_NUM)。	输入
pJpegParam	jpeg 编码参数	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_venc.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

AW_MPI_VENC_GetJpegParam

【目的】

获取编码器 jpeg 编码参数

【语法】

ERRORTYPE AW_MPI_VENC_SetJpegParam(VENC_CHN VeChn, const VENC_PARAM_JPEG_S *pJpegParam)

【参数】

参数	描述	
VeChn	编码通道号，范围：[0, VENC_MAX_CHN_NUM)。	输入
pJpegParam	jpeg 编码参数	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_venc.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

AW_MPI_VENC_SetJpegExifInfo

【目的】

设置编码器 jpegExif 编码参数

【语法】

ERRORTYPE AW_MPI_VENC_SetJpegExifInfo(VENC_CHN VeChn, const VENC_EXIFINFO_S*pJpegExifInfo)

【参数】

参数	描述	
VeChn	编码通道号，范围：[0, VENC_MAX_CHN_NUM)。	输入
pJpegExifInfo	jpegExif 编码参数	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_venc.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

AW_MPI_VENC_GetJpegExifInfo

【目的】

获取编码器 jpegExif 编码参数

【语法】

ERRORTYPE AW_MPI_VENC_SetJpegExifInfo(VENC_CHN VeChn, const VENC_EXIFINFO_S*pJpegExifInfo)

【参数】

参数	描述	
VeChn	编码通道号，范围：[0, VENC_MAX_CHN_NUM)。	输入
pJpegExifInfo	jpegExif 编码参数	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_venc.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

AW_MPI_VENC_GetJpegThumbBuffer

【目的】

获取编码器 jpeg 缩略图编码 buffer

【语法】

ERRORTYPE AW_MPI_VENC_GetJpegThumbBuffer(VENC_CHN VeChn, VENC_JPEG_THUMB_BUFFER_S *pThumbBuffer)

【参数】

参数	描述	
VeChn	编码通道号，范围：[0, VENC_MAX_CHN_NUM)。	输入
pThumbBuffer	缩略图 buffer	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_venc.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

AW_MPI_VENC_SetFrameRate

【目的】

设置编码通道帧率控制属性

【语法】

ERRORTYPE AW_MPI_VENC_SetFrameRate(VENC_CHN VeChn, const VENC_FRAME_RATE_S *pFrameRate)

【参数】

参数	描述	
VeChn	编码通道号，范围：[0, VENC_MAX_CHN_NUM)。	输入
pFrameRate	帧率属性	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_venc.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

AW_MPI_VENC_GetFrameRate

【目的】

获取编码通道帧率控制属性

【语法】

ERRORTYPE AW_MPI_VENC_GetFrameRate(VENC_CHN VeChn, const VENC_FRAME_RATE_S *pFrameRate)

【参数】

参数	描述	
VeChn	编码通道号，范围：[0, VENC_MAX_CHN_NUM)。	输入
pFrameRate	帧率属性	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_venc.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

AW_MPI_VENC_SetTimeLapse

【目的】

设置编码通道缩时/慢摄影状态下，用于编码的帧间隔时间。

【语法】

ERRORTYPE AW_MPI_VENC_SetTimeLapse(VENC_CHN VeChn, int64_t nTimeLapse)

【参数】

参数	描述	输入/输出
VeChn	编码通道号，范围：[0, VENC_MAX_CHN_NUM)。	输入
nTimeLapse	编码帧的间隔时间，单位微秒。设置后，编码通道根据编码帧间隔时间选定输入帧进行编码，编码后的输出帧的 PTS 根据编码通道目标输出帧率重新设置，从 0 开始，以确保录制的文件播放时按照输出帧率进行播放。 静态属性。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_venc.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

AW_MPI_VENC_GetTimeLapse

【目的】

获取编码通道缩时/慢摄影帧间隔时间

【语法】

ERRORTYPE AW_MPI_VENC_GetTimeLapse(VENC_CHN VeChn, int64_t *pTimeLapse)

【参数】

参数	描述	
VeChn	编码通道号，范围：[0, VENC_MAX_CHN_NUM)。	输入
pTimeLapse	编码帧的间隔时间，单位微秒。设置后，编码通道根据编码帧间隔时间选定输入帧进行编码，编码后的输出帧的 PTS 根据编码通道目标输出帧率重新设置，从 0 开始，以确保录制的文件播放时按照输出帧率进行播放。	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_venc.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

AW_MPI_VENC_SetColor2Grey

【目的】

开启或关闭一个通道的彩转灰功能。

【语法】

ERRORTYPE AW_MPI_VENC_SetColor2Grey(VENC_CHN VeChn, const

VENC_COLOR2GREY_S *pChnColor2Grey);

【参数】

参数	描述	
VeChn	编码通道号，范围：[0, VENC_MAX_CHN_NUM)。	输入
pChnColor2Grey	彩转灰配置信息。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

【注意】

无。

【举例】

无。

AW_MPI_VENC_GetColor2Grey

【目的】

获取一个通道是否开启彩转灰功能。

【语法】

ERRORTYPE AW_MPI_VENC_GetColor2Grey(VENC_CHN VeChn, VENC_COLOR2GREY_S *pChnColor2Grey);

【参数】

参数	描述	
VeChn	编码通道号，范围：[0, VENC_MAX_CHN_NUM)。	输入
pChnColor2Grey	获取开启或关闭彩转灰功能的参数。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

【注意】

无。

【举例】

无。

AW_MPI_VENC_SetCrop

【目的】

设置编码通道编码裁剪区域

【语法】

ERRORTYPE AW_MPI_VENC_SetCrop(VENC_CHN VeChn, const VENC_CROP_CFG_S *pCropCfg)

【参数】

参数	描述	
VeChn	编码通道号，范围：[0, VENC_MAX_CHN_NUM)。	输入
pCropCfg	裁剪区域。 动态属性。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_venc.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

AW_MPI_VENC_GetCrop

【目的】

获取编码通道编码裁剪区域

【语法】

ERRORTYPE AW_MPI_VENC_GetCrop(VENC_CHN VeChn, VENC_CROP_CFG_S *pCropCfg)

【参数】

参数	描述	
VeChn	编码通道号，范围：[0, VENC_MAX_CHN_NUM)。	输入
pCropCfg	裁剪区域	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_venc.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

AW_MPI_VENC_GetStreamBufInfo

【目的】

获取码流 buffer 的物理地址和大小。即 VBVBuffer 的整体大小

【语法】

ERRORTYPE AW_MPI_VENC_GetStreamBufInfo(VENC_CHN VeChn, VENC_STREAM_BUF_INFO_S*pStreamBufInfo)

【参数】

参数	描述	
VeChn	编码通道号，范围：[0, VENC_MAX_CHN_NUM)。	输入
pStreamBufInfo	vbvBuffer 信息	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_venc.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

AW_MPI_VENC_SetIntraRefresh

【目的】

P 帧帧内刷新。设置刷 I 宏块的参数。

【语法】

ERRORTYPE AW_MPI_VENC_SetIntraRefresh(VENC_CHN VeChn, VENC_PARAM_INTRA_REFRESH_S *pIntraRefresh)

【参数】

参数	描述	
VeChn	编码通道号，范围：[0, VENC_MAX_CHN_NUM)。	输入
pIntraRefresh	P 帧帧内刷新，刷 I 宏块的设置参数。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_venc.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

AW_MPI_VENC_GetIntraRefresh

【目的】

【语法】

ERRORTYPE AW_MPI_VENC_SetIntraRefresh(VENC_CHN VeChn, VENC_PARAM_INTRA_REFRESH_S *pIntraRefresh)

【参数】

参数	描述	
VeChn	编码通道号，范围：[0, VENC_MAX_CHN_NUM)。	输入
pIntraRefresh	P 帧帧内刷新，刷 I 宏块的设置参数。	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_venc.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

AW_MPI_VENC_SetSmartP

【目的】

P 帧 smart 编码参数。

【语法】

ERRORTYPE AW_MPI_VENC_SetSmartP(VENC_CHN VeChn, VencSmartFun *pSmartPParam)

【参数】

参数	描述	
VeChn	编码通道号，范围：[0, VENC_MAX_CHN_NUM)。	输入
pSmartPParam	smart 编码参数。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

【注意】

无。

【举例】

无。

AW_MPI_VENC_GetSmartP

【目的】

获取 smart 编码参数。

【语法】

ERRORTYPE AW_MPI_VENC_GetSmartP(VENC_CHN VeChn, VencSmartFun *pSmartPParam)

【参数】

参数	描述	输入/输出
VeChn	编码通道号，范围：[0, VENC_MAX_CHN_NUM)。	输入
pSmartPParam	smart 编码参数。	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

【注意】

无。

【举例】

无。

AW_MPI_VENC_SetBrightness

【目的】

配置 h264 和 h265 编码的亮暗阈值属性，和 smart 编码配合使用。

【语法】

ERRORTYPE AW_MPI_VENC_SetBrightness(VENC_CHN VeChn, VencBrightnessS *pBrightness);

【参数】

参数	描述	

VeChn	编码通道号，范围：[0, VENC_MAX_CHN_NUM)。	输入
pBrightness	亮暗阈值属性。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

【注意】

无。

【举例】

无。

AW_MPI_VENC_GetBrightness

【目的】

获取亮暗阈值属性。

【语法】

ERRORTYPE AW_MPI_VENC_GetBrightness(VENC_CHN VeChn, VencBrightnessS *pBrightness)

【参数】

参数	描述	输入/输出
VeChn	编码通道号，范围：[0, VENC_MAX_CHN_NUM)。	输入
pSmartPParam	亮暗阈值属性。	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

【注意】

无。

【举例】

无。

AW_MPI_VENC_SetVEFreq

【目的】

设置编码引擎时钟频率。

【语法】

ERRORTYPE AW_MPI_VENC_SetVEFreq(VENC_CHN VeChn, int nFreq); //nFreq: MHz;

【参数】

参数	描述	输入/输出
VeChn	编码通道号，范围：[0, VENC_MAX_CHN_NUM)。	输入
nFreq	引擎时钟频率，单位 MHz。范围：[480, 532, 600]，默认 480，推荐 532。 动态属性。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

【注意】

无。

【举例】

无。

AW_MPI_VENC_Set3DNR

【目的】

设置 3D 降噪。

【语法】

ERRORTYPE AW_MPI_VENC_Set3DNR(VENC_CHN VeChn, int b3DNRFlag);

【参数】

参数	描述	输入/输出
VeChn	编码通道号，范围：[0, VENC_MAX_CHN_NUM)。	输入
b3DNRFlag	3d 降噪等级。范围：[0, 6]。推荐值 3，数值越大降噪效果越强。 动态属性。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

【注意】

无。

【举例】

无。

AW_MPI_VENC_Get3DNR

【目的】

获取 3D 降噪等级。

【语法】

ERRORTYPE AW_MPI_VENC_Get3DNR(VENC_CHN VeChn, int *b3DNRFlag);

【参数】

参数	描述	输入/输出
VeChn	编码通道号，范围：[0, VENC_MAX_CHN_NUM)。	输入
b3DNRFlag	3d 降噪等级。范围：[0, 6]。推荐值 3，数值越大降噪效果越强。 动态属性。	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

【注意】

无。

【举例】

无。

AW_MPI_VENC_GetCacheState

【目的】

获取视频编码库缓冲状态。

【语法】

ERRORTYPE AW_MPI_VENC_GetCacheState(VENC_CHN VeChn, CacheState *pCacheState);

【参数】

参数	描述	输入/输出
VeChn	编码通道号，范围：[0, VENC_MAX_CHN_NUM)。	输入
pCacheState	视频编码库缓冲状态。 动态属性。	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

【注意】

无。

【举例】

无。

AW_MPI_VENC_SetRefParam

【目的】

设置编码高级跳帧参考。

【语法】

ERRORTYPE AW_MPI_VENC_SetRefParam(VENC_CHN VeChn, const VENC_PARAM_REF_S *pstRefParam);

【参数】

参数	描述	输入/输出
VeChn	编码通道号，范围：[0, VENC_MAX_CHN_NUM)。	输入
pstRefParam	高级跳帧参考参数。 静态属性。	输入

【返回值】

返回值	描述
0	成功

非 0	失败，其值见错误码。

【需求】

【注意】

无。

【举例】

无。

AW_MPI_VENC_GetRefParam

【目的】

获取编码高级跳帧参考参数。

【语法】

ERRORTYPE AW_MPI_VENC_GetRefParam(VENC_CHN VeChn, VENC_PARAM_REF_S *pstRefParam);

【参数】

参数	描述	输入/输出
VeChn	编码通道号，范围：[0, VENC_MAX_CHN_NUM)。	输入
pstRefParam	高级跳帧参考参数。 静态属性。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

【注意】

无。

【举例】

无。

AW_MPI_VENC_SetHorizonFlip

【目的】

设置编码水平镜像。

【语法】

ERRORTYPE AW_MPI_VENC_SetHorizonFlip(VENC_CHN VeChn, BOOL bHorizonFlipFlag);

【参数】

参数	描述	输入/输出
VeChn	编码通道号，范围：[0, VENC_MAX_CHN_NUM)。	输入
bHorizonFli pFlag	是否水平镜像。 动态属性。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

【注意】

无。

【举例】

无。

AW_MPI_VENC_GetHorizonFlip

【目的】

获取编码水平镜像。

【语法】

ERRORTYPE AW_MPI_VENC_GetHorizonFlip(VENC_CHN VeChn, BOOL *bpHorizonFlipFlag);

【参数】

参数	描述	输入/输出
VeChn	编码通道号，范围：[0, VENC_MAX_CHN_NUM)。	输入
bHorizonFli pFlag	是否水平镜像。 动态属性。	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

【注意】

无。

【举例】

无。

AW_MPI_VENC_SetAdaptiveIntraInP

【目的】

设置自适应调整 P 帧帧内预测等级属性。

【语法】

ERRORTYPE AW_MPI_VENC_SetAdaptiveIntraInP(VENC_CHN VeChn, BOOL bAdaptiveIntraInPFlag);

【参数】

参数	描述	输入/输出
VeChn	编码通道号，范围：[0, VENC_MAX_CHN_NUM)。	输入
bAdaptiveIntraInPFlag	是否打开自适应调整 P 帧帧内预测等级属性功能。 静态属性。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

【注意】

无。

【举例】

无。

AW_MPI_VENC_EnableNullSkip

【目的】

打开插空帧功能，在编码帧率不够的情况下，插入空帧达到预期帧率。

【语法】

ERRORTYPE AW_MPI_VENC_EnableNullSkip(VENC_CHN VeChn, BOOL bEnable);

【参数】

参数	描述	输入/输出

VeChn	编码通道号，范围：[0, VENC_MAX_CHN_NUM)。	输入
bEnable	打开插空帧功能。 静态属性。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

【注意】

无。

【举例】

无。

AW_MPI_VENC_EnablePSkip

【目的】

打开插 skip-P 帧功能，在编码帧率不够的情况下，插入 skip-P 帧达到预期帧率。

【语法】

ERRORTYPE AW_MPI_VENC_EnablePSkip(VENC_CHN VeChn, BOOL bEnable);

【参数】

参数	描述	输入/输出
VeChn	编码通道号，范围：[0, VENC_MAX_CHN_NUM)。	输入
bEnable	打开插 skip-P 帧功能。 静态属性。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

【注意】

无。

【举例】

无。

AW_MPI_VENC_SaveBsFile

【目的】

设置编码库保存码流。

【语法】

ERRORTYPE AW_MPI_VENC_SaveBsFile(VENC_CHN VeChn, VencSaveBSFile *pSaveParam);

【参数】

参数	描述	输入/输出
VeChn	编码通道号，范围：[0, VENC_MAX_CHN_NUM)。	输入
pSaveParam	设置参数决定保存码流的细节。 动态属性。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

【注意】

无。

【举例】

无。

AW_MPI_VENC_SetProcSet

【目的】

设置编码库 proc 信息的配置。

【语法】

ERRORTYPE AW_MPI_VENC_SetProcSet(VENC_CHN VeChn, VeProcSet *pVeProcSet);

【参数】

参数	描述	输入/输出
VeChn	编码通道号，范围：[0, VENC_MAX_CHN_NUM)。	输入
pVeProcSet	proc 信息配置属性。 动态属性。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

【注意】

无。

【举例】

无。

5.4. 数据结构说明

H264E_NALU_TYPE_E

【说明】

定义 H264 码流 NALU 类型。

【定义】

```
typedef enum H264E_NALU_TYPE_E
{
    H264E_NALU_PSLICE = 1,          /*PSLICE types*/
    H264E_NALU_ISLICE = 5,          /*ISLICE types*/
    H264E_NALU_SEI    = 6,          /*SEI types*/
    H264E_NALU_SPS    = 7,          /*SPS types*/
    H264E_NALU_PPS    = 8,          /*PPS types*/
    H264E_NALU_IPSLICE = 9,
    H264E_NALU_BUTT
} H264E_NALU_TYPE_E;
```

【成员】

成员名称	描述
H264E_NALU_PSLICE	码流为 P 帧类型。
H264E_NALU_ISLICE	码流为 I 帧类型。
H264E_NALU_SE	暂未使用。

I	
H264E_NALU_SPS	暂未使用。
H264E_NALU_PPS	暂未使用。
H264E_NALU_IPSLICE	暂未使用

【注意事项】

【相关数据类型及接口】

JPEGE_PACK_TYPE_E

【说明】

定义 JPEG 码流的 PACK 类型。

【定义】

```
typedef enum JPEGE_PACK_TYPE_E
{
    JPEGE_PACK_ECS = 5,                 /*ECS types*/
    JPEGE_PACK_APP = 6,                 /*APP types*/
    JPEGE_PACK_VDO = 7,                 /*VDO types*/
    JPEGE_PACK_PIC = 8,                 /*PIC types*/
    JPEGE_PACK_BUTT
} JPEGE_PACK_TYPE_E;
```

【成员】

成员名称	描述
JPEGE_PACK_ECS	未使用
JPEGE_PACK_APP	未使用
JPEGE_PACK_VDO	未使用
JPEGE_PACK_PIC	码流为完整的 JPEG 格式图片。

【注意事项】

【相关数据类型及接口】

H265E_NALU_TYPE_E

【说明】

定义 H265 码流 NALU 类型。

【定义】

```
typedef enum H265E_NALU_TYPE_E
{
    H265E_NALU_PSLICE = 1,          /*P SLICE types*/
    H265E_NALU_ISLICE = 19,         /*I SLICE types*/
    H265E_NALU_VPS    = 32,         /*VPS types*/
    H265E_NALU_SPS    = 33,         /*SPS types*/
    H265E_NALU_PPS    = 34,         /*PPS types*/
    H265E_NALU_SEI    = 39,         /*SEI types*/

    H265E_NALU_BUTT
} H265E_NALU_TYPE_E;
```

【成员】

成员名称	描述
H265E_NALU_PSLICE	码流为 P 帧类型。
H265E_NALU_ISLICE	码流为 I 帧类型。
H265E_NALU_VPS	暂未使用。
H265E_NALU_SPS	暂未使用。
H265E_NALU_PPS	暂未使用。
H265E_NALU_SEI	暂未使用

【注意事项】

【相关数据类型及接口】

VENC_DATA_TYPE_U

【说明】

编码输出的码流类型。

【定义】

```
typedef union VENC_DATA_TYPE_U
{
    H264E_NALU_TYPE_E      enH264EType;          /*H264E NALU types*/
    JPEGE_PACK_TYPE_E      enJPEGEType;          /*JPEGE pack types*/
    MPEG4E_PACK_TYPE_E     enMPEG4EType;         /*MPEG4E pack types*/
    H265E_NALU_TYPE_E      enH265EType;          /*H264E NALU types*/
}VENC_DATA_TYPE_U;
```

【成员】

成员名称	描述
enH264EType	H264 码流包类型
enJPEGEType	JPEG 码流包类型。
enMPEG4EType	MPEG4 码流包类型。不支持。
enH265EType	H265 码流包类型

【注意事项】

【相关数据类型及接口】

VENC_PACK_S

【说明】

定义视频编码包输出属性结构。

【定义】

```
typedef struct VENC_PACK_S
{
    //unsigned int     mPhyAddr;          /*the physics address of stream*/
    unsigned char     *mpAddr0;          /*the virtual address of stream*/
    unsigned char     *mpAddr1;
    unsigned int      mLen0;             /*the length of stream*/
    unsigned int      mLen1;

    uint64_t      mPTS;                  /*PTS*/
```

```
    BOOL    mbFrameEnd;             /*frame end? */

    VENC_DATA_TYPE_U    mDataType;          /*the type of stream*/
    unsigned int    mOffset;

    unsigned int mDataNum;
    VENC_PACK_INFO_S mPackInfo[8];
}VENC_PACK_S;
```

【成员】

成员名称	描述
mpAddr0	编码数据 buf0 部份地址（虚拟地址）
mpAddr1	编码数据 buf1 部份地址（虚拟地址）
mLen0	编码数据 buf0 部份长度
mLen1	编码数据 buf1 部份长度
mPTS	编码数据时间戳，单位微秒。
mbFrameEnd	帧结束标识。范围：[TRUE]，一定为 TRUE。
mDataType	数据流类型
mOffset	码流包有效数据和码流包首地址的偏移。一般为 0。
mDataNum	未使用。
mPackInfo[8]	未使用。

【注意事项】

buf1 部份不一定有

【相关数据类型及接口】

VENC_STREAM_S

【说明】

编码通道编码输出流结构。

【定义】

```
typedef struct VENC_STREAM_S
{
    VENC_PACK_S *mpPack;            /*stream pack attribute*/
    unsigned int        mPackCount;     /*the pack number of one frame stream*/
    unsigned int        mSeq;           /*the list number of stream*/
```

```
    union
    {
        VENC_STREAM_INFO_H264_S    mH264Info;       /*the stream info of h264*/
        VENC_STREAM_INFO_JPEG_S    mJpegInfo;       /*the stream info of jpeg*/
        VENC_STREAM_INFO_MPEG4_S mMpeg4Info;     /*the stream info of mpeg4*/
        VENC_STREAM_INFO_H265_S    mH265Info;       /*the stream info of h265*/
    };
}VENC_STREAM_S;
```

【成员】

成员名称	描述
mpPack	数据包数组，当前只填充第一个元素。
mPackCount	数据包个数，取值范围：[1]，当前只支持一次一个码流包，一个码流包就是一帧的模式。
mSeq	编码库内部装载该帧的 buffer 的 id 号。取值范围：[0,255]。
mH264Info	H264 编码包信息，未使用。
mJpegInfo	Jpeg 编码包信息，未使用。
mMpeg4Info	Mpeg4 编码包信息，未使用。
mH265Info	H265 编码包信息，未使用。

【注意事项】

【相关数据类型及接口】

VENC_ATTR_H264_S

【说明】

编码通道 H264 编码属性结构。

【定义】

```
typedef struct VENC_ATTR_H264_S
{
    unsigned int    MaxPicWidth;
    unsigned int    MaxPicHeight;

    unsigned int    BufSize;
    unsigned int    Profile;
```

```
    BOOL bByFrame;

    unsigned int    PicWidth;
    unsigned int    PicHeight;

    unsigned int    BFrameNum;
    unsigned int    RefNum;

    BOOL mbLongTermRef;
    BOOL FastEncFlag;
    int IQpOffset;
    int mVirtualIFrameInterval;
    BOOL mbPIntraEnable;
}VENC_ATTR_H264_S;
```

【成员】

成员名称	描述
MaxPicWidth	编码通道最大输出图片宽度，范围[192, 4096]。 静态属性。
MaxPicHeight	编码通道最大输出图片高度，范围[96, 4096]。 静态属性。
BufSize	编码器输出 buffer 的长度，范围[0,16*1024*1024]，0 表示系统自行决定输出 buffer 的长度。 静态属性。
Profile	profile 类型。0: baseline; 1:MP; 2:HP; 静态属性。
bByFrame	输入流模式：frame or filed 静态属性。
PicWidth	编码通道图片编码输出宽度。在满足 MaxPicWidth 范围的前提下，在当前输入图片宽度（VENC_ATTR_S::SrcPicWidth）的[1/4, 4]倍之间任意取值(必须是偶数)。 静态属性。
PicHeight	编码通道图片编码输出高度。在满足 MaxPicWidth 范围的前提下，在当前输入图片高度（VENC_ATTR_S::SrcPicHeight）的[1/4, 4]倍之间任意取值(必须是偶数)。

	静态属性。
BFrameNum	B 帧的数量（0 表示不支持 B 帧）。范围[0]，不支持 B 帧编码。 静态属性。
RefNum	参考帧数 （defaul: 0）。范围[0]，不支持配置参考帧数。 静态属性。
mbLongTermRef	开启长期参考帧。TRUE：开启。FALSE：关闭。 静态属性。
FastEncFlag	开启快速编码。开启后编码器会通过降低编码质量来提高编码速度。 静态属性。
IQpOffset	单独降低 I 帧编码质量，从而减少 I 帧的体积，缩小 I 帧和 P 帧的体积差距，便于网络平滑传输。范围[0,10)。建议设置为 0，如果超过 6 则 I 帧质量大幅下降。 静态属性。
mVirtualIFrameInterval	虚拟 I 帧间隔。编码时可以用 P 帧作为虚拟 I 帧，进一步降低码率。范围：[0, VENC_ATTR_S::MaxKeyInterval)，必须小于 I 帧间隔。 静态属性。
mbPIntraEnable	开启 P 帧帧内预测，提高编码质量。 静态属性。

【注意事项】

【相关数据类型及接口】

VENC_ATTR_H265_S

【说明】

编码通道 H265 编码属性结构。

【定义】

```
typedef struct VENC_ATTR_H265_S
{
    unsigned int    mMaxPicWidth;
    unsigned int    mMaxPicHeight;

    unsigned int    mBufSize;
    unsigned int    mProfile;
    BOOL mbByFrame;
```

```
    unsigned int    mPicWidth;
    unsigned int    mPicHeight;

    unsigned int    mBFrameNum;
    unsigned int    mRefNum;

    BOOL mbLongTermRef; //enable long term reference IDR Frame
    BOOL mFastEncFlag;      //for fast video encoder
    int IQpOffset;    //IQp offset value to offset I frame Qp to decrease I frame size.
    int mVirtualIFrameInterval;        /* (5, IDR_frame_interval), dynamic param */
    BOOL mbPIntraEnable;    //enalbe p frame intra
}VENC_ATTR_H265_S;
```

【成员】

成员名称	描述
mMaxPicWidth	编码通道最大输出图片宽度，范围[192, 4096]。 静态属性。
mMaxPicHeight	编码通道最大输出图片高度，范围[96, 4096]。 静态属性。
mBufSize	编码器输出 buffer 的长度，范围[0,16*1024*1024]，0 表示系统自行决定输出 buffer 的长度。 静态属性。
mProfile	profile 类型。0: baseline; 1:MP; 静态属性。
mbByFrame	输入流模式：frame or filed。范围 TRUE。只支持 frame 模式。 静态属性。
mPicWidth	编码通道图片编码输出宽度。在满足 MaxPicWidth 范围的前提下，在当前输入图片宽度（VENC_ATTR_S::SrcPicWidth）的[1/4, 4]倍之间任意取值(必须是偶数)。 静态属性。
mPicHeight	编码通道图片编码输出高度。在满足 MaxPicWidth 范围的前提下，在当前输入图片高度（VENC_ATTR_S::SrcPicHeight）的[1/4, 4]倍之间任意取值(必须是偶数)。 静态属性。

mBFrameNum	B 帧的数量。范围：[0]。不支持编码 B 帧。 静态属性。
mRefNum	参考帧数。范围：[0]。不支持配置参考帧数。 静态属性。
mbLongTermRef	开启长期参考帧。TRUE：开启。FALSE：关闭。 静态属性。
mFastEncFlag	开启快速编码。开启后编码器会通过降低编码质量来提高编码速度。 静态属性。
IQpOffset	单独降低 I 帧编码质量，从而减少 I 帧的体积，缩小 I 帧和 P 帧的体积差距，便于网络平滑传输。范围[0,10)。建议设置为 0，如果超过 6 则 I 帧质量大幅下降。 静态属性。
mVirtualIFrameInterval	虚拟 I 帧间隔。编码时可以用 P 帧作为虚拟 I 帧，进一步降低码率。范围：[0, VENC_ATTR_S::MaxKeyInterval)，必须小于 I 帧间隔。 静态属性。
mbPIntraEnable	开启 P 帧帧内预测，提高编码质量。 静态属性。

【注意事项】

【相关数据类型及接口】

VENC_ATTR_MJPEG_S

【说明】

编码通道 MJPEG 编码属性结构。

【定义】

```
typedef struct VENC_ATTR_MJPEG_S
{
    unsigned int    mMaxPicWidth;
    unsigned int    mMaxPicHeight;
    unsigned int    mBufSize;
    BOOL mbByFrame;

    unsigned int    mPicWidth;
    unsigned int    mPicHeight;
```

}VENC_ATTR_MJPEG_S;

【成员】

成员名称	描述
MaxPicWidth	编码通道最大输出图片宽度，范围[192, 4096]。 静态属性。
MaxPicHeight	编码通道最大输出图片高度，范围[96, 4096]。 静态属性。
BufSize	编码器输出 buffer 的长度，范围[0,16*1024*1024]，0 表示系统自行决定输出 buffer 的长度。 静态属性。
bByFrame	输入流模式：frame or filed。 静态属性。
PicWidth	编码通道图片编码输出宽度。在满足 MaxPicWidth 范围的前提下，在当前输入图片宽度（VENC_ATTR_S::SrcPicWidth）的[1/4, 4]倍之间任意取值(必须是偶数)。 静态属性。
PicHeight	编码通道图片编码输出高度。在满足 MaxPicWidth 范围的前提下，在当前输入图片高度（VENC_ATTR_S::SrcPicHeight）的[1/4, 4]倍之间任意取值(必须是偶数)。 静态属性。

【注意事项】

【相关数据类型及接口】

VENC_ATTR_JPEG_S

【说明】

编码通道 JPEG 编码属性结构。

【定义】

```
typedef struct VENC_ATTR_JPEG_S
{
    unsigned int    MaxPicWidth;    /*maximum width of a picture to be encoded, in pixel*/
    unsigned int    MaxPicHeight;   /*maximum height of a picture to be encoded, in pixel*/
    unsigned int    BufSize;        /*stream buffer size, 0:adaptive to bitRate */
```

```
    BOOL bByFrame;          /*get stream mode is field mode  or frame mode*/

    unsigned int   PicWidth;        /*width of a picture to be encoded, in pixel*/
    unsigned int   PicHeight;       /*height of a picture to be encoded, in pixel*/
    BOOL   bSupportDCF;             /*support dcf*/
}VENC_ATTR_JPEG_S;
```

【成员】

成员名称	描述
MaxPicWidth	编码通道最大输出图片宽度，范围[192, 4096]。 静态属性。
MaxPicHeight	编码通道最大输出图片高度，范围[96, 4096]。 静态属性。
BufSize	编码器输出 buffer 的长度，范围[0,16*1024*1024]，0 表示系统自行决定输出 buffer 的长度。 静态属性。
bByFrame	输入流模式：frame or filed。 静态属性。
PicWidth	编码通道图片编码输出宽度。在满足 MaxPicWidth 范围的前提下，在当前输入图片宽度（VENC_ATTR_S::SrcPicWidth）的[1/4, 4]倍之间任意取值(必须是偶数)。 动态属性。
PicHeight	编码通道图片编码输出高度。在满足 MaxPicWidth 范围的前提下，在当前输入图片高度（VENC_ATTR_S::SrcPicHeight）的[1/4, 4]倍之间任意取值(必须是偶数)。 动态属性。
bSupportDCF	是否支持 DCF，范围：FALSE。不支持 DCF。 静态属性。

【注意事项】

【相关数据类型及接口】

VENC_ATTR_S

【说明】

编码通道编码属性结构。

【定义】

```
typedef struct VENC_ATTR_S
{
    PAYLOAD_TYPE_E    Type;                 /*the type of payload*/
    union
    {
        VENC_ATTR_H264_S    AttrH264e;      /*attributes of h264*/
        VENC_ATTR_MJPEG_S AttrMjpeg;        /*attributes of mjpeg*/
        VENC_ATTR_JPEG_S    AttrJpeg;       /*attributes of jpeg*/
        VENC_ATTR_H265_S    AttrH265e;      /*attributes of h265*/
    };
    int MaxKeyInterval;                     /* wanted key frame interval*/
    unsigned int    SrcPicWidth;    /* source width of a picture sent to venc channel, in pixel*/
    unsigned int    SrcPicHeight;   /* source height of a picture sent to venc channel, in pixel*/
    VIDEO_FIELD_E   Field;
    PIXEL_FORMAT_E PixelFormat;
}VENC_ATTR_S;
```

【成员】

成员名称	描述
Type	编码类型，支持 PT_H264，PT_H265，PT_JPEG，PT_MJPEG
AttrH264e	H264 编码属性结构
AttrMjpeg	Mjpeg 编码属性结构
AttrJpeg	Jpeg 编码属性结构
AttrH265e	H265 编码属性结构
MaxKeyInterval	关键帧间隔，范围[0,+∞)。 静态属性。
SrcPicWidth	输入图片宽度，范围[192, 4096]。 静态属性。
SrcPicHeight	输入图片高度，范围[96, 4096]。 静态属性。
Field	filed or frame 模式。 静态属性。
PixelFormat	像素格式。 静态属性。

【注意事项】

【相关数据类型及接口】

VENC_ATTR_H264_CBR_S

【说明】

定义 H.264 编码通道 CBR 属性结构。

【定义】

```
typedef struct VENC_ATTR_H264_CBR_S
{
    unsigned int    mGop;
    unsigned int    mStatTime;
    unsigned int    mSrcFrmRate;
    unsigned int    fr32DstFrmRate ;
    unsigned int    mBitRate;
    unsigned int    mFluctuateLevel;
} VENC_ATTR_H264_CBR_S;
```

【成员】

成员名称	描述
mGop	暂不支持。范围：[0]。
mStatTime	统计时间的速率。暂不支持。范围：[0]。
mSrcFrmRate	编码通道输入帧率。已废弃。使用 VENC_FRAME_RATE_S 替代。范围：[0]
fr32DstFrmRate	编码通道输出帧率。已废弃。使用 VENC_FRAME_RATE_S 替代。范围：[0]
mBitRate	平均码率，单位 bps，范围：(0,+∞)。
mFluctuateLevel	最大码率相对平均码率的波动等级。暂不支持。取值范围：[0]。

【注意事项】

【相关数据类型及接口】

VENC_ATTR_H264_VBR_S

【说明】

定义 H.264 编码通道 VBR 属性结构。

【定义】

```
typedef struct VENC_ATTR_H264_VBR_S
```

```
{
    unsigned int        mGop;
    unsigned int        mStatTime;
    unsigned int        mSrcFrmRate;
    unsigned int        fr32DstFrmRate;
    unsigned int        mMaxBitRate;
    unsigned int        mMaxQp;
    unsigned int        mMinQp;
}VENC_ATTR_H264_VBR_S;
```

【成员】

成员名称	描述
mGop	暂不支持。范围：[0]。
mStatTime	统计时间的速率。暂不支持。范围：[0]。
mSrcFrmRate	编码通道输入帧率。已废弃。使用 VENC_FRAME_RATE_S 替代。范围：[0]
fr32DstFrmRate	编码通道输出帧率。已废弃。使用 VENC_FRAME_RATE_S 替代。范围：[0]
mMaxBitRate	编码器输出最大码率，单位 bps，范围：(0,+∞)。
mMaxQp	编码器输出图像最大 Qp 值。取值范围：(mMinQp, 51]，建议[30, 45]
mMinQp	编码器输出图像最小 Qp 值。取值范围：[0, 51]，建议[10,20]。

【注意事项】

【相关数据类型及接口】

VENC_ATTR_H264_FIXQP_S

【说明】

定义 H.264 编码通道 FixQp 属性结构。

【定义】

typedef struct VENC_ATTR_H264_FIXQP_S

```
{
    unsigned int        mGop;
    unsigned int        mSrcFrmRate;
    unsigned int        fr32DstFrmRate;
    unsigned int        mIQp;
```

```
    unsigned int        mPQp;
} VENC_ATTR_H264_FIXQP_S;
```

【成员】

成员名称	描述
mGop	暂不支持。范围：[0]。
mSrcFrmRate	编码通道输入帧率。已废弃。使用 VENC_FRAME_RATE_S 替代。范围：[0]
fr32DstFrmRate	编码通道输出帧率。已废弃。使用 VENC_FRAME_RATE_S 替代。范围：[0]
mIQp	I 帧 Qp 值。取值范围：(0, 51]
mPQp	P 帧 Qp 值。取值范围：[0, 51]

【注意事项】

【相关数据类型及接口】

VENC_ATTR_MJPEG_CBR_S

【说明】

定义 MJPEG 编码通道 CBR 属性结构。

【定义】

```
typedef struct VENC_ATTR_MJPEG_CBR_S
{
    unsigned int        mStatTime;
    unsigned int        mSrcFrmRate;
    unsigned int        fr32DstFrmRate;
    unsigned int        mBitRate;
    unsigned int        mFluctuateLevel;
} VENC_ATTR_MJPEG_CBR_S;
```

【成员】

成员名称	描述
mStatTime	统计时间的速率。暂不支持。范围：[0]。
mSrcFrmRate	编码通道输入帧率。已废弃。使用 VENC_FRAME_RATE_S 替代。范围：[0]
fr32DstFrmRate	编码通道输出帧率。已废弃。使用 VENC_FRAME_RATE_S 替代。范围：[0]
mBitRate	平均码率，单位 bps，范围：(0,+∞)。

mFluctuateLevel	最大码率相对平均码率的波动等级。暂不支持。取值范围：[0]。

【注意事项】

【相关数据类型及接口】

VENC_ATTR_MJPEG_FIXQP_S

【说明】

定义 MJPEG 编码通道 FixQp 属性结构。

【定义】

```
typedef struct VENC_ATTR_MJPEG_FIXQP_S
{
    unsigned int        mSrcFrmRate;
    unsigned int        fr32DstFrmRate;
    unsigned int        mQfactor;
}VENC_ATTR_MJPEG_FIXQP_S;
```

【成员】

成员名称	描述
mSrcFrmRate	编码通道输入帧率。已废弃。使用 VENC_FRAME_RATE_S 替代。范围：[0]
fr32DstFrmRate	编码通道输出帧率。已废弃。使用 VENC_FRAME_RATE_S 替代。范围：[0]
mQfactor	JPEG 图像编码质量。取值范围：[0, 100]，建议取值[50, 100]，值越大，图像质量越高，生成的文件 size 越大。

【注意事项】

【相关数据类型及接口】

VENC_ATTR_H265_CBR_S

【说明】

定义 H.265 编码通道 CBR 属性结构。

【定义】

```
typedef struct VENC_ATTR_H265_CBR_S
{
    unsigned int        mGop;
    unsigned int        mStatTime;
```

```
    unsigned int        mSrcFrmRate;
    unsigned int        fr32DstFrmRate ;
    unsigned int        mBitRate;
    unsigned int        mFluctuateLevel;
} VENC_ATTR_H264_CBR_S;
```

【成员】

成员名称	描述
mGop	暂不支持。范围：[0]。
mStatTime	统计时间的速率。暂不支持。范围：[0]。
mSrcFrmRate	编码通道输入帧率。已废弃。使用 VENC_FRAME_RATE_S 替代。范围：[0]
fr32DstFrmRate	编码通道输出帧率。已废弃。使用 VENC_FRAME_RATE_S 替代。范围：[0]
mBitRate	平均码率，单位 bps，范围：(0,+∞)。
mFluctuateLevel	最大码率相对平均码率的波动等级。暂不支持。取值范围：[0]。

【注意事项】

【相关数据类型及接口】

VENC_ATTR_H265_VBR_S

【说明】

定义 H.265 编码通道 VBR 属性结构。

【定义】

```
typedef struct VENC_ATTR_H265_VBR_S
{
    unsigned int        mGop;
    unsigned int        mStatTime;
    unsigned int        mSrcFrmRate;
    unsigned int        fr32DstFrmRate;
    unsigned int        mMaxBitRate;
    unsigned int        mMaxQp;
    unsigned int        mMinQp;
}VENC_ATTR_H265_VBR_S;
```

【成员】

成员名称	描述

mGop	暂不支持。范围：[0]。
mStatTime	统计时间的速率。暂不支持。范围：[0]。
mSrcFrmRate	编码通道输入帧率。已废弃。使用 VENC_FRAME_RATE_S 替代。范围：[0]
fr32DstFrmRate	编码通道输出帧率。已废弃。使用 VENC_FRAME_RATE_S 替代。范围：[0]
mMaxBitRate	编码器输出最大码率，单位 bps，范围：(0,+∞)。
mMaxQp	编码器输出图像最大 Qp 值。取值范围：(mMinQp, 51]，建议[30, 45]
mMinQp	编码器输出图像最小 Qp 值。取值范围：[0, 51]，建议[10,20]。

【注意事项】

【相关数据类型及接口】

VENC_ATTR_H265_FIXQP_S

【说明】

定义 H.265 编码通道 FixQp 属性结构。

【定义】

```
typedef struct VENC_ATTR_H265_FIXQP_S
{
    unsigned int    mGop;
    unsigned int    mSrcFrmRate;
    unsigned int    fr32DstFrmRate;
    unsigned int    mIQp;
    unsigned int    mPQp;
} VENC_ATTR_H264_FIXQP_S;
```

【成员】

成员名称	描述
mGop	暂不支持。范围：[0]。
mSrcFrmRate	编码通道输入帧率。已废弃。使用 VENC_FRAME_RATE_S 替代。范围：[0]
fr32DstFrmRate	编码通道输出帧率。已废弃。使用 VENC_FRAME_RATE_S 替代。范围：[0]
mIQp	I 帧 Qp 值。取值范围：(0, 51]

mPQp	P 帧 Qp 值。取值范围：[0, 51]

【注意事项】

【相关数据类型及接口】

VENC_ATTR_H265_ABR_S

【说明】

定义 H.265 编码通道 ABR 属性结构。

【定义】

```
typedef struct VENC_ATTR_H265_ABR_S
{
    unsigned int        mGop;
    unsigned int        mStatTime;
    unsigned int        mSrcFrmRate;
    unsigned int        fr32DstFrmRate ;
    unsigned int        mAvgBitRate;
    unsigned int        mMaxBitRate;
    unsigned int        mMinStaticPercent;
    unsigned int        mMaxIQp;
    unsigned int        mMinIQp;
}VENC_ATTR_H265_ABR_S;
```

【成员】

成员名称	描述
mGop	暂不支持。范围：[0]。
mStatTime	统计时间的速率。暂不支持。范围：[0]。
mSrcFrmRate	编码通道输入帧率。已废弃。使用 VENC_FRAME_RATE_S 替代。范围：[0]
fr32DstFrmRate	编码通道输出帧率。已废弃。使用 VENC_FRAME_RATE_S 替代。范围：[0]
mAvgBitRate	编码器输出平均码率。单位 bps，范围：(0,+∞)。
mMaxBitRate	编码器输出最大码率，单位 bps，范围：(0,+∞)。
mMinStaticPercent	静态场景下最小码率和最大码率的比值百分比，范围：[20, 50]。推荐 40。
mMaxIQp	编码器 I 帧最大 Qp 值。取值范围：(mMinIQp,45]。

mMinIQp	编码器 I 帧最小 Qp 值。取值范围：[25, 32]。

【注意事项】

【相关数据类型及接口】

VENC_RC_ATTR_S

【说明】

编码通道码率控制属性。

【定义】

```
typedef struct VENC_RC_ATTR_S
{
    VENC_RC_MODE_E mRcMode;                                    /*the type of rc*/
    union
    {
        VENC_ATTR_H264_CBR_S      mAttrH264Cbr;
        VENC_ATTR_H264_VBR_S      mAttrH264Vbr;
        VENC_ATTR_H264_FIXQP_S    mAttrH264FixQp;
        VENC_ATTR_H264_ABR_S      mAttrH264Abr;
        VENC_ATTR_H264_QPMAP_S    mAttrH264QpMap;

        VENC_ATTR_MPEG4_CBR_S     mAttrMpeg4Cbr;
        VENC_ATTR_MPEG4_FIXQP_S mAttrMpeg4FixQp;
        VENC_ATTR_MPEG4_VBR_S     mAttrMpeg4Vbr;

        VENC_ATTR_MJPEG_CBR_S     mAttrMjpegeCbr;
        VENC_ATTR_MJPEG_FIXQP_S mAttrMjpegeFixQp;
        VENC_ATTR_MJPEG_VBR_S     mAttrMjpegeVbr;

        VENC_ATTR_H265_CBR_S      mAttrH265Cbr;
        VENC_ATTR_H265_VBR_S      mAttrH265Vbr;
        VENC_ATTR_H265_FIXQP_S    mAttrH265FixQp;
        VENC_ATTR_H265_ABR_S      mAttrH265Abr;
        VENC_ATTR_H265_QPMAP_S    mAttrH265QpMap;
    };
    void*           pRcAttr ;
```

}VENC_RC_ATTR_S;

【成员】

成员名称	描述
mRcMode	RC 模式（码率控制模式）。 静态属性。
mAttrH264Cbr	H.264 协议编码通道 Cbr 模式属性。
mAttrH264Vbr	H.264 协议编码通道 Vbr 模式属性。
mAttrH264FixQp	H.264 协议编码通道 Fixqp 模式属性。
mAttrH264Abr	H.264 协议编码通道 Abr 模式属性（暂不支持）。
mAttrMjpegeFixQp	Mjpeg 协议编码通道 Fixqp 模式属性。
mAttrMjpegeCbr	Mjpeg 协议编码通道 cbr 模式属性。
mAttrMjpegeVbr	Mjpeg 协议编码通道 vbr 模式属性（暂不支持）
mAttrH265Cbr	H.265 协议编码通道 Cbr 模式属性。
mAttrH265Vbr	H.265 协议编码通道 Vbr 模式属性。
mAttrH265FixQp	H.265 协议编码通道 Fixqp 模式属性。
mAttrH265Abr	H.265 协议编码通道 Abr 模式属性
mAttrH265QpMap	H.265 协议编码通道 QpMap 模式属性（暂不支持）

【注意事项】

【相关数据类型及接口】

VENC_CHN_ATTR_S

【说明】

编码通道属性结构。

【定义】

```
typedef struct VENC_CHN_ATTR_S
{
    VENC_ATTR_S VeAttr;          /*the attribute of video encoder*/
    VENC_RC_ATTR_S RcAttr;       /*the attribute of rate    ctrl*/
}VENC_CHN_ATTR_S;
```

【成员】

成员名称	描述
VeAttr	编码属性
RcAttr	码率控制属性

【注意事项】

【相关数据类型及接口】

VENC_CHN_STAT_S

【说明】

编码通道状态结构。

【定义】

```
typedef struct VENC_CHN_STAT_S
{
    unsigned int mLeftPics;              /*left picture number */
    unsigned int mLeftStreamBytes;       /*left stream bytes*/
    unsigned int mLeftStreamFrames;      /*left stream frames*/
    unsigned int mCurPacks;              /*pack number of current frame*/
    unsigned int mLeftRecvPics;
/*Number of frames to be received. This member is valid after AW_MPI_VENC_StartRecvPicEx is called.*/
    unsigned int mLeftEncPics;
/*Number of frames to be encoded. This member is valid after AW_MPI_VENC_StartRecvPicEx is called.*/
}VENC_CHN_STAT_S;
```

【成员】

成员名称	描述
mLeftPics	剩余图片量，待编码。
mLeftStreamBytes	编码流剩余 Byte 数，待取走。
mLeftStreamFrames	编码流剩余帧数，待取走。
mCurPacks	当前帧的码流包个数，取值范围：[1]，当前只支持一帧一个码流包的模式。
mLeftRecvPics	待接收图片量（仅 AW_MPI_VENC_StartRecvPicEx 时有效），未使用

mLeftEncPics	已编码图片量（仅 AW_MPI_VENC_StartRecvPicEx 时有效），未使用

【注意事项】

【相关数据类型及接口】

VENC_EXIFINFO_S

【说明】

aw EXIF 信息结构，配置缩略图大小，以及供编码器生成 jpeg 图片信息，信息可以随意填写。

【定义】

```
typedef struct VENC_EXIFINFO_S //aw
{
    unsigned char   CameraMake[MM_INFO_LENGTH];
    unsigned char   CameraModel[MM_INFO_LENGTH];
    unsigned char   DateTime[MM_DATA_TIME_LENGTH];

    unsigned int    ThumbWidth;
    unsigned int    ThumbHeight;

    int             Orientation;    //value can be 0,90,180,270 degree
    unsigned int    fr32ExposureTime; //tag 0x829A, FRACTION32()
    unsigned int    fr32FNumber; //tag 0x829D, FRACTION32()
    short           ISOSpeed;//tag 0x8827

    short           MeteringMode; //tag 0x9207, ExifMeteringModeType
    unsigned int    fr32FocalLength; //tag 0x920A

    short           WhiteBalance; //tag 0xA403

    // gps info
    int             enableGpsInfo;
    double          gps_latitude;
    double          gps_longitude;
    double          gps_altitude;
    long            gps_timestamp;
    unsigned char   gpsProcessingMethod[MM_GPS_PROCESS_METHOD_LENGTH];
```

```
    unsigned char   CameraSerialNum[128];       //tag 0xA431 (exif 2.3 version)
    short           FocalLengthIn35mmFilm;      // tag 0xA405

    unsigned char   ImageName[128];             //tag 0x010D
    unsigned char   ImageDescription[128];      //tag 0x010E
} VENC_EXIFINFO_S;
```

【成员】

成员名称	描述
CameraMake	camera 厂商。 动态属性。
CameraModel	camera 型号 动态属性。
DateTime	日期时间 动态属性。
ThumbWidth	编码通道生成的缩略图宽度。 动态属性。
ThumbHeight	编码通道生成的缩略图高度。 动态属性。
Orientation	方向。 动态属性。
fr32ExposureTime	曝光值。 动态属性。
fr32FNumber	?
ISOSpeed	?
MeteringMode	?
fr32FocalLength	?
WhiteBalance	?
enableGpsInfo	?
gps_latitude	?
gps_longitude	?
gps_altitude	?
gps_timestamp	?
gpsProcessingMethod	?

CameraSerialNum	?
FocalLengthIn35mmFilm	?
ImageName	?
ImageDescription	?

【注意事项】

【相关数据类型及接口】

VENC_JPEG_THUMB_BUFFER_S

【说明】

【定义】

```
typedef struct VENC_JPEG_THUMB_BUFFER_S //aw
{
    unsigned char* ThumbAddrVir;
    unsigned int    ThumbLen;
} VENC_JPEG_THUMB_BUFFER_S;
```

【成员】

成员名称	描述
ThumbAddrVir	jpeg 图片 buffer 中的 thumbPic 的起始虚拟地址；
ThumbLen	thumbPic 的长度。

【注意事项】

【相关数据类型及接口】

VENC_PARAM_JPEG_S

【说明】

【定义】

```
typedef struct VENC_PARAM_JPEG_S
{
    unsigned int Qfactor;                /*image quality :[1,99]*/

    unsigned char   YQt[64];             /* y qt value */
    unsigned char   CbQt[64];            /* cb qt value */
```

```
    unsigned char    CrQt[64];          /* cr qt value */

    unsigned int MCUPerECS;          /*default value: 0, MCU number of one ECS*/
} VENC_PARAM_JPEG_S;
```

【成员】

成员名称	描述
Qfactor	JPEG 编码质量。范围：[0,100]，值越大，编码质量越高。 动态属性。
YQt	未使用
CbQt	未使用
CrQt	未使用
MCUPerECS	未使用

【注意事项】

【相关数据类型及接口】

VENC_ROI_CFG_S

【说明】

定义编码感兴趣区域信息。

【定义】

```
typedef struct VENC_ROI_CFG_S
{
    unsigned int    Index;    /* Index of an ROI. The system supports indexes ranging from 0 to 7 */
    BOOL bEnable;        /* Whether to enable this ROI */
    BOOL bAbsQp;         /* QP mode of an ROI.FALSE: relative QP.TURE: absolute QP.*/
    int    Qp;           /* QP value. */
    RECT_S    Rect;      /* Region of an ROI*/
}VENC_ROI_CFG_S;
```

【成员】

成员名称	描述
Index	索引号。范围：[0,7]。 动态属性。
bEnable	使能标记。

	动态属性。
bAbsQp	是否绝对 Qp 值（FALSE: 相对 Qp 值 TRUE:绝对 Qp 值） 动态属性。
Qp	当 qp 模式是相对 Qp 值时，取值范围[0, 51]，表示用该帧的 qp 值减去 Qp，作为绝对 Qp 值；当 Qp 模式是绝对 qp 值时，取值范围[0,51]。 动态属性。
Rect	区域范围，必须在图像范围内，X,Y,Width,Height 必须 16 对齐。 动态属性。

【注意事项】

【相关数据类型及接口】

VENC_COLOR2GREY_S

【说明】

彩转灰结构

【定义】

```
typedef struct VENC_COLOR2GREY_S
{
    BOOL bColor2Grey;                /* Whether to enable Color2Grey.*/
}VENC_COLOR2GREY_S;
```

【成员】

成员名称	描述
bColor2Grey	开启或关闭一个通道的彩转灰功能。TRUE：开启；FALSE：关闭。 动态属性。

【注意事项】

【相关数据类型及接口】

VENC_CROP_CFG_S

【说明】

裁剪区域结构

【定义】

```
typedef struct VENC_CROP_CFG_S
```

```
{
    BOOL bEnable;           /* Crop region enable */
    RECT_S  Rect;           /* Crop region, note: X must be multi of 16 */
}VENC_CROP_CFG_S;
```

【成员】

成员名称	描述
bEnable	是否使能
Rect	裁剪区域

【注意事项】

【相关数据类型及接口】

VENC_FRAME_RATE_S

【说明】

帧率设置结构

【定义】

```
typedef struct VENC_FRAME_RATE_S
{
    int SrcFrmRate;             /* Input frame rate of a  channel*/
    int DstFrmRate;             /* Output frame rate of a channel*/
} VENC_FRAME_RATE_S;
```

【成员】

成员名称	描述
SrcFrmRate	进入编码通道的帧率，取值范围：[1,240]。 静态属性。
DstFrmRate	编码通道输出帧率，取值范围：(0,SrcFrmRate]。 静态属性。

【注意事项】

【相关数据类型及接口】

VENC_STREAM_BUF_INFO_S

【说明】

vbvBuffer 信息的结构体。

【定义】

```
typedef struct VENC_STREAM_BUF_INFO_S
{
    unsigned int    PhyAddr;
    void *pUserAddr;
    unsigned int    BufSize;
} VENC_STREAM_BUF_INFO_S;
```

【成员】

成员名称	描述
PhyAddr	vbvBuffer 的起始物理地址。
pUserAddr	vbvBuffer 的起始虚拟地址。
BufSize	vbvBuffer 的大小。

【注意事项】

【相关数据类型及接口】

VENC_PARAM_INTRA_REFRESH_S

【说明】

P 帧帧内刷新，刷 I 宏块控制参数

【定义】

```
typedef struct VENC_PARAM_INTRA_REFRESH_S
{
    BOOL bRefreshEnable;
    BOOL bISliceEnable;
    unsigned int    RefreshLineNum;
    unsigned int    ReqIQp;
}VENC_PARAM_INTRA_REFRESH_S;
```

【成员】

成员名称	描述
bRefreshEnable	是否开启 P 帧帧内刷新功能。TRUE：开启；FALSE：关闭。 静态属性。

bISliceEnable	未使用。
RefreshLineNum	图像帧按列划分的区域个数。例如分为 10 个区域，则每 10 帧刷新一次。取值范围：(0, +∞]，推荐值 8。 静态属性。
ReqIQp	未使用。

【注意事项】

【相关数据类型及接口】

VENC_PARAM_REF_S

【说明】

高级跳帧参考参数。

【定义】

```
typedef struct VENC_PARAM_REF_S
{
    unsigned int    Base;       /*Base layer period*/
    unsigned int    Enhance;    /*Enhance layer period*/
    BOOL    bEnablePred;        /*Whether some frames at the base layer are referenced by other frames at the base layer. When bEnablePred is FALSE, all frames at the base layer reference IDR frames.*/
} VENC_PARAM_REF_S;
```

【成员】

成员名称	描述
Base	Base 层的周期。范围：[0, ＋∞)。
Enhance	Enhance 层的周期。范围：[0, ＋∞)。
bEnablePred	base 层的帧是否被 base 层其他帧用作参考。FALSE 表示 base 层所有帧都参考 IDR 帧。

【注意事项】

【相关数据类型及接口】

VencHeaderData

【说明】

spspps 数据信息。

【定义】

```
typedef struct VencHeaderData {
    unsigned char*  pBuffer;
    unsigned int    nLength;
}VencHeaderData;
```

【成员】

成员名称	描述
pBuffer	spspps 信息的 buffer 地址，buffer 是编码库内部 buffer。
nLength	buffer 的有效数据长度

【注意事项】

【相关数据类型及接口】

VencSmartFun

【说明】

smart 编码参数。

【定义】

```
typedef struct {
    unsigned char smart_fun_en;
    unsigned char img_bin_en;
    unsigned int img_bin_th;
    unsigned int shift_bits;
}VencSmartFun;
```

【成员】

成员名称	描述
smart_fun_en	Smart 功能开关标志。 静态属性。
img_bin_en	二值化输出开关标志，当 smart 标志设置为 1 时该标志强制为 1 静态属性。
img_bin_th	运动区域判别阈值，取值范围：[]，默认值 27。 静态属性。

shift_bits	阈值计算移位位数，取值范围：[]，默认值 2。 静态属性。

【注意事项】

【相关数据类型及接口】

VencBrightnessS

【说明】

配置 h264 和 h265 编码的亮暗阈值属性，与 smart 功能配合使用，对于 smart 检索之外的非运动区域（即背景区域），如果超出过亮或过暗阈值，将会被 Smart 功能处理，默认值设置为 60 / 200，将这两个阈值往平均值调节，将会提高背景区域的压缩效率，但是显示效果可能会变差；

【定义】

```
typedef struct VencBrightnessS {
    unsigned int        dark_th; //dark threshold, default 60, range[0, 255]
    unsigned int        bright_th; //bright threshold, default 200, range[0, 255]
}VencBrightnessS;
```

【成员】

成员名称	描述
dark_th	暗阈值，范围：[0, 255]，默认值 60 静态属性。
bright_th	亮阈值，范围：[0, 255]，默认值 200 静态属性。

【注意事项】

【相关数据类型及接口】

CacheState

【说明】

编码库 vbvbuffer 的缓冲状态。

【定义】

```
typedef struct CacheState {
    unsigned int mValidSizePercent;    // 0~100
    unsigned int mValidSize;           // unit:kB
    unsigned int mTotalSize;           // unit:kB
} CacheState;
```

【成员】

成员名称	描述
mValidSizePercent	有效数据的百分比。
mValidSize	buffer 的有效数据长度。
mTotalSize	buffer 的总长度。

【注意事项】

【相关数据类型及接口】

VencSaveBSFile

【说明】

配置参数详细描述编码库保存码流的细节。

【定义】

```
typedef struct VencSaveBSFile {
    char filename[256];
    unsigned char save_bsfile_flag;
    unsigned int save_start_time;
    unsigned int save_end_time;
}VencSaveBSFile;
```

【成员】

成员名称	描述
filename[256]	码流保存路径及名称。
save_bsfile_flag	是否开启保存码流功能
save_start_time	距离开始编码的时间间隔，以该间隔作为保存码流的起始时间
save_end_time	距离开始编码的时间间隔，以该间隔作为保存码流的结束时间

【注意事项】

【相关数据类型及接口】

VeProcSet

【说明】

编码库的 proc 信息设置。

【定义】

```
typedef struct VeProcSet {
    unsigned char       bProcEnable;
    unsigned int        nProcFreq;
```

```
    unsigned int        nStatisBitRateTime;
    unsigned int        nStatisFrRateTime;
}VeProcSet;
```

【成员】

成员名称	描述
bProcEnable	是否开启 proc 调试功能。
nProcFreq	开启 proc 调试功能时每隔多少帧更新一次编码通道参数信息。
nStatisBitRateTime	码率统计时间间隔（即以该时间间隔作为 proc 信息中的瞬时码率统计时间间隔），单位为毫秒，默认值是 1000ms
nStatisFrRateTime	帧率统计时间间隔（即以该时间间隔作为 proc 信息中的瞬时帧率统计时间间隔），单位为毫秒，默认值是 1000ms

【注意事项】

【相关数据类型及接口】

5.5. 错误码

错误码	宏定义	描述
0xA0088002	ERR_VENC_INVALID_CHNID	无效的编码通道号
0xA0088003	ERR_VENC_ILLEGAL_PARAM	编码参数设置无效
0xA0088004	ERR_VENC_EXIST	编码通道已经创建
0xA0088005	ERR_VENC_UNEXIST	编码通道未创建
0xA0088006	ERR_VENC_NULL_PTR	空指针
0xA0088007	ERR_VENC_NOT_CONFIG	编码通道未配置
0xA0088008	ERR_VENC_NOT_SUPPORT	操作不支持
0xA0088009	ERR_VENC_NOT_PERM	操作不允许
0xA008800C	ERR_VENC_NOMEM	系统内存不足
0xA008800D	ERR_VENC_NOBUF	编码通道缓存分配失败
0xA008800F	ERR_VENC_BUF_FULL	编码通道缓存满
0xA0088010	ERR_VENC_SYS_NOTREADY	系统没有初始化
0xA0088012	ERR_VENC_BUSY	编码通道忙
0xA0088014	ERR_VENC_SAMESTATE	编码通道状态相同
0xA0088015	ERR_VENC_INVALIDSTATE	编码通道无效的状态
0xA0088016	ERR_VENC_INCORRECT_STATE_TRANS	编码通道不正确的状态转换

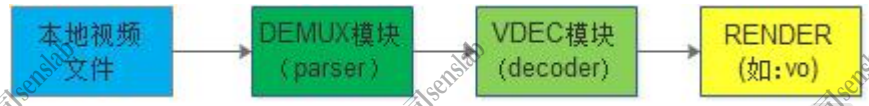
	ITION	
0xA0088017	ERR_VENC_INCORRECT_STATE_OPERATION	编码通道不正确的状态操作

6. 视频解码

6.1. 概述

VDEC 模块，即视频解码模块。本模块支持多路解码，且每路解码通道独立。

一个完整的本地文件播放流程如下所示,这里我们说的 VDEC 模块仅指送数据包到解码器解码输出图像帧(YUV 数据)的过程。



6.2. 功能描述

VDEC 模块接收待解码流输入，内部线程完成解码工作。

VDEC 模块支持 2 种数据输入输出方式：

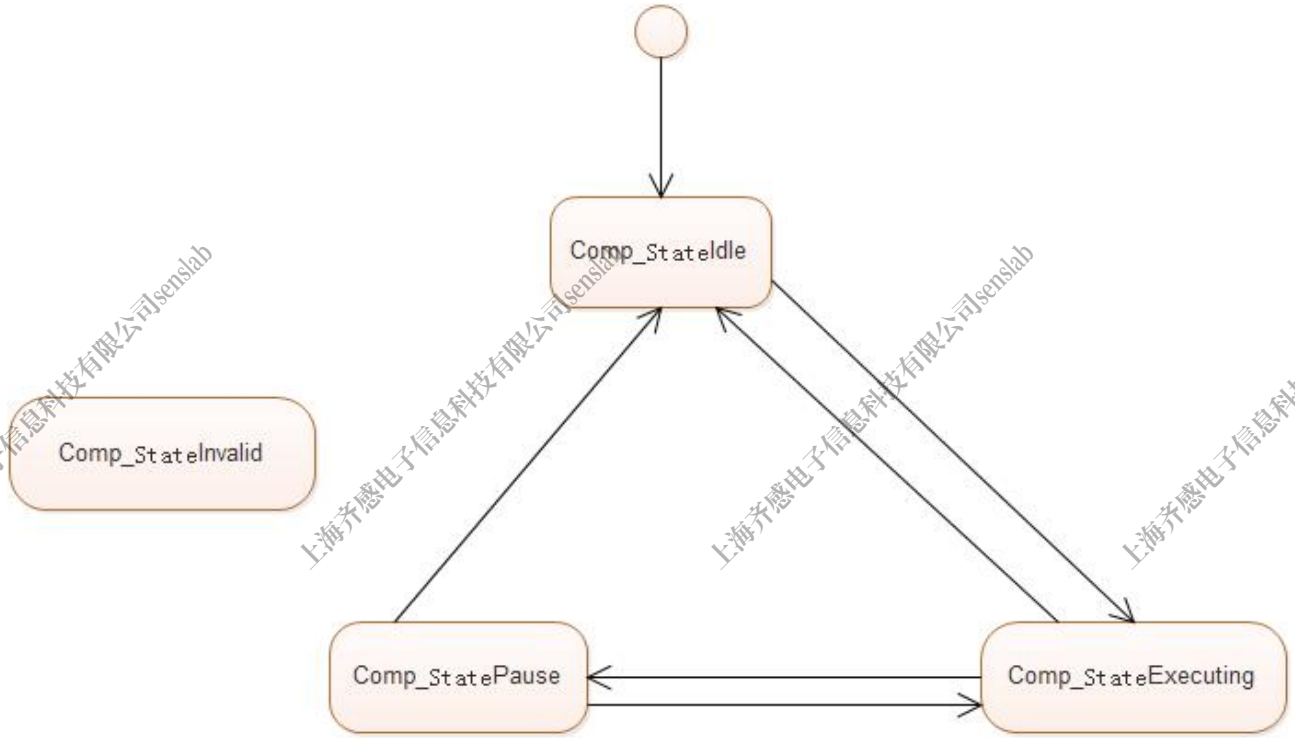
绑定方式。主控模块将 VDEC 模块的解码通道和待解码流输入组件、显示组件绑定，组件间内部传递数据。不需主控模块干预数据流入流出。

非绑定方式。主控模块调用 VDEC 模块的 mpi 层调用接口送入待解码流，获取解码帧、归还解码帧。

VDEC 模块支持图像缩放、旋转等功能。宽度和高度可分别缩小至 1/8 倍，实际宽高缩放比例用户可在解码器属性设置里面自定义设置(1/2 1/4 1/8)。支持旋转角度为：90、180、270 度。

6.3. 状态转换与 API 接口

6.3.1. 状态图



Vdec 组件内部状态设定为：

- COMP_StateLoaded：组件初始创建状态。
- COMP_StateIdle：组件完成初始化，参数设置、资源配置完毕，随时可以运行的状态。
- COMP_StateExecuting：运行状态。
- COMP_StatePause：暂停状态。
- COMP_StateInvalid：异常状态。

API AW_MPI_VDEC_CreateChn() 的实现过程会经过 COMP_StateLoaded 状态，到达 COMP_StateIdle。

组件内部状态转换的函数是：SendCommand(..., COMP_CommandStateSet, 目标 COMP_State, ...);

6.3.2. API 和状态

能够引起状态变化的 API，见状态转换图。

每个 API 只能在允许的状态下调用，如果不在允许的状态下调用 API，则无效。

API 列表如下：(允许被调用的状态栏填写 Y)

	COMP_StateIdle	COMP_StateExecuting	COMP_StatePause	COMP_StateInvalid	说明
AW_MPI_VDEC_CreateChn					引起状态转换。创建组件，完成后状态为 COMP_StateIdle
AW_MPI_VDEC_DestroyChn	Y				销毁组件
AW_MPI_VDEC_GetChnAttr	Y	Y	Y		
AW_MPI_VDEC_StartRecvStream	Y		Y		
AW_MPI_VDEC_StopRecvStream		Y	Y		
AW_MPI_VDEC_Pause		Y			
AW_MPI_VDEC_Resume			Y		引起状态转换。到 Executing，再恢复到 Idle。
AW_MPI_VDEC_Seek	Y	Y	Y		
AW_MPI_VDEC_Query	Y	Y	Y		
AW_MPI_VDEC_RegisterCallback					createChn 后必须马上 setcallback

AW_MPI_VDEC_SetStreamEof	Y	Y	Y		
AW_MPI_VDEC_ResetChn	Y				
AW_MPI_VDEC_SetChnParam	Y				
AW_MPI_VDEC_GetChnParam	Y	Y	Y		
AW_MPI_VDEC_SetProtocolParam	Y				
AW_MPI_VDEC_GetProtocolParam	Y	Y	Y		
AW_MPI_VDEC_SendStream	Y	Y	Y		
AW_MPI_VDEC_GetImage	Y	Y	Y		
AW_MPI_VDEC_ReleaseImage	Y	Y	Y		
AW_MPI_VDEC_SetRotate	Y				
AW_MPI_VDEC_GetRotate	Y	Y	Y		
AW_MPI_V		Y	Y		

DEC_GetChnLuma					

6.4. API 参考

视频解码模块主要提供视频解码通道(在本文档中通道等同于组件实例)的创建和销毁、视频解码通道的复位、开启和停止接收码流解码等功能。

AW_MPI_VDEC_CreateChn

【描述】

创建视频解码通道

【语法】

ERRORTYPE AW_MPI_VDEC_CreateChn(VDEC_CHN VdChn, const VDEC_CHN_ATTR_S *pstAttr);

【参数】

参数名称	描述	输入/输出
VdChn	解码通道号。 取值范围：[0, VDEC_MAX_CHN_NUM)。	输入
pstAttr	视频解码通道属性指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【需求】

头文件：mm_comm_vdec.h、mm_common.h

库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

AW_MPI_VDEC_DestroyChn

【描述】

销毁视频解码通道

【语法】

ERRORTYPE AW_MPI_VDEC_DestroyChn(VDEC_CHN VdChn);

【参数】

参数名称	描述	输入/输出
VdChn	解码通道号。 取值范围：[0, VDEC_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【需求】

头文件：mm_comm_vdec.h、mm_common.h

库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

AW_MPI_VDEC_GetChnAttr

【描述】

获取解码通道属性。

【语法】

ERRORTYPE AW_MPI_VDEC_GetChnAttr(VDEC_CHN VdChn, VDEC_CHN_ATTR_S *pstAttr);

【参数】

参数名称	描述	输入/输出
VdChn	解码通道号。 取值范围：[0,	输入

	VDEC_MAX_CHN_NUM)。	
pstAttr	解码通道属性指针。	输出

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【需求】

头文件：mm_comm_vdec.h、mm_common.h

库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

AW_MPI_VDEC_StartRecvStream

【描述】

开启解码通道，接收输入码流进行解码。组件状态转换为 Comp_StateExecuting。

【语法】

ERRORTYPE AW_MPI_VDEC_StartRecvStream(VDEC_CHN VdChn);

【参数】

参数名称	描述	输入/输出
VdChn	解码通道号。 取值范围：[0, VDEC_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【需求】

头文件：mm_comm_vdec.h、mm_common.h

库文件：libmedia_mpp.so

【注意】

如果通道未创建，则返回失败 AW_ERR_VENC_UNEXIST

如果当前已经开启接收，此接口也返回成功。

只有开启接收之后解码器才开始接收码流解码。

【举例】

无。

AW_MPI_VDEC_StopRecvStream

【描述】

停止编码通道接收输入数据。

【语法】

ERRORTYPE AW_MPI_VDEC_StopRecvStream(VDEC_CHN VdChn);

【参数】

参数名称	描述	输入/输出
VdChn	解码通道号。 取值范围：[0, VDEC_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【需求】

头文件：mm_comm_vdec.h、mm_common.h

库文件：libmedia_mpp.so

【注意】

如果通道未创建，则返回失败。

此接口并不判断当前是否停止接收，即如果当前已经停止接收，调用此接口也返回成功。

此接口用于解码通道停止接收码流解码，在解码通道销毁或复位前必须停止接收码流。

调用此接口仅停止接收码流解码，码流 buffer 并不会被清除。

【举例】

无。

AW_MPI_VDEC_Pause

【描述】

暂停解码，解码组件状态转为 COMP_StatePause。

【语法】

ERRORTYPE AW_MPI_VDEC_Pause(VDEC_CHN VdChn);

【参数】

参数名称	描述	输入/输出
VdChn	解码通道号。 取值范围：[0, VDEC_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【需求】

头文件：mm_comm_vdec.h、mm_common.h

库文件：libmedia_mpp.so

【注意】

只能从 Comp_StateExecuting 状态转换到 Comp_StatePause 状态，其他状态下返回失败。

此接口并不判断当前是否已经暂停，即如果当前已经暂停，调用此接口也返回成功。

【举例】

无。

AW_MPI_VDEC_Resume

【描述】

继续解码，解码组件状态从 COMP_StatePause 转为 Comp_StateExecuting。

【语法】

ERRORTYPE AW_MPI_VDEC_Resume(VDEC_CHN VdChn);

【参数】

参数名称	描述	输入/输出
VdChn	解码通道号。 取值范围：[0, VDEC_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【需求】

头文件：mm_comm_vdec.h、mm_common.h

库文件：libmedia_mpp.so

【注意】

只能从状态 Comp_StatePause 状态执行，其他状态下返回失败。

【举例】

无。

AW_MPI_VDEC_Seek

【描述】

设置视频解码通道完成跳播后播放的准备。

【语法】

ERRORTYPE AW_MPI_VDEC_Seek(VDEC_CHN VdChn);

【参数】

参数名称	描述	输入/输出
VdChn	解码通道号。 取值范围：[0, VDEC_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【需求】

头文件：mm_comm_vdec.h、mm_common.h

库文件：libmedia_mpp.so

【注意】

无

【举例】

无。

AW_MPI_VDEC_Query

【描述】

查询视频解码通道状态。

【语法】

ERRORTYPE AW_MPI_VDEC_Query(VDEC_CHN VdChn, VDEC_CHN_STAT_S *pstStat);

【参数】

参数名称	描述	输入/输出
VdChn	解码通道号。 取值范围：[0, VDEC_MAX_CHN_NUM)。	输入
pstStat	通道当前状态	输出

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【需求】

头文件：mm_comm_vdec.h、mm_common.h

库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

AW_MPI_VDEC_RegisterCallback

【描述】

设置解码通道回调

【语法】

ERRORTYPE AW_MPI_VDEC_RegisterCallback(VDEC_CHN VdChn, MPPCallbackInfo *pCallback);

【参数】

参数名称	描述	输入/输出
VdChn	解码通道号。 取值范围：[0, VDEC_MAX_CHN_NUM)。	输入
pCallback	回调信息。	输出

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【需求】

- 头文件：mm_comm_vdec.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

AW_MPI_VDEC_SetStreamEof

【描述】

设置解码输入码流结束标志

【语法】

ERRORTYPE AW_MPI_VDEC_SetStreamEof(VDEC_CHN VdChn, BOOL bEofFlag);

【参数】

参数名称	描述	输入/输出
VdChn	解码通道号。 取值范围：[0, VDEC_MAX_CHN_NUM)。	输入
bEofFlag	结束标记，TRUE：码流结束；FALSE：码流未结束。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【需求】

头文件：mm_comm_vdec.h、mm_common.h

库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

AW_MPI_VDEC_ResetChn

【描述】

复位解码通道，但不会重置已设置的解码参数，不会释放已分配的码流缓冲和视频帧。解码通

道复位后，缓冲数据都被清空，随时等待再次解码。

【语法】

ERRORTYPE AW_MPI_VDEC_ResetChn(VDEC_CHN VdChn);

【参数】

参数名称	描述	输入/输出
VdChn	解码通道号。 取值范围：[0, VDEC_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【需求】

头文件：mm_comm_vdec.h、mm_common.h

库文件：libmedia_mpp.so

【注意】

Reset 并不存在的通道，返回失败 AW_ERR_VENC_UNEXIST。

如果一个通道没有停止接收码流而 reset 通道，则返回失败。

【举例】

无。

AW_MPI_VDEC_SetChnParam

【描述】

设置解码通道参数。

【语法】

ERRORTYPE AW_MPI_VDEC_SetChnParam(VDEC_CHN VdChn, VDEC_CHN_PARAM_S* pstParam);

【参数】

参数名称	描述	输入/输出
VdChn	解码通道号。 取值范围：[0, VDEC_MAX_CHN_NUM)。	输入
pstParam	解码通道参数	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【需求】

头文件：mm_comm_vdec.h、mm_common.h

库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

AW_MPI_VDEC_GetChnParam

【描述】

获取解码通道参数。

【语法】

ERRORTYPE AW_MPI_VDEC_GetChnParam(VDEC_CHN VdChn, VDEC_CHN_PARAM_S* pstParam);

【参数】

参数名称	描述	输入/输出
VdChn	解码通道号。 取值范围：[0, VDEC_MAX_CHN_NUM)。	输入
pstParam	解码通道参数指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【需求】

头文件：mm_comm_vdec.h、mm_common.h

库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

AW_MPI_VDEC_SendStream

【描述】

发送待解码码流给解码通道进行解码。

【语法】

ERRORTYPE AW_MPI_VDEC_SendStream(VDEC_CHN VdChn, const VDEC_STREAM_S *pstStream, int s32MilliSec);

【参数】

参数名称	描述	输入/输出
VdChn	解码通道号。 取值范围：[0, VDEC_MAX_CHN_NUM)。	输入
pstStream	码流信息结构指针	输入
s32MilliSec	发送码流超时时间。 取值范围：[-1, +∞) -1：阻塞 0：非阻塞 >0：超时时间	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【需求】

头文件：mm_comm_vdec.h、mm_common.h

库文件：libmedia_mpp.so

【注意】

超时时间 s32MilliSec 的含义，-1 表示必须等到该 stream 进入解码通道 vbvBuffer 中；0：立刻返回结果，如果当前等待解码的 vbvBuffer 满，返回失败；>0：如果 vbvBuffer 满，等待到设定的时间再返回超时。

仅用于组件非绑定方式

【举例】

无。

AW_MPI_VDEC_GetImage

【描述】

获取解码后的帧。

【语法】

ERRORTYPE AW_MPI_VDEC_GetImage(VDEC_CHN VdChn, VIDEO_FRAME_INFO_S *pstFrameInfo,int s32MilliSec);

【参数】

参数名称	描述	输入/输出
VdChn	解码通道号。 取值范围：[0, VDEC_MAX_CHN_NUM)。	输入
pstFrameInfo	解码帧结构体指针	输出
s32MilliSec	获取解码帧的超时时间。 取值范围：[-1, +∞) -1：阻塞 0：非阻塞 0：超时时间，单位毫秒	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【需求】

头文件：mm_comm_vdec.h、mm_common.h

库文件：libmedia_mpp.so

【注意】

仅用于组件非绑定方式

【举例】

无。

AW_MPI_VDEC_ReleaseImage

【描述】

释放解码帧给解码通道。

【语法】

ERRORTYPE AW_MPI_VDEC_ReleaseImage(VDEC_CHN VdChn, VIDEO_FRAME_INFO_S *pstFrameInfo);

【参数】

参数名称	描述	输入/输出
VdChn	解码通道号。 取值范围：[0, VDEC_MAX_CHN_NUM)。	输入
pstFrameInfo	帧结构体指针，使用时只需填写 pstFramaInfo->mId 即可。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【需求】

头文件：mm_comm_vdec.h、mm_common.h

库文件：libmedia_mpp.so

【注意】

仅用于组件非绑定方式

【举例】

无。

AW_MPI_VDEC_SetRotate

【描述】

设置解码旋转顺时针角度。

【语法】

ERRORTYPE AW_MPI_VDEC_SetRotate(VDEC_CHN VdChn, ROTATE_E enRotate);

【参数】

参数名称	描述	输入/输出
VdChn	解码通道号。 取值范围：[0, VDEC_MAX_CHN_NUM)。	输入
enRotate	旋转角度枚举类型。 静态属性。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【需求】

头文件：mm_comm_vdec.h、mm_common.h

库文件：libmedia_mpp.so

【注意】

必须在解码开始前设置，解码过程中设置无效。

【举例】

无。

AW_MPI_VDEC_GetRotate

【描述】

获取解码旋转顺时针角度。

【语法】

ERRORTYPE AW_MPI_VDEC_GetRotate(VDEC_CHN VdChn, ROTATE_E *penRotate);

【参数】

参数名称	描述	输入/输出
VdChn	解码通道号。 取值范围：[0, VDEC_MAX_CHN_NUM)。	输入
penRotate	旋转角度枚举类型指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【需求】

头文件：mm_comm_vdec.h、mm_common.h

库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

AW_MPI_VDEC_ReopenVideoEngine

【描述】

重置解码引擎。解码库内部销毁 vbvBuffer 和 frame buffers。需要在解码通道检测到图像分辨率变化并通过 callback 通知之后调用。

【语法】

ERRORTYPE AW_MPI_VDEC_ReopenVideoEngine(VDEC_CHN VdChn);

【参数】

参数名称	描述	输入/输出
VdChn	解码通道号。 取值范围：[0, VDEC_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【需求】

头文件：mm_comm_vdec.h、mm_common.h

库文件：libmedia_mpp.so

【注意】

【举例】

无。

6.5. 数据结构说明

VIDEO_MODE_E

【说明】

解码器输入码流方式

【定义】

```
typedef enum VIDEO_MODE_E {
    VIDEO_MODE_STREAM = 0,     /*send by stream*/
    VIDEO_MODE_FRAME,          /*send by frame*/

    VIDEO_MODE_BUTT
} VIDEO_MODE_E;
```

【成员】

成员名称	描述
VIDEO_MODE_STREAM	流模式
VIDEO_MODE_FRAME	帧模式
VIDEO_MODE_BUTT	无效

【注意事项】

无。

【相关数据类型及接口】

无。

VDEC_CHN_ATTR_S

【说明】

【定义】

```
typedef struct VDEC_CHN_ATTR_S {
    PAYLOAD_TYPE_E mType;       /* video type to be decoded */
    unsigned int mBufSize;      /* stream buf size(Byte) */
    unsigned int mPriority;     /* priority */
    unsigned int mPicWidth;     /* max pic width */
    unsigned int mPicHeight; /* max pic height */
    int mInitRotation;     //clockwise rotation: val=0 no rotation, val=1 90 degree; val=2 180 degree, val=3 270 degree
    PIXEL_FORMAT_E mOutputPixelFormat;
    union {
        VDEC_ATTR_JPEG_S mVdecJpegAttr;      /* structure with jpeg or mjpeg type */
        VDEC_ATTR_VIDEO_S mVdecVideoAttr; /* structure with video ( h264/mpeg4) */
    };
} VDEC_CHN_ATTR_S;
```

【成员】

成员名称	描述
mType	解码类型
mBufSize	vbvBuffer 的长度。单位字节。取值范围：[0, +∞)。0 表示解码通道自行决定。

	静态属性。
mPriority	未使用。
mPicWidth	解码输出图片的最大宽度。范围：[0, 3840]。0 表示没有限制。如果原图宽度超过最大宽度，解码通道将进行压缩，压缩比为 1/2,1/4,1/8。原图宽度不能超过 3840，否则无法解码。 静态属性。
mPicHeight	解码输出图片的最大高度。范围：[0, 2160]。0 表示没有限制。如果原图高度超过最大高度，解码通道将进行压缩，压缩比为 1/2,1/4,1/8。原图高度不能超过 2160，否则无法解码。 静态属性。
mInitRotation	解码输出图片旋转角度。范围：[0,3]。1：顺时针旋转 90 度；2：顺时针旋转 180 度；3：顺时针旋转 270 度。 静态属性。
mOutputPixelFormat	解码输出图片相素格式。 静态属性。
mVdecJpegAttr	未使用。
mVdecVideoAttr	未使用。

【注意事项】

无。

【相关数据类型及接口】

无。

VDEC_STREAM_S

【说明】

解码器输入码流

【定义】

```
typedef struct VDEC_STREAM_S {
    unsigned char* pAddr;    /* stream address */
    unsigned int mLen;       /* stream len */
    uint64_t mPTS;           /* time stamp */
    BOOL mbEndOfFrame;       /* is the end of a frame */
    BOOL mbEndOfStream;      /* is the end of all stream */
} VDEC_STREAM_S;
```

【成员】

成员名称	描述

pAddr	输入码流地址
mLen	输入码流数据长度
mPTS	时间戳
mbEndOfFrame	是否是一帧数据的结束
mbEndOfStream	是否是码流的结尾

【注意事项】

无。

【相关数据类型及接口】

无。

VDEC_DECODE_ERROR_S

【说明】

解码器错误类型

【定义】

```
typedef struct VDEC_DECODE_ERROR_S {
    int mFormatErr;          /* format error. eg: do not support filed */
    int mPicSizeErrSet;      /* picture width or height is larger than chnnel width or height*/
    int mStreamUnsprt;       /* unsupport the stream specification */
    int mPackErr;            /* stream package error */
    int mPrtclNumErrSet;     /* protocol num is not enough. eg: slice, pps, sps */
    int mRefErrSet;          /* refrence num is not enough */
    int mPicBufSizeErrSet;   /* the buffer size of picture is not enough */
} VDEC_DECODE_ERROR_S;
```

【成员】

成员名称	描述
mFormatErr	
mPicSizeErrSet	
mStreamUnsprt	
mPackErr	
mPrtclNumErrSet	
mRefErrSet	
mPicBufSizeErrSet	

【注意事项】

无。

【相关数据类型及接口】

无。

VDEC_CHN_STAT_S

【说明】

解码通道属性

【定义】

```
typedef struct VDEC_CHN_STAT_S {
    PAYLOAD_TYPE_E mType;                   /* video type to be decoded */
    unsigned int mLeftStreamBytes;      /* left stream bytes waiting for decode */
    unsigned int mLeftStreamFrames;
        /* left frames waiting for decode,only valid for H264D_MODE_FRAME */
    unsigned int mLeftPics;                 /* pics waiting for output */
    BOOL mbStartRecvStream;                 /* had started recv stream? */
    unsigned int mRecvStreamFrames;     /* how many frames of stream has been received. valid
when                                                send by frame. */
    unsigned int mDecodeStreamFrames;
                                                /* how many frames of stream has been decoded.
valid when send by frame. */
    VDEC_DECODE_ERROR_S mVdecDecErr;    /* information about decode error */
} VDEC_CHN_STAT_S;
```

【成员】

成员名称	描述
mType	解码类型
mLeftStreamBytes	解码器输入有多少数据待解码
mLeftStreamFrames	解码器输入待解码数据有多少帧
mLeftPics	解码器输出剩余多少图片没有取
mbStartRecvStream	是否开始接收待解码数据
mRecvStreamFrames	接收到的待解码数据有多少帧（按帧传送方式时），未使用。
mDecodeStreamFrames	解码出来的数据有多少帧（按帧传送方式时），未使用。
mVdecDecErr	解码器错误类型，未使用。

【注意事项】

无。

【相关数据类型及接口】

无。

VDEC_CHN_PARAM_S

【说明】

解码通道参数

【定义】

```
typedef struct VDEC_CHN_PARAM_S {
    int mChanErrThr;
    int mChanStrmOFThr;
    int mDecMode;
    int mDecOrderOutput;
    VIDEO_FORMAT_E mVideoFormat;
    COMPRESS_MODE_E mCompressMode;
} VDEC_CHN_PARAM_S;
```

【成员】

成员名称	描述
mChanErrThr	未使用
mChanStrmOFThr	未使用
mDecMode	解码模式。0：普通解码；1：I 帧和 P 帧解码；2：I 帧解码。目前只支持 0 和 1。 动态属性。
mDecOrderOutput	未使用
mVideoFormat	未使用
mCompressMode	未使用

【注意事项】

无。

【相关数据类型及接口】

无。

VDEC_PRTCL_PARAM_S

【说明】

【定义】

```
typedef struct VDEC_PRTCL_PARAM_S {
    int mMaxSliceNum;        /* max    slice num support */
    int mMaxSpsNum;          /* max sps num support */
```

```
    int mMaxPpsNum;          /* max pps num support */
    int mDisplayFrameNum;    /* display frame num */
} VDEC_PRTCL_PARAM_S;
```

【成员】

成员名称	描述
mMaxSliceNum	最大 slice 数量，未使用。
mMaxSpsNum	最大 sps 数量，未使用。
mMaxPpsNum	最大 pps 数量，未使用。
mDisplayFrameNum	显示帧数量，未使用。

【注意事项】

无。

【相关数据类型及接口】

无。

6.6. 错误码

错误代码	宏定义	描述
0xA0058002	ERR_VDEC_INVALID_CHNID	通道 ID 超出合法范围
0xA0058003	ERR_VDEC_ILLEGAL_PARAM	参数超出合法范围
0xA0058004	ERR_VDEC_EXIST	试图申请或者创建已经存在的设备、通道或者资源
0xA0058006	ERR_VDEC_NULL_PTR	函数参数中有空指针
0xA0058007	ERR_VDEC_NOT_CONFIG	使用前未配置
0xA0058008	ERR_VDEC_NOT_SUPPORT	不支持的参数或者功能
0xA0058009	ERR_VDEC_NOT_PERM	该操作不允许，如试图修改静态配置参数
0xA0058005	ERR_VDEC_UNEXIST	试图使用或者销毁不存在的设备、通道或者资源
0xA005800C	ERR_VDEC_NOMEM	分配内存失败，如系统内存不足
0xA005800D	ERR_VDEC_NOBUF	分配缓存失败，如申请的数据缓冲区太大
0xA005800E	ERR_VDEC_BUF_EMPTY	缓冲区中无数据

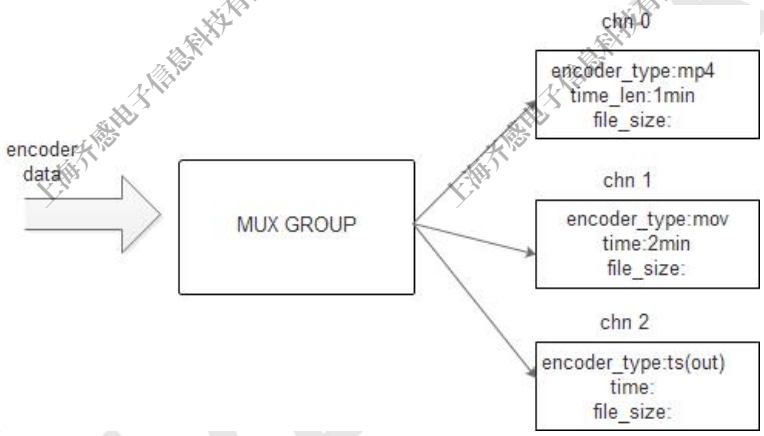
0xA005800F	ERR_VDEC_BUF_FULL	缓冲区中数据满
0xA0058010	ERR_VDEC_SYS_NOTREADY	系统没有初始化或没有加载相应模块
0xA0058012	ERR_VDEC_BUSY	VENC 系统忙
0xA0058014	ERR_VDEC_SAMESTATE	状态相同
0xA0058015	ERR_VDEC_INVALIDSTATE	无效的状态
0xA0058016	ERR_VDEC_INCORRECT_STATE_TRANSITION	状态转换出错
0xA0058011	ERR_VDEC_BADADDR	非法地址

7. MUX 模块

7.1. 概述

mux 模块，即文件封装模块。本模块以 muxGroup 为单位，一个 muxGroup 包含一个或多个 mux 通道，属于同一 muxGroup 的 mux 通道对输入的同一视频、音频编码数据流进行封装。

使用 muxGroup 的原因是：有时不仅需要把视频、音频数据流封装为 mp4 文件本地存储，同时还需封装为 raw 码流或 ts 码流通过网络传输，这时需要对接收的同一视频、音频流数据同时进行 2 路封装处理（1 路处理对应一个 mux 通道），故设计 muxGroup。目前一个 muxGroup 最多支持两个 muxchn 输出。

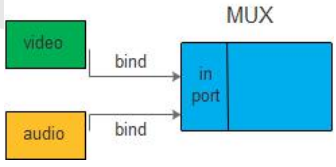


7.2. 功能描述与使用

mux 通道内部线程完成封装和写入（写卡）的功能。

mux 模块只支持绑定方式的数据输入。主控模块必须将 muxGroup 组件和编码组件绑定。mux 模块的数据输入对象是 muxGroup。muxChannel 在 group 内部接收处理数据。

一个 muxGroup 输入端最多支持 2 个输入绑定（video enc/audio enc）。

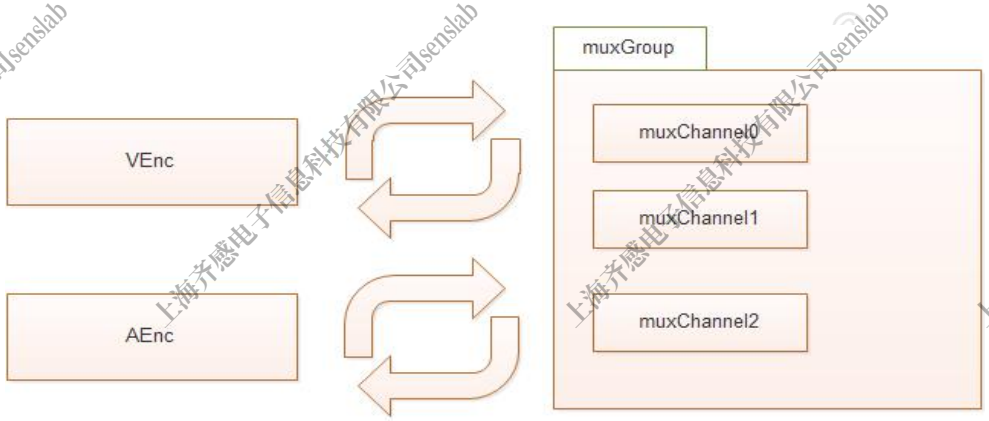


本地存储操作时，应用 app 仅需要设置录像文件 fd 与 mux 进行互动。录像时，mux 会在当前录像文件结束前 10 秒向 app 应用发消息 MPP_EVENT_NEXT_FD，获取下一个录像文件的句柄 fd（mux 内部会对此 fd 进行 dup 操作，应用 app 应自己 close 该 fd）；mux 在封装完一个文件后会向 app 应用发送 MPP_EVENT_RECORD_DONE 消息，表示一个录像文件已经完成。

对于非本志存储操作（如 ts 网络传输），设置 chn 属性时可设置 mCallbackOutflag 标志（数据操作由外部回调完成），mux 会向应用发出消息 MPP_EVENT_BSFRAME_AVAILABLE 消息，由应用自行处理数据。

mux 模块本身不对数据流进行任何的拷贝操作，所有的数据 buffer 均由输入端绑定通道（venc 或者 aenc 组件实例）提供。因此在销毁 muxGroup 和 muxGroup 的输入绑定端的时候，muxGroup 必须先于输入绑定端组件销毁（销毁 muxGroup 时会处理所有关联数据 buffer），将相关的编码数据 packet buffer 还给 venc 或者 aenc 组件。

7.2.1. muxGroup 和 muxChannel



如上图所示：muxGroup 和 VEnc,AEnc 组件交互数据，Group 下的各 muxChannel 处理同样的音视频数据，同时满足不同的封装要求。

Group 参数是配置给 Group 下所有 channel 的参数。

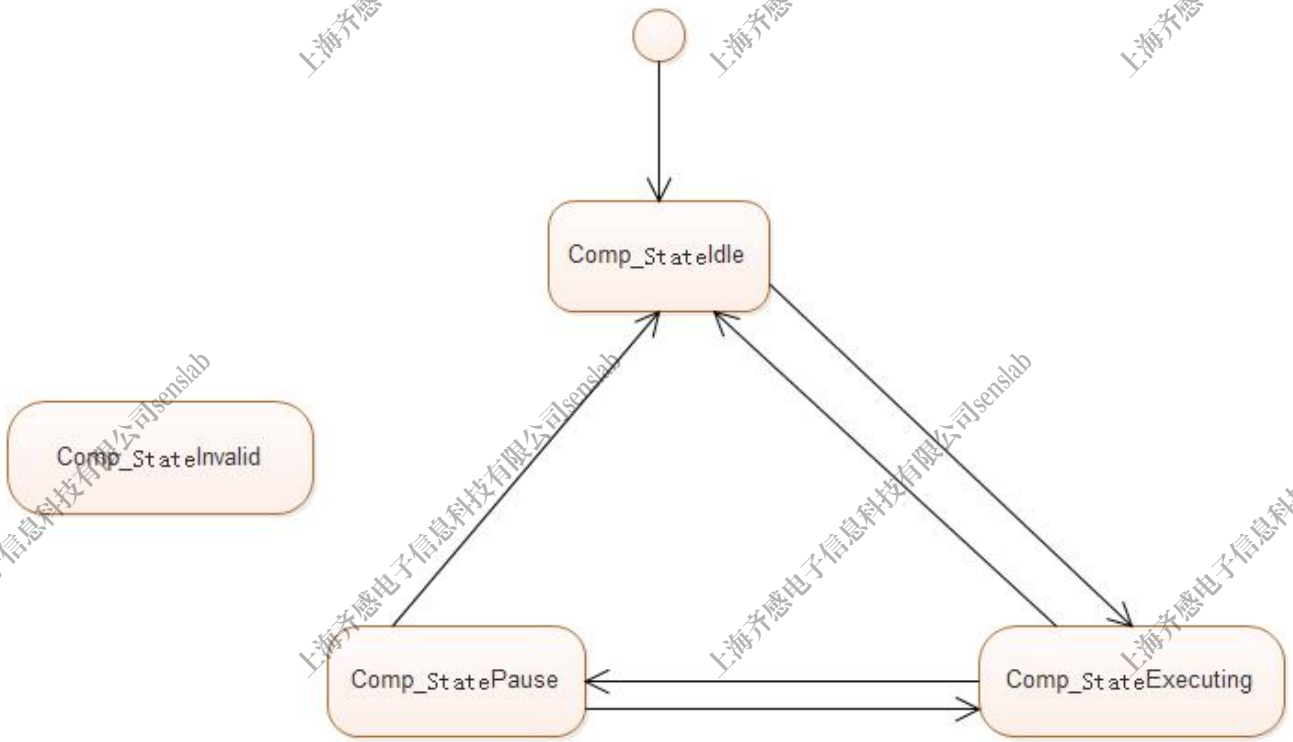
channel 参数是单独设置给指定的 channel 的。

Group 参数和 channel 参数的优先级为：channel > group。即如果某个 channel 的 channel 参数和 group 参数有冲突，那么以 channel 参数为准。

muxGroup 属性和 muxChannel 属性的关系也是：channel > group。

muxChannel 必须归属于某个 muxGroup 之下，muxGroup 同一操作其下的 channel，如 start, stop, reset 等。channel 和 Group 的隶属关系，由调用者在创建 muxChannel 时指定。必须先创建 muxGroup，再创建 muxChannel。

7.2.2. 状态图



demux 组件内部状态设定为：

- COMP_StateLoaded：组件初始创建状态。
- COMP_StateIdle：组件完成初始化，参数设置、资源配置完毕，随时可以运行的状态。
- COMP_StateExecuting：运行状态。
- COMP_StatePause：暂停状态。
- COMP_StateInvalid：异常状态。

AW_MPI_MUX_CreateChn()的实现过程会经过 COMP_StateLoaded 状态，到达 COMP_StateIdle。

组件内部状态转换的函数是：SendCommand(..., COMP_CommandStateSet, 目标 COMP_State, ...)；

7.2.3. API 和状态

能够引起状态变化的 API，见状态转换图。

每个 API 只能在允许的状态下调用，如果不在允许的状态下调用 API，则无效。

API 列表如下：(允许被调用的状态栏填写 Y)

	COMP_StateIdle	COMP_Pause	COMP_StateExecuting	COMP_StateInvalid	说明
AW_MPI_MUX_CreateGrp					创建 muxgroup
AW_MPI_MUX_DestroyGrp					销毁 muxgroup
AW_MPI_MUX_StartGrp	Y				启动 muxgroup，引起状态转换，到 Executing
AW_MPI_MUX_StopGrp		Y	Y		重置组件到初始化状态
AW_MPI_MUX_GetGrpAttr	Y		Y		
AW_MPI_MUX_SetGrpAttr	Y		Y		
AW_MPI_MUX_SetH264SpsPpsInfo	Y		Y		
AW_MPI_MUX_SetH265SpsPpsInfo	Y		Y		
AW_MPI_MUX_CreateChn	Y		Y		

AW_MPI_MUX_DestroyChn	Y		Y		
AW_MPI_MUX_GetChnAttr	Y		Y		
AW_MPI_MUX_SetChnAttr	Y		Y		
AW_MPI_MUX_SwitchFd			Y		
AW_MPI_MUX_RegisterCallback					

7.3. API 参考

mux 模块主要提供 muxGroup， mux 通道(在本文档中通道等同于组件实例)的创建和销毁、通道的复位、开启和停止等功能。

AW_MPI_MUX_CreateGrp

【描述】

创建 muxGroup。

【语法】

ERRORTYPE AW_MPI_MUX_CreateGrp(MUX_GRP muxGrp, MUX_GRP_ATTR_S *pGrpAttr);

【参数】

参数名称	描述	输入/输出
muxGrp	MUX GROUP 号。 取值范围：[0, MUX_MAX_GRP_NUM)。	输入
pGrpAttr	MUX GROUP 属性指针	输入

【返回值】

返回值	描述
0	成功

非 0	失败，参见错误码。

【注意】

无。

【举例】

无。

AW_MPI_MUX_DestroyGrp

【描述】

销毁 muxGroup

【语法】

ERRORTYPE AW_MPI_MUX_DestroyGrp(MUX_GRP muxGrp);

【参数】

参数名称	描述	输入/输出
muxGrp	MUX GROUP 号。 取值范围：[0, MUX_MAX_GRP_NUM)。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【注意】

无。

【举例】

无。

AW_MPI_MUX_StartGrp

【描述】

启动 muxGroup 接收数据。

【语法】

ERRORTYPE AW_MPI_MUX_StartGrp(MUX_GRP muxGrp);

【参数】

参数名称	描述	输入/输出
muxGrp	MUX GROUP 号。 取值范围：[0,	输入

	MUX_MAX_GRP_NUM)。	

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【注意】

无。

【举例】

无。

AW_MPI_MUX_StopGrp

【描述】

停止 muxGroup 接收数据。

【语法】

ERRORTYPE AW_MPI_MUX_StopGrp(MUX_GRP muxGrp);

【参数】

参数名称	描述	输入/输出
muxGrp	MUX GROUP 号。 取值范围：[0, MUX_MAX_GRP_NUM)。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【注意】

无。

【举例】

无。

AW_MPI_MUX_GetGrpAttr

【描述】

获取 muxGroup 属性。

【语法】

ERRORTYPE AW_MPI_MUX_GetGrpAttr(MUX_GRP muxGrp, MUX_GRP_ATTR_S *pGrpAttr);

【参数】

参数名称	描述	输入/输出
muxGrp	MUX GROUP 号。 取值范围：[0, MUX_MAX_GRP_NUM)。	输入
pGrpAttr	muxGroup 属性指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【注意】

无。

【举例】

无。

AW_MPI_MUX_SetGrpAttr

【描述】

设置 muxGroup 属性。

【语法】

ERRORTYPE AW_MPI_MUX_SetGrpAttr(MUX_GRP muxGrp, MUX_GRP_ATTR_S *pGrpAttr);

【参数】

参数名称	描述	输入/输出
dmxChn	MUX GROUP 号。 取值范围：[0, MUX_MAX_GRP_NUM)。	输入
pGrpAttr	muxGroup 属性指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【注意】

- 如果通道未创建，则返回失败。
- 此接口并不判断当前是否已经暂停解析，如果已经暂停，调用此接口也返回成功。

【举例】

无。

AW_MPI_MUX_SetH264SpsPpsInfo

【描述】

设置 muxgroup 的 H264spspps 信息

【语法】

ERRORTYPE AW_MPI_MUX_SetH264SpsPpsInfo(MUX_GRP muxGrp, VencHeaderData *pH264SpsPpsInfo);

【参数】

参数名称	描述	输入/输出
muxGrp	MUX GROUP 号。 取值范围：[0, MUX_MAX_GRP_NUM)。	输入
pH264SpsPpsInfo	H264 spspps 信息头	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【注意】

无。

【举例】

无。

AW_MPI_MUX_SetH265SpsPpsInfo

【描述】

设置 muxgroup 的 spspps 信息

【语法】

ERRORTYPE AW_MPI_MUX_SetH265SpsPpsInfo(MUX_GRP muxGrp, VencHeaderData *pH264SpsPpsInfo);

【参数】

参数名称	描述	输入/输出
muxGrp	MUX GROUP 号。 取值范围：[0, MUX_MAX_GRP_NUM)。	输入
pH265SpsPpsInfo	H265 spspps 信息头	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【注意】

无。

【举例】

无。

AW_MPI_MUX_CreateChn

【描述】

创建 mux 通道，必须指定归属的 muxGroup。

【语法】

ERRORTYPE AW_MPI_MUX_CreateChn(MUX_GRP muxGrp, MUX_CHN muxChn, MUX_CHN_ATTR_S *pChnAttr， int nFd);

【参数】

参数名称	描述	输入/输出
muxGrp	MUX GROUP 号。 取值范围：[0, MUX_MAX_GRP_NUM)。	输入
muxChn	MUX chn 号。 取值范围：[0, MUX_MAX_CHN_NUM)。	输入
pChnAttr	mux 通道属性指针	输入
nFd	文件句柄	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【注意】

无。

【举例】

无。

AW_MPI_MUX_DestroyChn

【描述】

销毁 mux 通道，从 muxGroup 中撤出。

【语法】

ERRORTYPE AW_MPI_MUX_DestroyChn(MUX_GRP muxGrp, MUX_CHN muxChn);

【参数】

参数名称	描述	输入/输出
muxGrp	MUX GROUP 号。 取值范围：[0, MUX_MAX_GRP_NUM)。	输入
muxChn	MUX chn 号。 取值范围：[0, MUX_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【注意】

无。

【举例】

无。

AW_MPI_MUX_GetChnAttr

【描述】

获取 mux 通道属性。

【语法】

ERRORTYPE AW_MPI_MUX_GetChnAttr(MUX_GRP muxGrp, MUX_CHN muxChn, MUX_CHN_ATTR_S *pChnAttr);

【参数】

参数名称	描述	输入/输出
muxGrp	MUX GROUP 号。 取值范围：[0, MUX_MAX_GRP_NUM)。	输入
muxChn	MUX chn 号。	输入

	取值范围：[0, MUX_MAX_CHN_NUM)。	
pChnAttr	mux 通道属性指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【注意】

无。

【举例】

无。

AW_MPI_MUX_SetChnAttr

【描述】

设置 mux 通道属性。

【语法】

ERRORTYPE AW_MPI_MUX_SetChnAttr(MUX_GRP muxGrp, MUX_CHN muxChn, MUX_CHN_ATTR_S *pChnAttr);

【参数】

参数名称	描述	输入/输出
muxGrp	MUX GROUP 号。 取值范围：[0, MUX_MAX_GRP_NUM)。	输入
muxChn	MUX chn 号。 取值范围：[0, MUX_MAX_CHN_NUM)。	输入
pChnAttr	mux 通道属性指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【注意】

无。

【举例】

无。

AW_MPI_MUX_SwitchFd

【描述】

切换（录像）文件句柄

【语法】

ERRORTYPE AW_MPI_MUX_SwitchFd(MUX_GRP muxGrp, MUX_CHN muxChn, int fd, int nFallocateLen);

【参数】

参数名称	描述	输入/输出
muxGrp	MUX GROUP 号。 取值范围：[0, MUX_MAX_GRP_NUM)。	输入
muxChn	MUX chn 号。 取值范围：[0, MUX_MAX_CHN_NUM)。	输入
pChnAttr	mux 通道属性指针	输入
nFallocateLen	文件长度	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【注意】

无。

【举例】

无。

AW_MPI_MUX_RegisterCallback

【描述】

设置 muxgroup 的回调函数。

【语法】

ERRORTYPE AW_MPI_MUX_RegisterCallback(MUX_GRP muxGrp, MPPCallbackInfo *pCallback);

【参数】

参数名称	描述	输入/输出
muxGrp	MUX GROUP 号。 取值范围：[0, MUX_MAX_GRP_NUM)。	输入
pCallback	回调参数指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【注意】

无。

【举例】

无。

7.4. 数据类型

MUX_GRP_ATTR_S

【说明】

mux group 属性数据结构体。

【定义】

```
typedef struct MUX_GRP_ATTR_S
{
    // video
    int mHeight;
    int mWidth;
    int mVideoFrmRate;    // *1000
    int mCreateTime;
    int mMaxKeyInterval;
    PAYLOAD_TYPE_E mVideoEncodeType;     //VENC_CODEC_H264
```

```
    int mRotateDegree;    //0, 90, 180, 270

    // audio
    int mChannels;
    int mBitsPerSample;
    int mSamplesPerFrame; //sample_cnt_per_frame
    int mSampleRate;
    PAYLOAD_TYPE_E mAudioEncodeType;    //AUDIO_ENCODER_AAC_TYPE
}MUX_GRP_ATTR_S;
```

【成员】

成员名称	描述
mHeight	视频高度
mWidth	视频宽度
mVideoFrmRate	视频帧率
mCreateTime	创建时间
mMaxKeyInterval	关键帧间隔
mVideoEncodeType	视频编码类型
mRotateDegree	旋转角转，取值范围：[0,90, 180,270]
mChannels	音频通路数，取值范围：[1,2]
mBitsPerSample	音频采样深度，取值范围：[16]
mSamplesPerFrame	音频每帧采样数，取值范围：[1024]
mSampleRate	音频采样率，取值范围：8000, 16000, 44100, 48000。
mAudioEncodeType	音频编码类型

【注意事项】

无。

【相关数据类型及接口】

无。

MUX_CHN_ATTR_S

【说明】

muxChannel 属性参数结构体。

【定义】

```
typedef struct MUX_CHN_ATTR_S
```

```
{
    int mMuxerId;
    MEDIA_FILE_FORMAT_E mMediaFileFormat;
    int64_t mMaxFileDuration;     //unit:ms
    int64_t mMaxFileSizeBytes;    //unit:byte
    int         mFallocateLen;
    BOOL        mCallbackOutFlag;     //send data through callback.
    FSWRITEMODE mFsWriteMode;
    int             mSimpleCacheSize;
}MUX_CHN_ATTR_S;
```

【成员】

成员名称	描述
mMuxerId	muxer 实例的 id 号，由调用者指定，同一 muxerGroup 下的 muxerChannel 的 muxerId 号不能重复。
mMediaFileFormat	文件封装类型
mMaxFileDuration	文件最大持续时间（ms）
mMaxFileSizeBytes	文件最大长度
mFallocateLen	文件预分配长度
mCallbackOutFlag	是否回调输出（用于网络传输），如果为 TRUE，则不写文件系统，通过 callback 输出数据。
mFsWriteMode	写卡操作方式类型，只支持 FSWRITEMODE_SIMPLECACHE 和 FSWRITEMODE_DIRECT。
mSimpleCacheSize	simpleCahce 写卡模式下的 cache 大小。

【注意事项】

本地写卡时支持 2 种写卡方式

【相关数据类型及接口】

无。

CdxFdT

【说明】

CdxFdT 参数结构体。

【定义】

```
typedef struct CdxFdT
```

```
{
    int mFd;
    int mnFallocateLen;
    int mIsImpact;
    int mMuxerId;
}CdxFdT;
```

【成员】

成员名称	描述
mFd	文件句柄
mnFallocateLen	文件预分配长度，单位：字节。
mIsImpact	是否是碰撞文件
mMuxerId	muxid

【注意事项】

无。

【相关数据类型及接口】

7.5. 错误码

错误代码	宏定义	描述
0xA0658002	ERR_MUX_INVALID_CHNID	通道 ID 超出合法范围
0xA0658003	ERR_MUX_ILLEGAL_PARAM	参数超出合法范围
0xA0658004	ERR_MUX_EXIST	试图申请或者创建已经存在的设备、通道或者资源
0xA0658006	ERR_MUX_NULL_PTR	函数参数中有空指针
0xA0658007	ERR_MUX_NOT_CONFIG	使用前未配置
0xA0658008	ERR_MUX_NOT_SUPPORT	不支持的参数或者功能
0xA0658009	ERR_MUX_NOT_PERM	该操作不允许，如试图修改静态配置参数
0xA0658005	ERR_MUX_UNEXIST	试图使用或者销毁不存在的设备、通道或者资源
0xA065800C	ERR_MUX_NOMEM	分配内存失败，如系统内存不足

0xA065800D	ERR_MUX_NOBUF	分配缓存失败，如申请的数据缓冲区太大
0xA0658010	ERR_MUX_SYS_NOTREADY	系统没有初始化或没有加载相应模块
0xA0658012	ERR_MUX_BUSY	MUX 系统忙

8. DEMUX 模块

8.1. 概述

demux 模块，即文件解封装模块。本模块支持创建多个 demux 通道，每路通道的解封装过程独立。

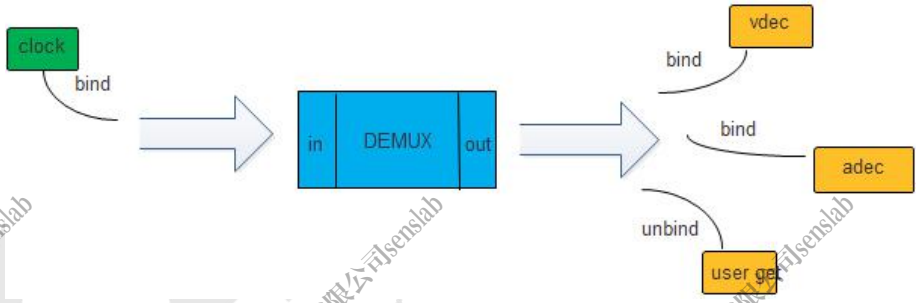
8.2. 功能描述与使用

demux 通道内部线程完成读媒体文件，分包功能。

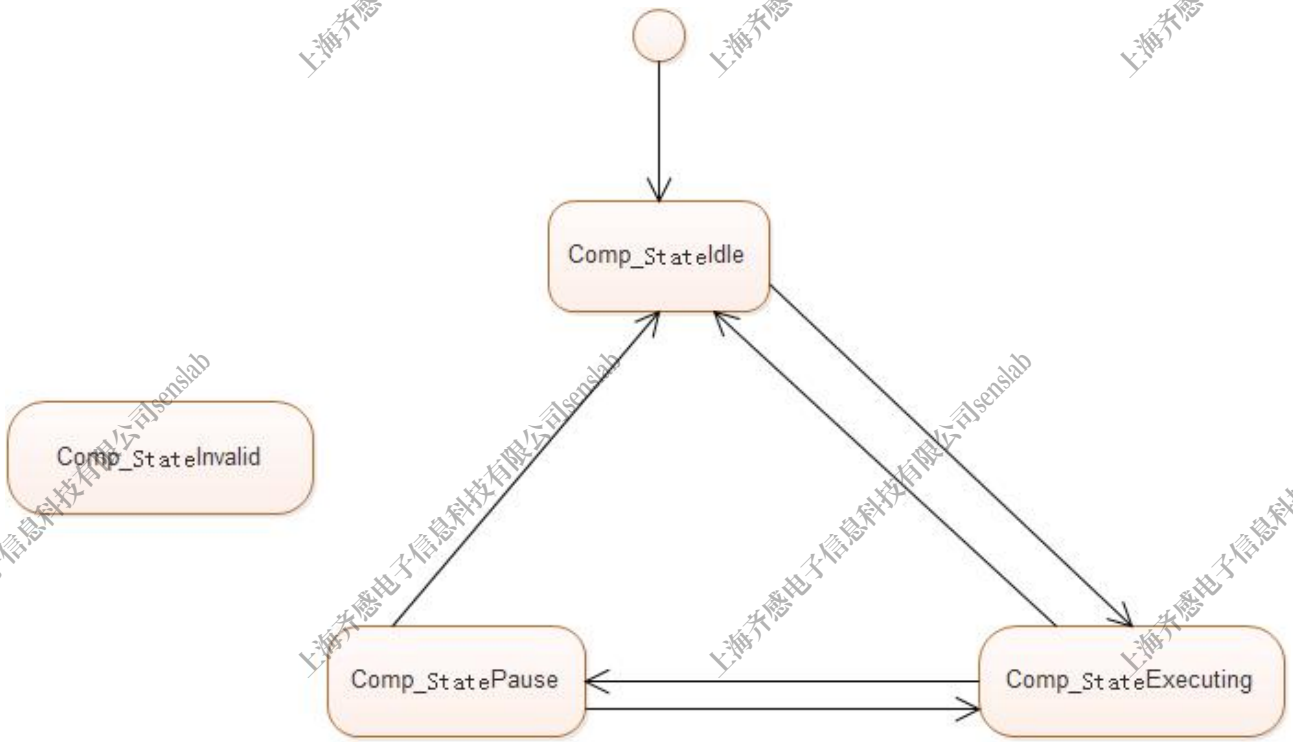
demux 模块输入、输出端口均支持绑定与绑定非方式。输入输出端口连接支持情况如下图所示。输入端口最多支持 1 路 clock 输入绑定，输出端口最多支持 2 路输出绑定（vdec/adec）。

demux 通道支持只解析视频包或者只解析音频包;支持 seek 解析。

注意事项:输出绑定方式时,demux 模块本身不分配存放解析包所需要的内存，相应内存由 demux 输出端绑定端口 vdec/adec 模块分配。



8.2.1. 状态图



demux 组件内部状态设定为：

- COMP_StateLoaded：组件初始创建状态。
- COMP_StateIdle：组件完成初始化，参数设置、资源配置完毕，随时可以运行的状态。
- COMP_StateExecuting：运行状态。
- COMP_StatePause：暂停状态。
- COMP_StateInvalid：异常状态。

AW_MPI_DEMUX_CreateChn() 的实现过程会经过 COMP_StateLoaded 状态，到达 COMP_StateIdle。

组件内部状态转换的函数是：SendCommand(..., COMP_CommandStateSet, 目标 COMP_State, ...);

8.2.2. API 和状态

能够引起状态变化的 API，见状态转换图。

每个 API 只能在允许的状态下调用，如果不在允许的状态下调用 API，则无效。

API 列表如下：(允许被调用的状态栏填写 Y)

	COMP_StateIdle	COMP_Pause	COMP_StateExecuting	COMP_StateInvalid	说明
AW_MPI_DEMUX_CreateChn					创建 demux 通道
AW_MPI_DEMUX_DestroyChn	Y	Y	Y	Y	销毁 demux 通道
AW_MPI_DEMUX_RegisterCallback					设置通道回调
AW_MPI_DEMUX_SetChnAttr	Y		Y		设置通道属性
AW_MPI_DEMUX_GetChnAttr	Y		Y		获取通道属性
AW_MPI_DEMUX_GetMediaInfo	Y	Y	Y		获取解析出来的媒体信息
AW_MPI_DEMUX_Start	Y	Y			开始解析
AW_MPI_DEMUX_Stop		Y	Y		停止解析
AW_MPI_DEMUX_Pause			Y		暂信解析

AW_MPI_DEMUX_ResetChn	Y				重置通道
AW_MPI_DEMUX_Seek		Y	Y		跳转
AW_MPI_DEMUX_getDmxOutPutBuf			Y		取解析出来的数据包 buffer(非绑定模式)
AW_MPI_DEMUX_releaseDmxBuf			Y		还 buffer(与 AW_MPI_DEMUX_getDmxOutPutBuf 成对使用)
AW_MPI_DEMUX_DisableTrack	Y				
AW_MPI_DEMUX_DisableMediaTrack	Y				

8.3. API 参考

demux 模块主要提供 demux 通道(在本文档中通道等同于组件实例)的创建和销毁、通道的复位、开启和停止等功能。

AW_MPI_DEMUX_CreateChn

【描述】

创建 demux 通道

【语法】

ERRORTYPE AW_MPI_DEMUX_CreateChn(PARAM_IN DEMUX_CHN dmxChn, PARAM_IN const DEMUX_CHN_ATTR_S *pstAttr);

【参数】

参数名称	描述	输入/输出
dmxChn	demux 通道号。 取值范围：[0, DEMUX_MAX_CHN_NUM)。	输入
pstAttr	demux 通道属性指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【需求】

- 头文件：mm_comm_demux.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

AW_MPI_DEMUX_DestroyChn

【描述】

销毁 demux 通道

【语法】

ERRORTYPE AW_MPI_DEMUX_DestroyChn(PARAM_IN DEMUX_CHN dmxChn);

【参数】

参数名称	描述	输入/输出

dmxChn	demux 通道号。 取值范围：[0, DEMUX_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【需求】

- 头文件：mm_comm_demux.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

AW_MPI_DEMUX_RegisterCallback

【描述】

设置 demux 通道回调。

【语法】

ERRORTYPE AW_MPI_DEMUX_RegisterCallback(PARAM_IN DEMUX_CHN dmxChn, PARAM_IN MPPCallbackInfo *pCallback);

【参数】

参数名称	描述	输入/输出
dmxChn	demux 通道号。 取值范围：[0, DEMUX_MAX_CHN_NUM)。	输入
pCallback	回调参数指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【需求】

- 头文件：mm_comm_demux.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

AW_MPI_DEMUX_SetChnAttr

【描述】

设置 demux 通道属性。

【语法】

ERRORTYPE AW_MPI_DEMUX_SetChnAttr(PARAM_IN DEMUX_CHN dmxChn, PARAM_IN DEMUX_CHN_ATTR_S *pAttr);

【参数】

参数名称	描述	输入/输出
dmxChn	demux 通道号。 取值范围：[0, DEMUX_MAX_CHN_NUM)。	输入
pAttr	属性批针	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【需求】

- 头文件：mm_comm_demux.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

AW_MPI_DEMUX_GetChnAttr

【描述】

获取 demux 通道属性。

【语法】

ERRORTYPE AW_MPI_DEMUX_GetChnAttr(PARAM_IN DEMUX_CHN dmxChn, PARAM_OUT DEMUX_CHN_ATTR_S *pstAttr);

【参数】

参数名称	描述	输入/输出
dmxChn	demux 通道号。 取值范围：[0, DEMUX_MAX_CHN_NUM)。	输入
pstAttr	属性指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【需求】

- 头文件：mm_comm_demux.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

AW_MPI_DEMUX_GetMediaInfo

【描述】

获取 demux 通道的媒体文件信息。

【语法】

ERRORTYPE AW_MPI_DEMUX_GetMediaInfo(PARAM_IN DEMUX_CHN dmxChn, PARAM_OUT DEMUX_MEDIA_INFO_S *pMediaInfo);

【参数】

参数名称	描述	输入/输出
dmxChn	demux 通道号。 取值范围：[0, DEMUX_MAX_CHN_NUM)。	输入
pMediaInfo	媒体信息指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【需求】

- 头文件：mm_comm_demux.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

AW_MPI_DEMUX_Start

【描述】

开启 demux 通道，解析文件。

【语法】

ERRORTYPE AW_MPI_DEMUX_Start(PARAM_IN DEMUX_CHN dmxChn);

【参数】

参数名称	描述	输入/输出
dmxChn	demux 通道号。 取值范围：[0, DEMUX_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【需求】

- 头文件：mm_comm_demux.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

- 如果通道未创建，则返回失败。
- 此接口并不判断当前是否已经开始解析，如果已经开始解析，调用此接口也返回成功。

【举例】

无。

AW_MPI_DEMUX_Stop

【描述】

停止 demux 通道解析文件。

【语法】

ERRORTYPE AW_MPI_DEMUX_Stop(PARAM_IN DEMUX_CHN dmxChn);

【参数】

参数名称	描述	输入/输出
dmxChn	demux 通道号。 取值范围：[0, DEMUX_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【需求】

- 头文件：mm_comm_demux.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

- 如果通道未创建，则返回失败。
- 此接口并不判断当前是否已经停止解析，如果已经停止，调用此接口也返回成功。
- 此接口用于 demux 通道停止解析文件，在解码通道销毁或复位前必须停止解析文件。

【举例】

无。

AW_MPI_DEMUX_Pause

【描述】

暂停 demux 通道解析文件。

【语法】

ERRORTYPE AW_MPI_DEMUX_Pause(PARAM_IN DEMUX_CHN dmxChn);

【参数】

参数名称	描述	输入/输出
dmxChn	demux 通道号。 取值范围：[0, DEMUX_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【需求】

- 头文件：mm_comm_demux.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

- 如果通道未创建，则返回失败。
- 此接口并不判断当前是否已经暂停解析，如果已经暂停，调用此接口也返回成功。

【举例】

无。

AW_MPI_DEMUX_ResetChn

【描述】

复位 demux 通道，但不会重置已设置的通道参数。通道复位后，缓冲数据都被清空，随时可以从文件开始处重新解析数据。

【语法】

ERRORTYPE AW_MPI_DEMUX_ResetChn(PARAM_IN DEMUX_CHN dmxChn);

【参数】

参数名称	描述	输入/输出
dmxChn	demux 通道号。 取值范围：[0, DEMUX_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【需求】

- 头文件：mm_comm_demux.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

- Reset 并不存在的通道，返回失败 AW_ERR_VENC_UNEXIST。
- 如果通道没有停止而 reset 通道，则返回失败。

【举例】

无。

AW_MPI_DEMUX_Seek

【描述】

跳转到媒体文件某个时间点解析

【语法】

ERRORTYPE AW_MPI_DEMUX_Seek(PARAM_IN DEMUX_CHN dmxChn, PARAM_IN int msec);

【参数】

参数名称	描述	输入/输出
dmxChn	demux 通道号。 取值范围：[0, DEMUX_MAX_CHN_NUM)。	输入

msec	时间点（ms）	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【需求】

- 头文件：mm_comm_demux.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

会清空已解析 buffer list

【举例】

无。

AW_MPI_DEMUX_getDmxOutPutBuf

【描述】

获取 demux 解析出来的数据包

【语法】

ERRORTYPE AW_MPI_DEMUX_getDmxOutPutBuf(PARAM_IN DEMUX_CHN dmxChn, PARAM_OUT DemuxCompOutputBuffer *pDmxOutBuf, PARAM_IN int nMilliSec);

【参数】

参数名称	描述	输入/输出
dmxChn	demux 通道号。 取值范围：[0, DEMUX_MAX_CHN_NUM)。	输入
pDmxOutBuf	数据包指针	输出
nMilliSec	获取数据的超时时间。 -1 表示阻塞模式； 0 表示非阻塞模式； >0 表示阻塞 s32MilliSec 毫秒，超时则报错返回。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【需求】

- 头文件：mm_comm_demux.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

仅适用于非绑定模式

【举例】

无。

AW_MPI_DEMUX_releaseDmxBuf

【描述】

释放 demux 解析出来的数据包 buffer

【语法】

ERRORTYPE AW_MPI_DEMUX_releaseDmxBuf(PARAM_IN DEMUX_CHN dmxChn, PARAM_OUT DemuxCompOutputBuffer *pDmxOutBuf);

【参数】

参数名称	描述	输入/输出
dmxChn	demux 通道号。 取值范围：[0, DEMUX_MAX_CHN_NUM)。	输入
pDmxOutBuf	数据包指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【需求】

- 头文件：mm_comm_demux.h、mm_common.h
- 库文件：libmedia_mpp.so

【注意】

仅适用于非绑定模式，与 AW_MPI_DEMUX_getDmxOutPutBuf 成对使用

【举例】

无。

8.4. 数据类型

STREAMTYPE_E

【说明】

流媒体类型

【定义】

```
typedef enum STREAMTYPE_E{
    STREAMTYPE_NETWORK,
    STREAMTYPE_LOCALFILE,
}STREAMTYPE_E;
```

【成员】

成员名称	描述
STREAMTYPE_NETWORK	网络 stream。
STREAMTYPE_LOCALFILE	本地文件 stream。

【注意事项】

无。

【相关数据类型及接口】

无。

SOURCETYPE_E

【说明】

源类型

【定义】

```
typedef enum SOURCETYPE_E{
    SOURCETYPE_FD,
    SOURCETYPE_FILEPATH,
    SOURCETYPE_WRITER_CALLBACK = 6, //for recoder writer
}SOURCETYPE_E;
```

【成员】

成员名称	描述
SOURCETYPE_FD	文件来源是 fd。
SOURCETYPE_FILEPATH	文件来源是字符串路径。
SOURCETYPE_WRITER_CALLBACK	不支持。

【注意事项】

无。

【相关数据类型及接口】

无。

CEDARX_MEDIA_TYPE

【说明】

媒体类别

【定义】

```
typedef enum CEDARX_MEDIA_TYPE{
    CEDARX_MEDIATYPE_NORMAL = 0 ,
    CEDARX_MEDIATYPE_RAWMUSIC     ,
    CEDARX_MEDIATYPE_3D_VIDEO     ,
    CEDARX_MEDIATYPE_DRM_VIDEO ,
    CEDARX_MEDIATYPE_DRM_WVM_VIDEO ,
    CEDARX_MEDIATYPE_DRM_ES_BASED_VIDEO,
    CEDARX_MEDIATYPE_DRM_CONTAINNER_BASED_VIDEO,
    CEDARX_MEDIATYPE_BD,
    CEDARX_SOURCE_MULTI_URL,
}CEDARX_MEDIA_TYPE;
```

【成员】

成员名称	描述
CEDARX_MEDIATYPE_NORMAL	
CEDARX_MEDIATYPE_RAWMUSIC	
CEDARX_MEDIATYPE_3D_VIDEO	
CEDARX_MEDIATYPE_DRM_VIDEO	
CEDARX_MEDIATYPE_DRM_WVM_VIDEO	
CEDARX_MEDIATYPE_DRM_ES_BASED_VIDEO	
CEDARX_MEDIATYPE_DRM_CONTAINNER_BASED_VIDEO	
CEDARX_MEDIATYPE_BD	
CEDARX_SOURCE_MULTI_URL	

【注意事项】

无。

【相关数据类型及接口】

无。

DEMUX_DISABLE_TRACKINFO

【说明】

轨道类型

【定义】

```
typedef enum DEMUX_DISABLE_TRACKINFO {
    DEMUX_DISABLE_AUDIO_TRACK      = 0x01,
    DEMUX_DISABLE_VIDEO_TRACK      = 0x02,
    DEMUX_DISABLE_SUBTITLE_TRACK = 0x04,
} DEMUX_DISABLE_TRACKINFO;
```

【成员】

成员名称	描述
DEMUX_DISABLE_AUDIO_TRACK	音频轨道
DEMUX_DISABLE_VIDEO_TRACK	视频轨道
DEMUX_DISABLE_SUBTITLE_TRACK	subtitle

【注意事项】

无。

【相关数据类型及接口】

无。

DEMUX_CHN_ATTR_S

【说明】

通道属性

【定义】

```
typedef struct DEMUX_CHN_ATTR_S
{
    STREAMTYPE_E mStreamType;
    SOURCETYPE_E mSourceType;
    char* mSourceUrl;
    int mFd;
    int mDemuxDisableTrack;      //DEMUX_DISABLE_AUDIO_TRACK
}DEMUX_CHN_ATTR_S;
```

【成员】

成员名称	描述
mStreamType;	stream 类型
mSourceType	文件来源类别
mSourceUrl	url 地址
mFd	文件句柄
mDemuxDisableTrack	禁止解析的轨道类型，DEMUX_DISABLE_TRACKINFO 枚举类型的组合。

【注意事项】

无。

【相关数据类型及接口】

无。

DEMUX_VIDEO_STREAM_INFO_S

【说明】

通道属性

【定义】

```
typedef struct DEMUX_VIDEO_STREAM_INFO_S
{
    PAYLOAD_TYPE_E    mCodecType;
    int    mWidth;    //display width
    int    mHeight;
    int    mFrameRate;    // x1000
}DEMUX_VIDEO_STREAM_INFO_S;
```

【成员】

成员名称	描述
mCodecType	视频编码类型
mWidth	视频图像宽度
mHeight	视频图像高度
mFrameRate	帧率，单位 x1000。

【注意事项】

无。

【相关数据类型及接口】

无。

DEMUX_AUDIO_STREAM_INFO_S

【说明】

通道属性

【定义】

```
typedef struct DEMUX_AUDIO_STREAM_INFO_S
{
    PAYLOAD_TYPE_E     mCodecType;
    int mChannelNum;
    int mBitsPerSample;
    int mSampleRate;
    unsigned char strLang[MAX_LANG_CHAR_SIZE];
}DEMUX_AUDIO_STREAM_INFO_S;
```

【成员】

成员名称	描述
mCodecType	音频编码类型
mChannelNum	音轨的声道数
mBitsPerSample	sample 采样位数
mSampleRate	采样率
strLang	音轨名称

【注意事项】

无。

【相关数据类型及接口】

无。

DEMUX_MEDIA_INFO_S

【说明】

通道属性

【定义】

```
typedef struct DEMUX_MEDIA_INFO_S
{
    unsigned int mDuration; //unit: ms
    int mAudioNum, mAudioIndex;
    int mVideoNum, mVideoIndex;
    int mSubtitleNum, mSubtitleIndex;
```

DEMUX_AUDIO_STREAM_INFO_S
mAudioStreamInfo[DEMUX_MAX_AUDIO_STREAM_NUM];
DEMUX_VIDEO_STREAM_INFO_S
mVideoStreamInfo[DEMUX_MAX_VIDEO_STREAM_NUM];
DEMUX_SUBTITLE_STREAM_INFO_S
mSubtitleStreamInfo[DEMUX_MAX_SUBTITLE_STREAM_NUM];
}DEMUX_MEDIA_INFO_S;

【成员】

成员名称	描述
mDuration	文件时长。单位 ms。
mAudioNum	文件包含的音频轨道数量。
mAudioIndex	选定播放的音频轨道 index，范围 [0, mAudioNum)。
mVideoNum	文件包含的视频轨道数量。
mVideoIndex	选定播放的视频轨道 index，范围 [0, mVideoNum)。
mSubtitleNum	文件包含的字幕轨道数量，不支持。
mSubtitleIndex	选定播放的字幕轨道 index，范围 [0, mSubtitleNum)，不支持。
mAudioStreamInfo	audioTrack 的流信息数组
mVideoStreamInfo	videoTrack 的流信息数组
mSubtitleStreamInfo	subtitleTrack 的流信息数组，未使用。

【注意事项】

无。

【相关数据类型及接口】

无。

DemuxCompOutputBuffer

【说明】

Demux 组件的输出 buffer 数据结构。

【定义】

typedef struct DemuxCompOutputBuffer
{
int nTobeFillLen; // [in] indicate data size.

```
    unsigned int video_stream_type; // [in] indicate VIDEO_TYPE_MAJOR,
CDX_VIDEO_STREAM_MAJOR

    unsigned char* pBuffer;          // [out]
    unsigned char* pBufferExtra;     // [out]
    unsigned int nBufferLen;         // [out] size of the buffer , in bytes
    unsigned int nBufferExtraLen;    // [out] size of the buffer extra, in bytes

    unsigned int nCtrlBits;          // [in]
    unsigned int nFilledLen;         // [in]
    int64_t  nTimeStamp;             // [in]
    int64_t  duration;               // [in] for subtitle
    //unsigned int  subDispEndTime;
    int infoVersion;                 // [in]
    void *pChangedStreamsInfo;       // [in] VideoInfo/AudioInfo/SubtitleInfo
    int media_type; //add by andy for video/audio/subtitle
}DemuxCompOutputBuffer;
```

[in][out]相对于绑定模式下进行隧道链接的 vdec 而言。如果 demux 组件处于非绑定模式，无意义。

【成员】

成员名称	描述
nTobeFillLen	申请的 buffer 长度。
video_stream_type	标记码流编号，针对 3D 双流格式。取值：VIDEO_TYPE_MAJOR。
pBuffer	装填待解码数据的第一段 buffer 起始虚拟地址。
pBufferExtra	装填待解码数据的第二段 buffer 起始虚拟地址。
nBufferLen	第一段 buffer 的长度。
nBufferLenExtraLen	第二段 buffer 的长度
nCtrlBits	指示 buffer 数据的特征，为以下宏的组合。 #define CEDARV_FLAG_PTS_VALID 0x2 #define CEDARV_FLAG_FIRST_PART 0x8 #define CEDARV_FLAG_LAST_PART

	0x10 CEDARV_FLAG_PTS_VALID：是否携带有效的 PTS。 CEDARV_FLAG_FIRST_PART：buffer 数据是否包含一帧的起始。 CEDARV_FLAG_LAST_PART：buffer 数据是否包含一帧的结束。
nFilledLen	buffer 中有效数据的长度。
nTimeStamp	时间戳，单位微秒。
duration	仅用于表示 subtitle 字幕的持续时间，单位微秒。
infoVersion	码流信息版本，取值范围：[0, +∞)，当码流改变时，版本号加 1。
pChangedStreamsInfo	改变后的码流信息。
media_type	码流类型，取值范围：enum CdxMediaTypeE。

【注意事项】

无。

【相关数据类型及接口】

无。

8.5. 错误码

错误代码	宏定义	描述
0xA0648002	ERR_DEMUX_INVALID_CHNID	通道 ID 超出合法范围
0xA0648003	ERR_DEMUX_ILLEGAL_PARAM	参数超出合法范围
0xA0648004	ERR_DEMUX_EXIST	试图申请或者创建已经存在的设备、通道或者资源
0xA0648006	ERR_DEMUX_NULL_PTR	函数参数中有空指针
0xA0648007	ERR_DEMUX_NOT_CONFIG	使用前未配置
0xA0648008	ERR_DEMUX_NOT_SUPPORT	不支持的参数或者功能
0xA0648009	ERR_DEMUX_NOT_PERM	该操作不允许，如试图修改静

		态配置参数
0xA0648005	ERR_DEMUX_UNEXIST	试图使用或者销毁不存在的设备、通道或者资源
0xA064800C	ERR_DEMUX_NOMEM	分配内存失败，如系统内存不足
0xA064800D	ERR_DEMUX_NOBUF	分配缓存失败，如申请的数据缓冲区太大
0xA0648010	ERR_DEMUX_SYS_NOTREADY	系统没有初始化或没有加载相应模块
0xA0648012	ERR_DEMUX_BUSY	VENC 系统忙

9. 音频

9.1. 概述

➢ **音频输入输出**

音频输入输出接口简称为 AIO（Audio Input/Output）接口，用于向下对接 alsa driver 和 Audio Codec，向上提供 api 对接应用程序，完成声音的录制和播放。AIO 接口分为两种类型：输入模式、输出模式。当为输入类型时，称为 AIP，当为输出类型时，称为 AOP。软件中负责抽象音频接口输入功能的单元，称之为 AI 设备；负责抽象输出功能的单元，称之为 AO 设备。一个 AI 设备下可以挂多个 AI 通道(组件)，实现音频数据复用。

AIO 设备向下通过 alsa-lib 与 alsa 内核驱动对接，来进行 pcm 数据交互、播放/采集控制、音量大小/静音状态调整等操作。AI 设备支持板卡 mic 的声音采集，AO 设备支持 lineout 和 hdmi 两种方式输出。

AIO 设备向上对接 AIO 通道，AIO 通道通过各自的输入/输出端接口对接 app。AI、AO 通道为单端口通道，AI 通道提供 outport 用于向 app 输出采集的 pcm 数据，AO 通道提供 input 通道用于 app 向其送 pcm 数据进行音频播放。例如，对于 AI 组件，当 pcm 和 mixer 控件初始化完毕并且开始采集 pcm 数据后，app 可以从 AI 组件 outport 拿 pcm 数据，其 inport 在内部实现固定绑定到音频采集的环型 buffer；对于 AO 组件，app 可以向其 inport 不停地送 pcm 数据，其 outport 内部实现了固定绑定到音频输出设备的环形 buffer。

➢ **音频编码解码**

音频编解码组件简称为 AEnc、ADec，用于压缩/解压缩 pcm 数据。App 可创建多个 AEnc、ADec 通道，各通道可设置不同编解码格式等参数，并且各通道独立运行，互不干扰。

AEnc 编码通道为双端口通道，其 inport 接收外部送来的 pcm 数据，而不关心数据来自哪个模块，例如可来自 AI 通道，也可来自本地磁盘文件；其 outport 用于输出编码码流，可送到 muxer 模块、ADec 模块或本地磁盘文件。

ADec 解码通道也为双端口通道，其 inport 接收压缩的音频码流，也不关心数据来自哪个模块，例如可来自 demuxer 模块，也可来自网络，也可来自 AEnc 模块；其 outport 用于输出 pcm 数据，可送到 AO 通道、AI 通道或本地磁盘文件。

从 AI 组件获取了 pcm 数据后，通常送编码器进行数据压缩，AEnc 组件即为音频编码的设备抽象。通过编码器类型和码率的设置，可以进行多种格式的音频压缩，目前包括以下格式：aac、mp3、adpcm、pcm、g711a、g711u、g726。

AEnc 组件为双端口(inport、outport)组件，即 app 可以操作输入端口和输出端口的数据，如向 AEnc 输入端口送 pcm 数据，编码完成后，app 再从 AEnc 输出端口取压缩后的音频码流数据。

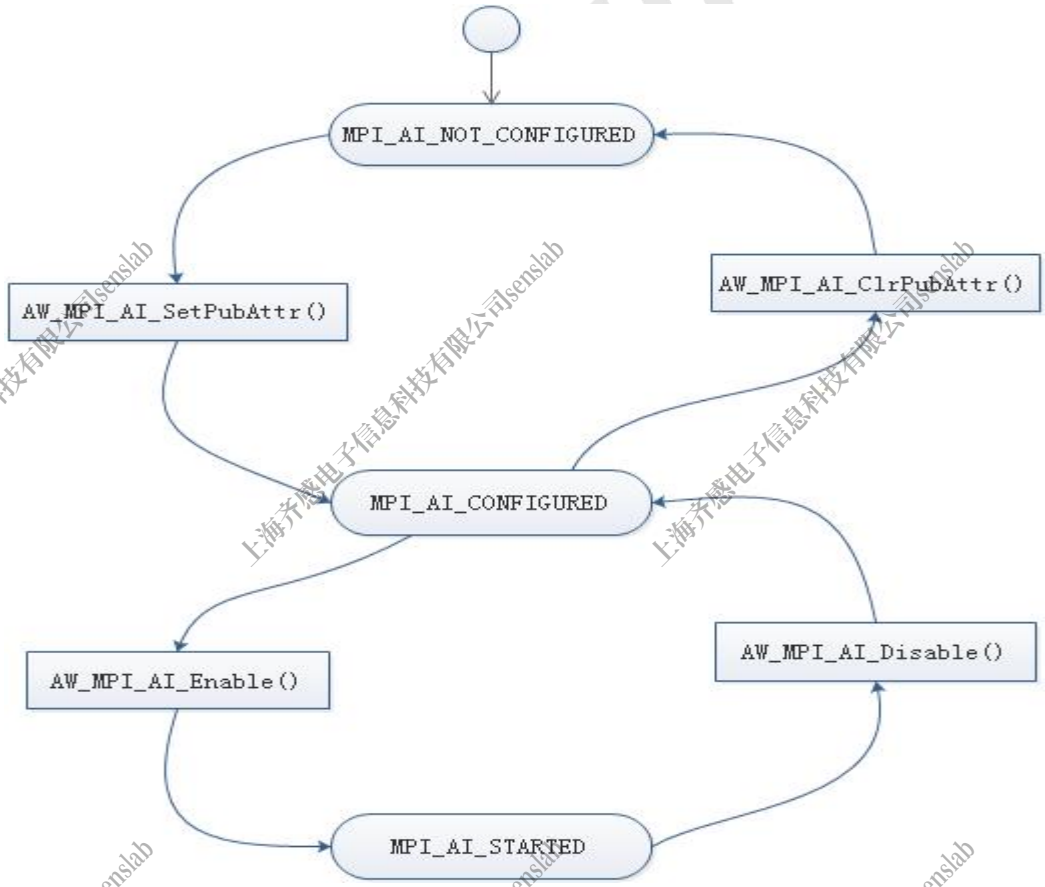
从 AI 通道到 AEnc 通道进行 pcm 数据传递可以分为 tunnel 方式(又称绑定方式)和 non-tunnel 方式(又称非绑定方式)，tunnel 方式为组件之间内部自动传递数据方式，app 不用参与数据传递过程，而只需专注于应用层面的业务开发；non-tunnel 方式为 app 主动管理 pcm 数据，即从 AI 组件拿数据，然后将其送给 AEnc 组件，再从 AEnc 组件输出端口拿编码后的数据，之后将其送 muxer 组件或网传后，最后再利用该包数据向 AEnc 组件还帧。

通过网络或文件解封装后得到的音频码流数据，需要送解码器进行解码以恢复为原始音频数据，ADec 组件即是对该功能的抽象。目前其支持的解码类型与编码类型相同。与 AEnc 组件类似，其也为双端口组件，数据传递也包括 tunnel 方式和 non-tunnel 方式。其实现的功能与 AEnc 正好相反，不再累述。

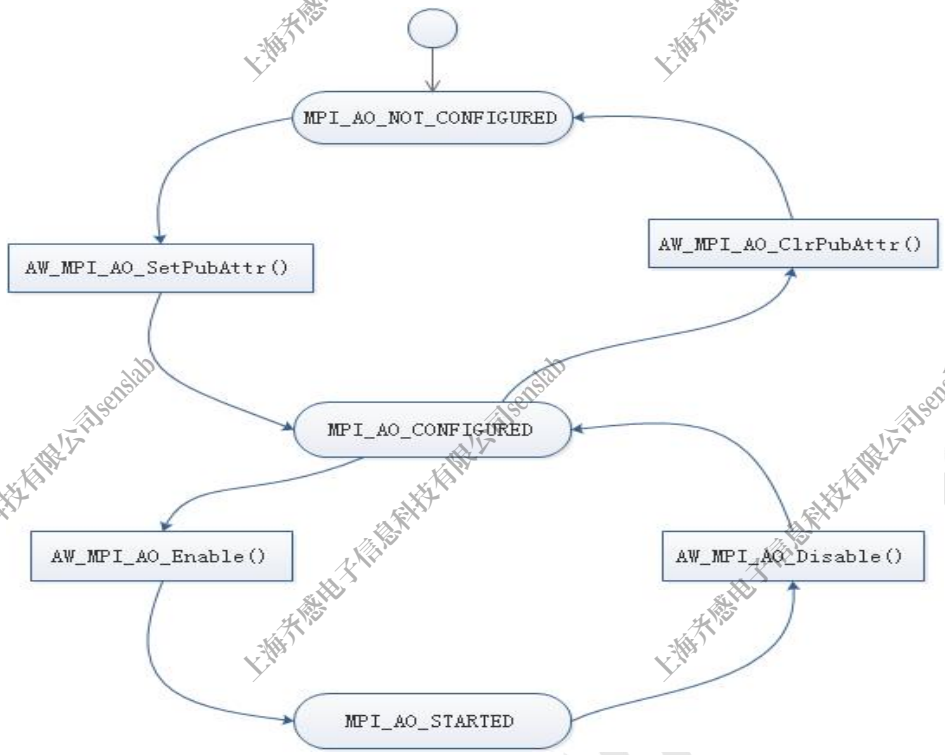
9.2. 功能描述

AIO 设备、AI 通道、AO 通道、AEnc 通道、ADec 通道内部都有一个线程，其按状态机的方式工作，其状态转换图如下：

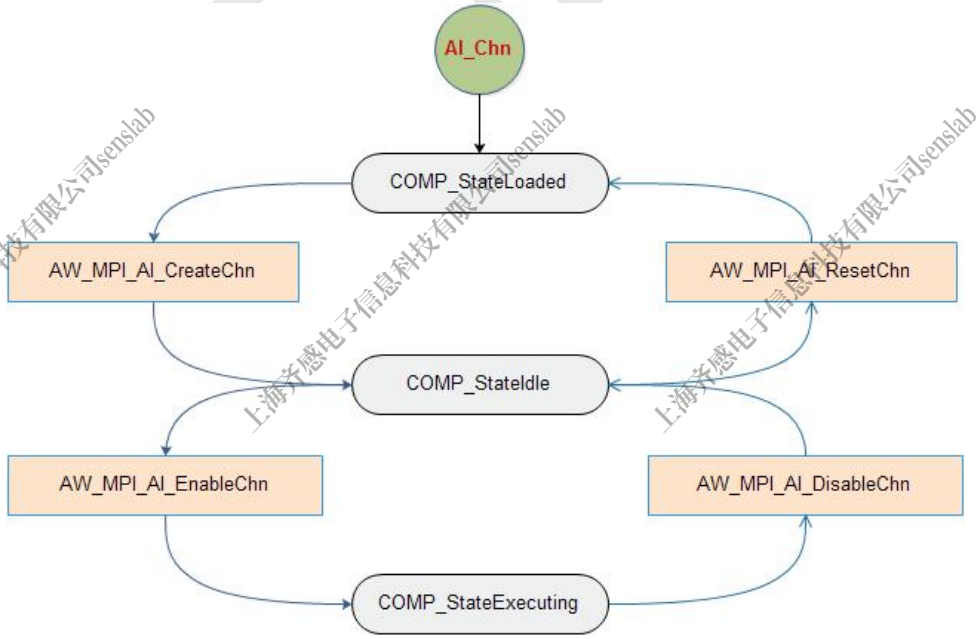
9.2.1. AI 设备状态图



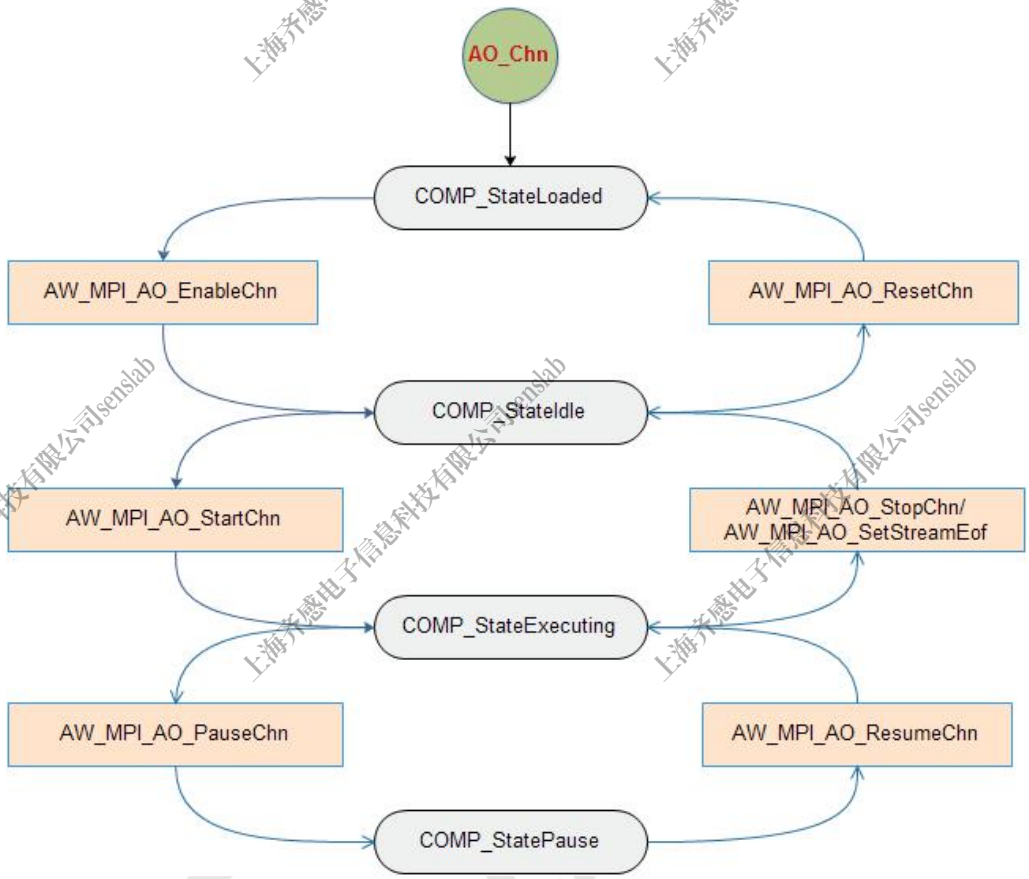
9.2.2. AO 设备状态图



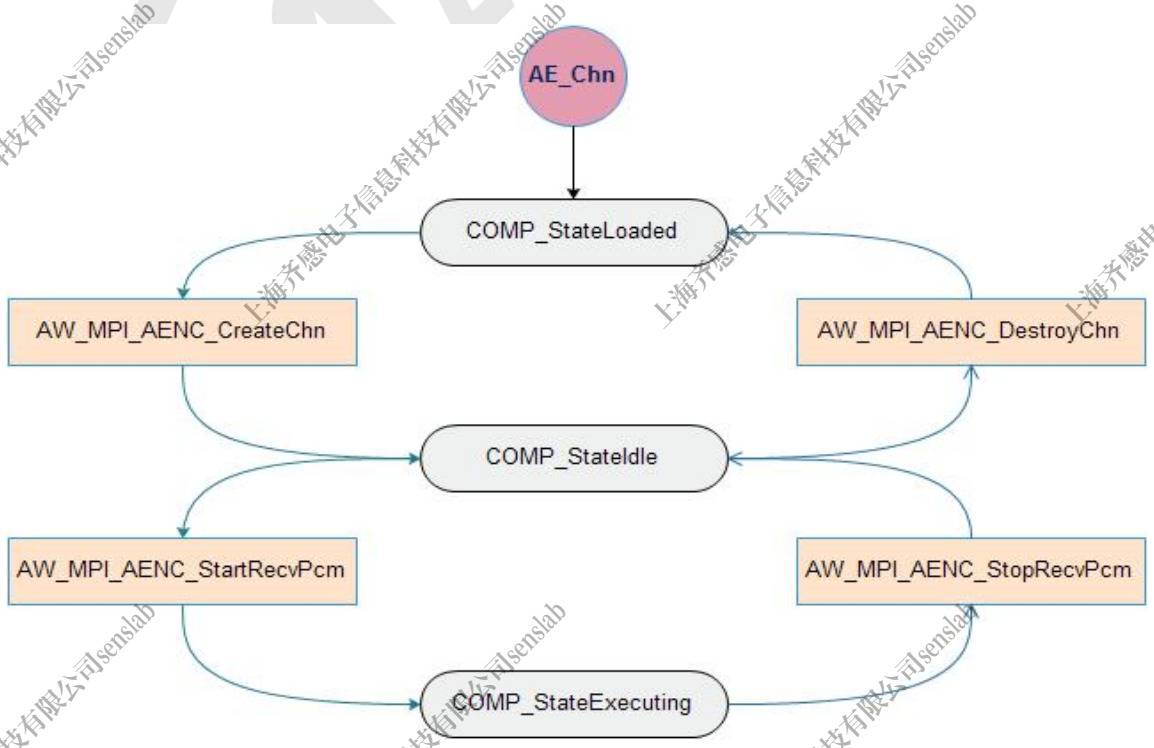
9.2.3. AI 通道状态图



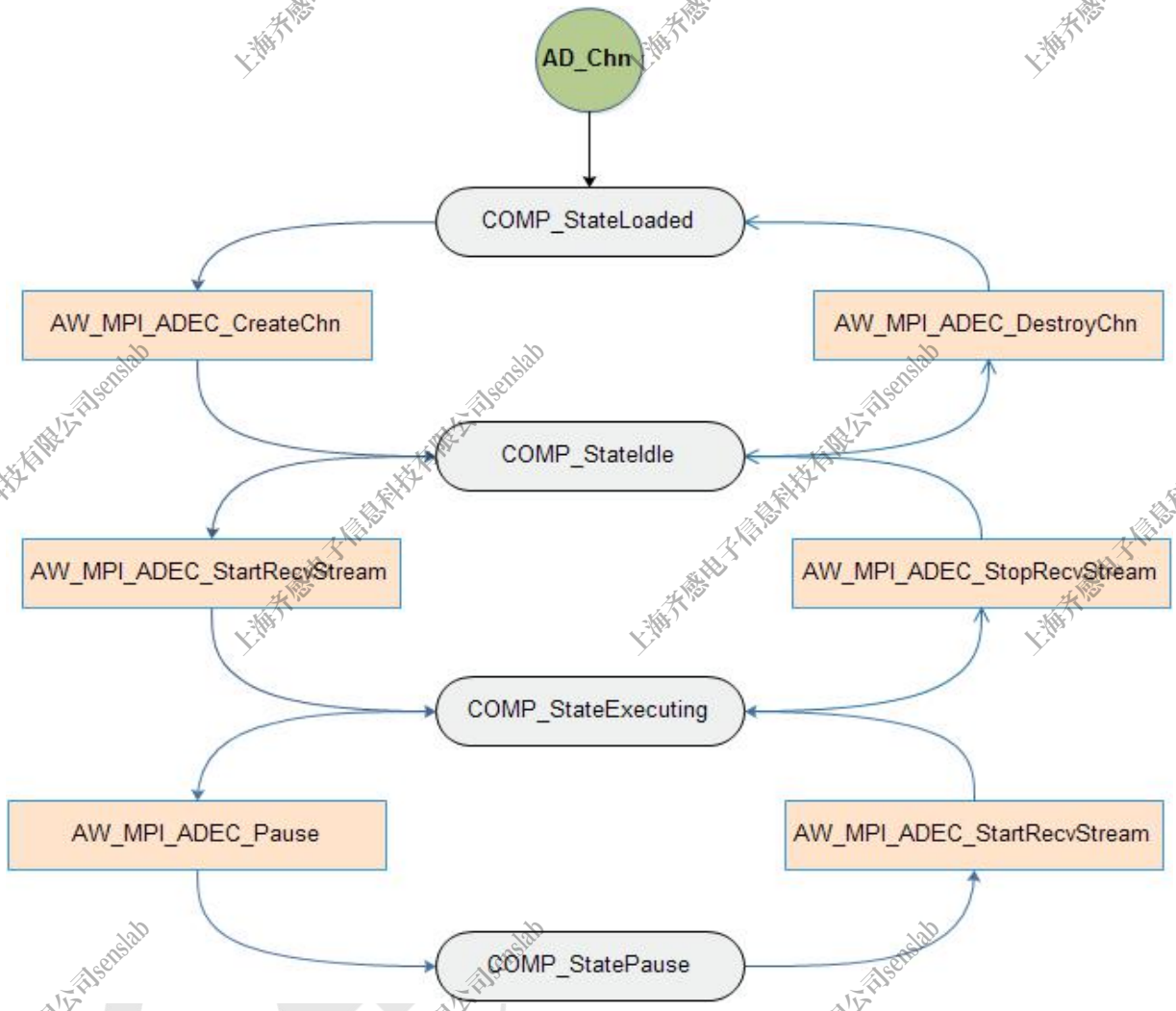
9.2.4. AO 通道状态图



9.2.5. AEnc 通道状态图



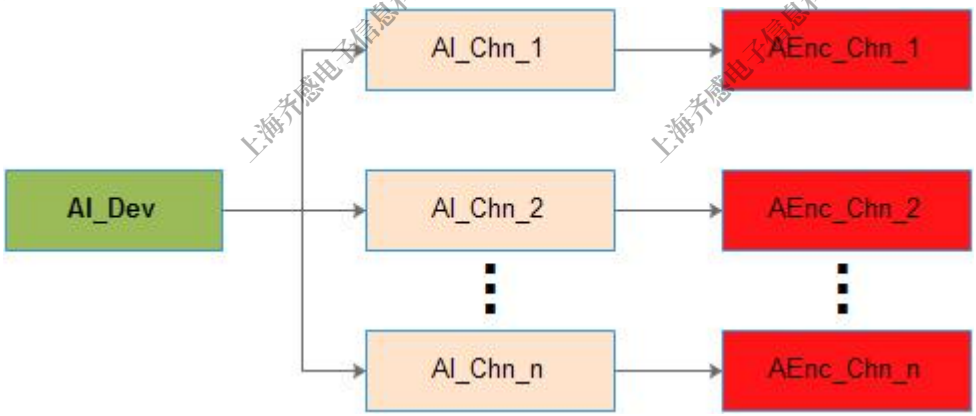
9.2.6. ADec 通道状态图



9.2.7. AIO 设备与通道

设备：从软硬件上划分，其属于硬件的范畴，由 IC 设计决定其数量。

通道：属于软件上的虚拟范畴，可存在多个，即一个设备下可挂多个通道实例。例如，在多路录制时，一个 AI 设备下挂多个 AI 虚拟通道，则每个通道的输入数据都相同，因为其是由同一个音频硬件采集得到的；每个 AI 通道后面又绑定其对应的 AEnc 通道，通过对 AEnc 设置不同的编码格式，可得到多路输入音频相同但编码格式不同的输出码流。如下为其拓扑图：



注意，目前 mpp 在音频输出上只支持一个 AO 设备下挂一个 AO 通道，暂不支持多路 AO 通道同时存在的混音功能。

mpi 通过提供 ai/ao 通道的控制接口，间接地操作 ai/ao 设备。在应用程序开发时，通过操作 ai/ao 通道提供的接口，可完成 ai/ao 设备使能、去使能等操作。

9.3. 音频接口调用流程介绍

使用 mpp 进行音频的应用程序开发时，需特别注意接口调用流程及顺序，否则容易出现调用失败、段错误、帧节点回收出错等问题。

通常情况下，AI/AO 通道调用流程为：设备属性设置、设备使能、通道创建、通道使能、送 pcm 数据、停止通道、复位通道、销毁通道、设备去使能。

AEnc/ADec 通道调用流程类似：通道创建、启动通道、送/还 pcm 数据或 stream 数据、停止接收 pcm 或 stream、复位通道、销毁通道。

9.3.1. AI 通道使用流程

进行音频采集，需打开 ai 设备、创建 ai 通道后才能进行音频 pcm 的采集，参考例子为 sample_ai.c，其流程如下：

```
Step1:AW_MPI_AI_SetPubAttr()     //设置 ai 设备的 pcm 采集参数
Step2:AW_MPI_AI_Enable()         //启动 ai 设备，目前可不调用
Step3:AW_MPI_AI_CreateChn()      //创建 ai 通道，并将其挂到 ai 设备下
Step4:AW_MPI_AI_EnableChn()      //运行 ai 通道
//AI 设备往 AI 通道源源不断送 pcm，App 拿数据/还帧
loop
{
Step5:AW_MPI_AI_GetFrame()       //app 去拿 pcm 数据
Step6:AW_MPI_AI_ReleaseFrame() //app 还帧给 ai 通道
}
Step7:AW_MPI_AI_DisableChn()     //停止 ai 通道接收数据
```

Step8:AW_MPI_AI_ResetChn()　　//ai 通道复位

Step9:AW_MPI_AI_DestroyChn()　　//销毁 ai 通道，并从 ai 设备下退出

Step10:AW_MPI_AI_Disable()　　//停止 ai 设备，目前可不调用

其中，step2 的接口目前可不调用，其已经做到了 step3 的接口中，其作用为在创建通道时根据需要来打开 AI 设备。当 AI 通道在第一次调用 AW_MPI_AI_CreateChn()时，其实现中会主动调用 AW_MPI_AI_Enable()打开 AI 设备，第二个以及后面的通道在调用 AW_MPI_AI_CreateChn()创建通道时，其内部实现则不需再次打开 AI 设备，而是直接单纯的创建组件。

其中，step9 的 AW_MPI_AI_DestroyChn()会将该通道从 AI 设备所管理的多个通道列表中删除（即使 ai 设备又采集到 pcm 数据，但不会再往本通道传递了，而会送往其它 AI 通道），接下来释放本通道的全部资源。

其中，step10 的 AW_MPI_AI_Disable()的接口可不调用，因其已经做到了 step9 的接口实现中。当销毁通道(AW_MPI_AI_DestroyChn)时，AI 通道从 AI 设备下删除，若 AI 设备下只包含一个 AI 通道，则在销毁通道的同时会停止 AI 设备，即最后一个通道在销毁通道时会连带关闭 AI 设备。那么，当最后一个通道调用 AW_MPI_AI_DestroyChn()时，其内部实现会自动关闭 AI 设备，用户无需手动调用 Step10 的 AW_MPI_AI_Disable()。

9.3.2. AO 通道使用流程

进行音频播放，需打开 ao 设备、创建 ao 通道后才能进行音频 pcm 数据的播放，参考例子为 sample_ao.c，其流程如下：

Step1:AW_MPI_AO_SetPubAttr()　　//设置 ao 设备的 pcm 参数

Step2:AW_MPI_AO_Enable()　　//启动 ao 设备，可不调用

Step3:AW_MPI_AO_EnableChn()　　//创建 ao 通道，并将其挂到 ao 设备下

Step4:AW_MPI_AO_StartChn()　　//运行 ao 通道

//用户往 AO 通道源源不断送 pcm

loop

{

Step5:AW_MPI_AO_SendFrame()　　//app 手动送 pcm 数据到 ao 通道

}

Step7:AW_MPI_AO_StopChn()　　//停止 ao 通道

Step8:AW_MPI_AO_DisableChn()　　//销毁 ao 通道，并关闭 ao 设备

Step9:AW_MPI_AO_Disable()　　//停止 ao 设备，可不调用

其中，step2 的 AW_MPI_AO_Enable()目前可不调用，其已经在 step3 的 AW_MPI_AO_EnableChn()接口实现中被调用了。

其中，step8 的 AW_MPI_AO_DisableChn()会将本通道从 AO 设备下删除，并关闭 AO 设备，接下来释放本通道的全部资源。即该接口内部实现中已经调用了 step9 的 AW_MPI_AO_Disable()接口，

用户在使用 AO 做应用开发时可不调用该接口。

9.3.3. AEnc 通道调用流程

进行音频编码，需按照通道创建、启动通道、送 pcm 数据、取 stream 数据、还帧、停止通道、复位通道、销毁通道的顺序使用，参考例子为 sample_aenc.c，其流程如下：

```
Step1:AW_MPI_AEnc_CreateChn()          //创建通道
Step2:AW_MPI_AEnc_StartRecvPcm()       //启动通道
//non-tunnel 方式下用户往 AEnc 通道源源不断送 pcm、取 stream、还 stream。。。
loop
{
Step3:AW_MPI_AEnc_SendFrame()          //app 手动送 pcm 数据到 aenc 通道
Step4:AW_MPI_AEnc_GetStream()          //app 手动拿编码数据
Step4:AW_MPI_AEnc_ReleaseStream()      //app 手动还帧到编码库
}
Step5:AW_MPI_AEnc_StopRecvPcm()        //停止通道
Step6:AW_MPI_AEnc_ResetChn()           //复位通道
Step7:AW_MPI_AEnc_DestroyChn()         //销毁通道
```

注意，在 non-tunnel 方式下，应用程序需手动方式送 pcm 数据到 aenc 通道，组件内部线程对 pcm 数据进行编码，生成的 stream 在输出队列中进行管理，等待用户取走；当用户取走 stream 后，用户还需将该帧码流还给编码通道，以释放其占用的 buffer。在 tunnel 方式下，内部数据通路进行了动态绑定(AI-AEnc)，应用无需手动调用送帧、取帧、还帧的接口，因其在内部实现中自动被调用。

9.3.4. ADec 通道调用流程

进行音频解码，需按照通道创建、启动通道、送 stream 数据、取 pcm 数据、还帧、停止通道、复位通道、销毁通道的顺序使用，参考例子为 sample_adec.c，其流程如下：

```
Step1:AW_MPI_ADec_CreateChn()          //创建通道
Step2:AW_MPI_ADec_StartRecvStream()    //启动通道
//non-tunnel 方式下用户往 ADec 通道源源不断送 stream、取 pcm、还 pcm。。。
loop
{
Step3:AW_MPI_ADec_SendStream()         //app 手动送 stream 数据到 adec 通道
Step4:AW_MPI_ADec_GetFrame()           //app 手动拿 pcm 数据
Step4:AW_MPI_ADec_ReleaseFrame()       //app 手动还帧到解码库
}
Step5:AW_MPI_ADec_StopRecvStream()     //停止通道
```

Step6:AW_MPI_ADec_ResetChn() //复位通道

Step7:AW_MPI_ADec_DestroyChn() //销毁通道

注意，在 non-tunnel 方式下，应用程序需手动方式送 stream 数据到 adec 通道，组件内部线程对 stream 数据进行解码，生成的 pcm 在输出队列中进行管理，等待用户取走；当用户取走 pcm 后，用户还需将该帧 pcm 数据还给解码通道，以释放其占用的 buffer。在 tunnel 方式下，内部数据通路进行了动态绑定(ADec-AO)，应用无需手动调用送帧、取帧、还帧的接口，因其在内部实现中自动被调用。

9.4. API 接口

9.4.1. 音频输入

AW_MPI_AI_SetPubAttr

【目的】

设置 AI 设备属性。

【语法】

ERRORTYPE AW_MPI_AI_SetPubAttr(AUDIO_DEV AudioDevId, const AIO_ATTR_S *pstAttr);

【参数】

参数	描述	
AudioDevId	音频设备号。	输入
pstAttr	AI 设备属性指针。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

无。

【举例】

无。

AW_MPI_AI_GetPubAttr

【目的】

获取 AI 设备属性。

【语法】

ERRORTYPE AW_MPI_AI_GetPubAttr(AUDIO_DEV AudioDevId, AIO_ATTR_S *pstAttr);

【参数】

参数	描述	
AudioDevId	音频设备号。	输入
pstAttr	AI 设备属性指针。	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

无。

【举例】

无。

AW_MPI_AI_Enable

【目的】

启用 AI(音频采集)设备。

【语法】

ERRORTYPE AW_MPI_AI_Enable(AUDIO_DEV AudioDevId);

【参数】

参数	描述	
AudioDevId	音频设备号。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 必须在启用前配置音频设备属性，否则返回属性未配置错误。
- 如果音频设备已经处于启用状态，则直接返回成功。此种场景常见于多路音视频同时录制时，最早一路已经开始采集声音数据后，又创建一个 recorder 进行音视频录制。多个 AI 组件实例同时运行时，音频数据复用，即硬件采集到的每帧 pcm 数据分别送往不同 AI 组件实例，又各自送往其对应的 AEnc 组件实例。

【举例】

无。

AW_MPI_AI_Disable

【目的】

禁用 AI(音频采集)设备。

【语法】

ERRORTYPE AW_MPI_AI_Disable(AUDIO_DEV AudioDevId);

【参数】

参数	描述	
AudioDevId	音频设备号。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 如果音频采集设备已经处于禁用状态，则直接返回成功。
- 禁用音频设备前必须先禁用该设备下已启用的所有 AI 通道。
- 要求在禁用 AI 设备之前，先禁用与之关联、使用 AI 的音频数据的 AENC 通道和 AO 设备，否则可能导致该接口调用失败。

【举例】

多路音视频录制时，最先结束录制的 recorder 在禁用音频设备时，会遍历使用该设备的 AI 通道。如果还有其它 AI 通道在使用该音频设备，则只关闭该 AI 通道而不关闭该音频设备就直接返回；最后一个 recorder 退出时，由于音频采集设备只被该 recorder 的 AI 通道占用，因此直接关闭 AI 通道和音频设备。

综上有，音频采集设备在被共享时，只有等到最后一次被禁用时才能真正释放。

AW_MPI_AI_CreateChn

【目的】

创建 AI 通道。

【语法】

ERRORTYPE AW_MPI_AI_CreateChn(AUDIO_DEV AudioDevId, AI_CHN AiChn);

【参数】

参数	描述	
AudioDevId	音频设备号。	输入
AiChn	AI 通道号	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 创建 AI 通道前，必须保证先启用其所属的 AI 设备，否则返回设备未启动的错误码。
- 创建 AI 通道后才能调用启用通道(AW_MPI_AI_EnableChn)接口。

【举例】

无。

AW_MPI_AI_DestroyChn

【目的】

销毁 AI 通道。

【语法】

ERRORTYPE AW_MPI_AI_DestroyChn(AUDIO_DEV AudioDevId, AI_CHN AiChn);

【参数】

参数	描述	
AudioDevId	音频设备号。	输入
AiChn	AI 通道号	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 销毁 AI 通道前，必须保证该 AI 通道已被复位。
- 录音结束时的操作步骤：DisableChn->ResetChn->DestroyChn。

【举例】

无。

AW_MPI_AI_ResetChn

【目的】

复位 AI 通道。

【语法】

ERRORTYPE AW_MPI_AI_ResetChn(AUDIO_DEV AudioDevId, AI_CHN AiChn);

【参数】

参数	描述	
AudioDevId	音频设备号。	输入

AiChn	AI 通道号	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 录音结束时的操作步骤：DisableChn->ResetChn->DestroyChn。

【举例】

无。

AW_MPI_AI_PauseChn

【目的】

暂停 AI 通道。

【语法】

ERRORTYPE AW_MPI_AI_PauseChn(AUDIO_DEV AudioDevId, AI_CHN AiChn);

【参数】

参数	描述	
AudioDevId	音频设备号。	输入
AiChn	AI 通道号	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 用于录音过程中暂停本通道的声音采集，若同时存在其它 AI 通道，并不影响其它

通道的声音采集。

【举例】

无。

AW_MPI_AI_ResumeChn

【目的】

恢复 AI 通道状态至运行态。

【语法】

ERRORTYPE AW_MPI_AI_ResumeChn(AUDIO_DEV AudioDevId, AI_CHN AiChn);

【参数】

参数	描述	
AudioDevId	音频设备号。	输入
AiChn	AI 通道号	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 将 AI 通道从暂停态转换为运行态，需配合 AW_MPI_AI_PauseChn 接口使用。

【举例】

无。

AW_MPI_AI_EnableChn

【目的】

启用 AI 通道。

【语法】

ERRORTYPE AW_MPI_AI_EnableChn(AUDIO_DEV AudioDevId, AI_CHN AiChn);

【参数】

参数	描述	
AudioDevId	音频设备号。	输入
AiChn	AI 通道号	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 启用 AI 通道前，必须先启用其所属的 AI 设备并且该 AI 通道已被创建，否则返回设备未启动的错误码。

【举例】

无。

AW_MPI_AI_DisableChn

【目的】

禁用 AI 通道。

【语法】

ERRORTYPE AW_MPI_AI_DisableChn(AUDIO_DEV AudioDevId, AI_CHN AiChn);

【参数】

参数	描述	
AudioDevId	音频设备号。	输入
AiChn	AI 通道号	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 禁用 AI 通道前，必须保证 AI 设备已启用并且该 AI 通道已处于运行状态，否则返回错误码。

【举例】

无。

AW_MPI_AI_GetFrame

【目的】

获取音频帧。

【语法】

ERRORTYPE AW_MPI_AI_GetFrame(AUDIO_DEV AudioDevId, AI_CHN AiChn, AUDIO_FRAME_S *pstFrm, AEC_FRAME_S *pstAecFrm, int s32MilliSec);

【参数】

参数	描述	
AudioDevId	音频设备号。	输入
AiChn	AI 通道号	输入
pstFrm	音频帧结构体指针。	输出
pstAecFrm	回声抵消参考帧结构体指针。	输出
s32MilliSec	获取数据的超时时间 -1 表示阻塞模式，无数据时一直等待； 0 表示非阻塞模式，无数据时则报错返回； >0 表示阻塞 s32MilliSec 毫秒，超时则报错返回。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 如果 AI 的回声抵消功能已使能，pstAecFrm 不能是空指针；如果 AI 的回声抵

消功能没有使能，pstAecFrm 可以置为空。

- AI 模块会缓存音频帧数据，用于用户态获取。缓存的深度通过 AW_MPI_AI_SetChnParam 接口设定，默认为 0。
- s32MilliSec 的值必须大于等于-1，等于-1 时采用阻塞模式获取数据，等于 0 时采用非阻塞模式获取数据，大于 0 时，阻塞 s32MilliSec 毫秒后，没有数据则返回超时并报错。
- 获取音频帧数据前，必须先使能对应的 AI 设备和 AI 通道。

【举例】

无。

AW_MPI_AI_ReleaseFrame

【目的】

释放音频帧。

【语法】

ERRORTYPE AW_MPI_AI_ReleaseFrame(AUDIO_DEV AudioDevId, AI_CHN AiChn, AUDIO_FRAME_S *pstFrm, AEC_FRAME_S *pstAecFrm);

【参数】

参数	描述	
AudioDevId	音频设备号。	输入
AiChn	AI 通道号	输入
pstFrm	音频帧结构体指针。	输入
pstAecFrm	回声抵消参考帧结构体指针。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 如果不需要释放回声抵消参考帧，pstAecFrm 置为 NULL 即可。
- 当 app 取走 pcm 数据后，调用该 api 可以释放 AI 组件的 pcmBufferManager 缓冲

队列中的对应的数据帧，释放出空间以便于存储采集到的最新的 pcm 数据。

【举例】

无。

AW_MPI_AI_SetChnParam

【目的】

设置 AI 通道属性。

【语法】

ERRORTYPE AW_MPI_AI_SetChnParam(AUDIO_DEV AudioDevId, AI_CHN AiChn, AI_CHN_PARAM_S *pstChnParam);

【参数】

参数	描述	
AudioDevId	音频设备号。	输入
AiChn	AI 通道号	输入
pstChnParam	音频通道参数结构体指针。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 通道参数目前只有一个成员变量，用于设置用户获取音频帧的缓存深度，默认深度为 0。该成员变量的值不能大于 30。
- 建议先调用 AW_MPI_AI_GetChnParam 接口获取默认配置，再调用本接口修改配置，以便于后续扩展。

【举例】

无。

AW_MPI_AI_GetChnParam

【目的】

获取 AI 通道属性。

【语法】

ERRORTYPE AW_MPI_AI_GetChnParam(AUDIO_DEV AudioDevId, AI_CHN AiChn, AI_CHN_PARAM_S *pstChnParam);

【参数】

参数	描述	
AudioDevId	音频设备号。	输入
AiChn	AI 通道号	输入
pstChnParam	音频通道参数结构体指针。	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

无。

【举例】

无。

AW_MPI_AI_EnableReSmp

【目的】

启用 AI 重采样。

【语法】

ERRORTYPE AW_MPI_AI_EnableReSmp(AUDIO_DEV AudioDevId, AI_CHN AiChn, AUDIO_SAMPLE_RATE_E enOutSampleRate);

【参数】

参数	描述	
AudioDevId	音频设备号。	输入
AiChn	AI 通道号	输入
enOutSampleRate	音频重采样的输出采样率。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 在启用 AI 通道之后，调用此接口启用重采样功能。
- 允许重复启用重采样功能，但必须保证后配置的属性与之前配置的属性一样。
- 在禁用 AI 通道之后，如果重新启用 AI 通道，并使用重采样功能，需调用此接口重新启用重采样。

【举例】

无。

AW_MPI_AI_DisableReSmp

【目的】

禁用 AI 重采样。

【语法】

ERRORTYPE AW_MPI_AI_DisableReSmp(AUDIO_DEV AudioDevId, AI_CHN AiChn);

【参数】

参数	描述	
AudioDevId	音频设备号。	输入
AiChn	AI 通道号	输入

【返回值】

返回值	描述

0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 不再使用 AI 重采样功能的话，应该调用此接口将其禁用。
- 要求在调用此接口之前，先禁用使用该 AI 设备相应通道音频数据的 AENC 通道和 AO 通道，否则可能导致该接口调用失败。

【举例】

无。

AW_MPI_AI_SetVqeAttr

【目的】

设置 AI 的声音质量增强功能相关属性。

【语法】

ERRORTYPE AW_MPI_AI_SetVqeAttr(AUDIO_DEV AiDevId, AI_CHN AiChn,AI_VQE_CONFIG_S *pstVqeConfig);

【参数】

参数	描述	
AudioDevId	音频设备号。	输入
AiChn	AI 通道号	输入
pstVqeConfig	音频输入声音质量增强配置结构体指针	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

无。

【举例】

无。

AW_MPI_AI_GetVqeAttr

【目的】

获取 AI 的声音质量增强功能相关属性。

【语法】

ERRORTYPE AW_MPI_AI_GetVqeAttr(AUDIO_DEV AiDevId, AI_CHN AiChn, AI_VQE_CONFIG_S *pstVqeConfig);

【参数】

参数	描述	
AudioDevId	音频设备号。	输入
AiChn	AI 通道号	输入
pstVqeConfig	音频输入声音质量增强配置结构体指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 获取声音质量增强功能相关属性前必须先设置相对应 AI 通道的声音质量增强功能相关属性。

【举例】

无。

AW_MPI_AI_EnableVqe

【目的】

启用 AI 的声音质量增强功能。

【语法】

ERRORTYPE AW_MPI_AI_EnableVqe(AUDIO_DEV AiDevId, AI_CHN AiChn);

【参数】

参数	描述	
AudioDevId	音频设备号。	输入
AiChn	AI 通道号	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 启用声音质量增强功能前必须先启用相对应的 AI 通道。
- 多次使能相同 AI 通道的声音质量增强功能时，返回成功。
- 禁用 AI 通道后，如果重新启用 AI 通道，并使用声音质量增强功能，需调用此接口重新启用声音质量增强功能。

【举例】

无。

AW_MPI_AI_DisableVqe

【目的】

禁用 AI 的声音质量增强功能。

【语法】

ERRORTYPE AW_MPI_AI_DisableVqe(AUDIO_DEV AiDevId, AI_CHN AiChn);

【参数】

参数	描述	
AudioDevId	音频设备号。	输入
AiChn	AI 通道号	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 不再使用 AI 声音质量增强功能时，应该调用此接口将其禁用。
- 多次禁用相同 AI 通道的声音质量增强功能，返回成功。

【举例】

无。

AW_MPI_AI_SetTrackMode

【目的】

设置 AI 声道模式。

【语法】

ERRORTYPE AW_MPI_AI_SetTrackMode(AUDIO_DEV AudioDevId, AUDIO_TRACK_MODE_E enTrackMode);

【参数】

参数	描述	
AudioDevId	音频设备号。	输入
enTrackMode	AI 声道模式。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 在 AI 设备成功启用后再调用此接口。

- 该接口功能目前不支持。
- AI 设备工作在 I2S 模式时，支持设置声道模式，PCM 模式下不支持。

【举例】

无。

AW_MPI_AI_GetTrackMode

【目的】

获取 AI 声道模式。

【语法】

ERRORTYPE AW_MPI_AI_GetTrackMode(AUDIO_DEV AudioDevId, AUDIO_TRACK_MODE_E *penTrackMode);

【参数】

参数	描述	
AudioDevId	音频设备号。	输入
penTrackMode	AI 声道模式。	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 在 AI 设备成功启用后再调用此接口。
- AI 设备工作在 I2S 模式时，支持设置声道模式，PCM 模式下不支持。

【举例】

无。

AW_MPI_AI_ClrPubAttr

【目的】

清空 pub 属性。

【语法】

ERRORTYPE AW_MPI_AI_ClrPubAttr(AUDIO_DEV AudioDevId);

【参数】

参数	描述	
AudioDevId	音频设备号。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 清除设备属性前，需要先停止设备。

【举例】

无。

AW_MPI_AI_SaveFile

【目的】

开启音频输入保存文件功能。

【语法】

ERRORTYPE AW_MPI_AI_SaveFile(AUDIO_DEV AudioDevId, AI_CHN AiChn, AUDIO_SAVE_FILE_INFO_S *pstSaveFileInfo);

【参数】

参数	描述	
AudioDevId	音频设备号。	输入
AiChn	AI 通道号	输入
pstSaveFileInfo	音频保存文件属性结构体指针。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

无。

【举例】

无。

AW_MPI_AI_QueryFileStatus

【目的】

查询 AI 通道当前 pcm 文件保存状态。

【语法】

ERRORTYPE AW_MPI_AI_QueryFile(AUDIO_DEV AudioDevId, AI_CHN AiChn, AUDIO_SAVE_FILE_INFO_S *pstSaveFileInfo);

【参数】

参数	描述	
AudioDevId	音频设备号。	输入
AiChn	AI 通道号	输入
pstSaveFileInfo	音频保存文件属性结构体指针。	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

无。

【举例】

无。

AW_MPI_AI_SetVqeVolume

【目的】

设置 AI 设备音量大小。

【语法】

ERRORTYPE AW_MPI_AI_SetVqeVolume(AUDIO_DEV AudioDevId, AI_CHN AiChn, int s32VolumeDb);

【参数】

参数	描述	
AudioDevId	音频设备号。	输入
AiChn	AI 通道号	输入
s32VolumeDb	音频设备音量大小。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

无。

【举例】

无。

AW_MPI_AI_GetVqeVolume

【目的】

获取 AI 设备音量大小。

【语法】

ERRORTYPE AW_MPI_AI_GetVqeVolume(AUDIO_DEV AudioDevId, AI_CHN AiChn, int *ps32VolumeDb);

【参数】

参数	描述	
AudioDevId	音频设备号。	输入
AiChn	AI 通道号	输入
ps32VolumeDb	音频设备音量大小指针。	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

无。

【举例】

无。

AW_MPI_AI_RegisterCallback

【目的】

设备回调函数给 AI 通道。

【语法】

ERRORTYPE AW_MPI_AI_RegisterCallback(AUDIO_DEV AudioDevId, AI_CHN AiChn, MPPCallbackInfo *pCallback);

【参数】

参数	描述	
AudioDevId	音频设备号。	输入
AiChn	AI 通道号	输入
pCallback	来自 app 层的回调信息。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 在音视频录制过程中，音频数据从 AI 组件成功送到 AEnc 组件后，通过该回调信息，将音频时间长度送往 recorder 主控模块（组件向框架传递消息），用于统计文件中的音频 duration，以方便进行音视频同步处理。

【举例】

无。

AW_MPI_AI_SetVolume

【目的】

设置 AI 设备声音采集音量大小。

【语法】

ERRORTYPE AW_MPI_AI_SetVolume(AUDIO_DEV AudioDevId, int s32VolumeDb);

【参数】

参数	描述	
AudioDevId	音频设备号。	输入
s32VolumeDb	待设置的音量值。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

无。

【举例】

无。

AW_MPI_AI_GetVolume

【目的】

获取 AI 设备声音采集音量值。

【语法】

ERRORTYPE AW_MPI_AI_GetVolume(AUDIO_DEV AudioDevId, int *ps32VolumeDb);

【参数】

参数	描述	
AudioDevId	音频设备号。	输入
ps32VolumeDb	待设置的音量值指针。	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

无。

【举例】

无。

AW_MPI_AI_SetMute

【目的】

设置 AI 设备静音状态。

【语法】

ERRORTYPE AW_MPI_AI_SetMute(AUDIO_DEV AudioDevId, int bEnableFlag);

【参数】

参数	描述	
AudioDevId	音频设备号。	输入
bEnableFlag	待设置的静音标志值。	输入

【返回值】

返回值	描述
0	成功

非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 参数 bEnableFlag 为欲设置的静音状态值。当该值设置为 1 时则表示设置 AI 设备为静音状态，为 0 时为取消静音状态。

【举例】

无。

AW_MPI_AI_GetMute

【目的】

获取 AI 设备静音状态值。

【语法】

ERRORTYPE AW_MPI_AI_GetMute(AUDIO_DEV AudioDevId, int *pbEnableFlag);

【参数】

参数	描述	
AudioDevId	音频设备号。	输入
pbEnableFlag	待设置的静音状态值指针。	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

无。

【举例】

无。

9.4.2. 音频输出

AW_MPI_AO_SetPubAttr

【目的】

设置 AO 设备属性。

【语法】

ERRORTYPE AW_MPI_AO_SetPubAttr(AUDIO_DEV AudioDevId, const AIO_ATTR_S *pstAttr);

【参数】

参数	描述	
AudioDevId	AO 设备号。	输入
pstAttr	音频输出设备属性指针。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 在设置属性之前需要保证 AO 处于禁用状态，如果处于启用状态则需要首先禁用 AO 设备。
- AO 必须和 DA 配合起来才能正常工作，用户必须清楚 DA 发送的数据分布和通道的关系才能从正确的通道发送数据。
- 对接外置 Codec 时，由于时序的问题，在 AO 设备从模式下，建议用户先配置好对接的 Codec，再配置 AO 设备；而在 AO 设备主模式下，建议用户先配置好 AO 设备，再配置对接的 Codec。对接内置 Codec 时，都需要先配置内置 Codec，再配置 AO 设备。
- 对接内置 Codec 时，AI 设备 0 和 AO 设备 0 的帧同步时钟与位流时钟不能共用，u32ClkSel 需要配置为 0。
- AO 设备主模式时，决定 AO 设备输出时钟的关键配置项是采样率、采样精度以及通道数目，采样精度乘以通道数目即为 AO 设备时序一次采样的位宽。
- 扩展标志对 AO 设备无效。
- AO 设备属性结构体中其他项请参见 AI 模块中相关接口的描述。

【举例】

无。

AW_MPI_AO_GetPubAttr

【目的】

获取 AO 设备属性。

【语法】

ERRORTYPE AW_MPI_AO_GetPubAttr(AUDIO_DEV AudioDevId, AIO_ATTR_S *pstAttr);

【参数】

参数	描述	
AudioDevId	AO 设备号。	输入
pstAttr	音频输出设备属性指针。	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 获取的属性为前一次配置的属性。
- 如果从未配置过属性，则返回属性未配置的错误。

【举例】

无。

AW_MPI_AO_ClrPubAttr

【目的】

清除 AO 设备属性。

【语法】

ERRORTYPE AW_MPI_AO_ClrPubAttr(AUDIO_DEV AudioDevId);

【参数】

参数	描述	
AudioDevId	AO 设备号。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 清除设备属性前，需要先停止设备。

【举例】

无。

AW_MPI_AO_Enable

【目的】

启用 AO 设备。

【语法】

ERRORTYPE AW_MPI_AO_Enable(AUDIO_DEV AudioDevId);

【参数】

参数	描述	
AudioDevId	AO 设备号。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 要求在启用前配置 AO 设备属性，否则会返回属性未配置的错误。
- 如果 AO 设备已经启用，则直接返回成功。

【举例】

无。

AW_MPI_AO_Disable

【目的】

禁用 AO 设备。

【语法】

ERRORTYPE AW_MPI_AO_Disable(AUDIO_DEV AudioDevId);

【参数】

参数	描述	
AudioDevId	AO 设备号。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 如果 AO 设备已经禁用，则直接返回成功。
- 禁用 AO 设备前必须先禁用设备下所有 AO 通道。

【举例】

无。

AW_MPI_AO_EnableChn

【目的】

创建 AO 通道。

【语法】

ERRORTYPE AW_MPI_AO_EnableChn(AUDIO_DEV AudioDevId, AO_CHN AoChn);

【参数】

参数	描述	
AudioDevId	AO 设备号。	输入
AoChn	AO 通道号。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 启用 AO 通道前，必须先启用其所属的 AO 设备，否则返回设备未启动的错误码。

【举例】

无。

AW_MPI_AO_DisableChn

【目的】

销毁 AO 通道。

【语法】

ERRORTYPE AW_MPI_AO_DisableChn(AUDIO_DEV AudioDevId, AO_CHN AoChn);

【参数】

参数	描述	
AudioDevId	AO 设备号。	输入
AoChn	AO 通道号。	输入

【返回值】

返回值	描述

0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 销毁 AO 通道时，该通道会从其所属的 AO 设备的播放管理列表中退出，当只有该 AO 通道占用该 AO 设备时，会主动关闭 AO 设备；如果还有其 AO 通道占用 AO 设备，则等到最后一个 AO 设备退出时，才会主动关闭 AO 设备。
- 释放 AO 通道和 AO 设备流程：AO_StopChn->AO_DisableChn->AO_Disable

【举例】

无。

AW_MPI_AO_StartChn

【目的】

启用 AO 通道。

【语法】

ERRORTYPE AW_MPI_AO_StartChn(AUDIO_DEV AudioDevId, AO_CHN AoChn);

【参数】

参数	描述	
AudioDevId	AO 设备号。	输入
AoChn	AO 通道号。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 启用 AO 通道前，必须先启用其所属的 AO 设备，否则返回设备未启动的错误码。

【举例】

无。

AW_MPI_AO_StopChn

【目的】

停止 AO 通道。

【语法】

ERRORTYPE AW_MPI_AO_StopChn(AUDIO_DEV AudioDevId, AO_CHN AoChn);

【参数】

参数	描述	
AudioDevId	AO 设备号。	输入
AoChn	AO 通道号。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

【举例】

无。

AW_MPI_AO_RegisterCallback

【目的】

停止 AO 通道。

【语法】

ERRORTYPE AW_MPI_AO_RegisterCallback(AUDIO_DEV AudioDevId, AO_CHN AoChn, MPPCallbackInfo *pCallback);

【参数】

参数	描述	
AudioDevId	AO 设备号。	输入

AoChn	AO 通道号。	输入
pCallback	来自 app 层的回调信息。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

【举例】

无。

AW_MPI_AO_SendFrame

【目的】

发送 AO 音频帧。

【语法】

ERRORTYPE AW_MPI_AO_SendFrame(AUDIO_DEV AudioDevId, AO_CHN AoChn, const AUDIO_FRAME_S *pstData, int s32MilliSec);

【参数】

参数	描述	
AudioDevId	AO 设备号。	输入
AoChn	AO 通道号。	输入
pstData	音频帧结构体指针。	输入
s32MilliSec	发送数据的超时时间。 -1 表示阻塞模式； 0 表示非阻塞模式； >0 表示阻塞 s32MilliSec 毫秒，超时则报错返回。	输入

【返回值】

返回值	描述

0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 该接口用于 app 主动发送音频帧至 AO 输出，如果 AO 通道已经通过系统绑定（AW_MPI_SYS_Bind）接口与 AI 或 ADEC 绑定，不需要也不建议调此接口。
- s32MilliSec 的值必须大于等于-1，等于-1 时采用阻塞模式发送数据，等于 0 时采用非阻塞模式发送数据，大于 0 时，阻塞 s32MilliSec 毫秒后，则返回超时并报错。
- 调用该接口发送音频帧到 AO 输出时，必须先使能对应的 AO 通道。

【举例】

无。

AW_MPI_AO_EnableReSmp

【目的】

启用 AO 重采样。

【语法】

ERRORTYPE AW_MPI_AO_EnableReSmp(AUDIO_DEV AudioDevId, AO_CHN AoChn, AUDIO_SAMPLE_RATE_E enInSampleRate);

【参数】

参数	描述	
AudioDevId	AO 设备号。	输入
AoChn	AO 通道号。	输入
enInSampleRate	音频重采样的输入采样率。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 应该在启用 AO 通道之后，绑定 AO 通道之前，调用此接口启用重采样功能。
- 允许重复启用重采样功能，但必须保证后配置的重采样输入采样率与之前配置的重采样输入采样率一样。
- 在禁用 AO 通道后，如果重新启用 AO 通道，并使用重采样功能，需调用此接口重新启用重采样。
- AO 重采样的输入采样率必须与 AO 设备属性配置的采样率不相同。

【举例】

无。

AW_MPI_AO_DisableReSmp

【目的】

禁用 AO 重采样。

【语法】

ERRORTYPE AW_MPI_AO_DisableReSmp(AUDIO_DEV AudioDevId, AO_CHN AoChn);

【参数】

参数	描述	
AudioDevId	AO 设备号。	输入
AoChn	AO 通道号。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 不再使用 AO 重采样功能的话，应该调用此接口将其禁用。

【举例】

无。

AW_MPI_AO_PauseChn

【目的】

暂停 AO 通道。

【语法】

ERRORTYPE AW_MPI_AO_PauseChn(AUDIO_DEV AudioDevId, AO_CHN AoChn);

【参数】

参数	描述	
AudioDevId	AO 设备号。	输入
AoChn	AO 通道号。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- AO 通道暂停后，如果绑定的 ADEC 通道继续向此通道发送音频帧数据，发送的音频帧数据将会被阻塞；而如果绑定的 AI 通道继续向此通道发送音频帧数据，在通道缓冲未满的情况下则将音频帧放入缓冲区，在满的情况下则将音频帧丢弃。
- AO 通道为禁用状态时，不允许调用此接口暂停 AO 通道。

【举例】

无。

AW_MPI_AO_ResumeChn

【目的】

恢复 AO 通道。

【语法】

ERRORTYPE AW_MPI_AO_ResumeChn(AUDIO_DEV AudioDevId, AO_CHN AoChn);

【参数】

参数	描述	
AudioDevId	AO 设备号。	输入
AoChn	AO 通道号。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- AO 通道暂停后可以通过调用此接口重新恢复。
- AO 通道为暂停状态或使能状态下，调用此接口返回成功；否则调用将返回错误。

【举例】

无。

AW_MPI_AO_Seek

【目的】

Player 跳播播放。

【语法】

ERRORTYPE AW_MPI_AO_Seek(AUDIO_DEV AudioDevId, AO_CHN AoChn);

【参数】

参数	描述	
AudioDevId	AO 设备号。	输入
AoChn	AO 通道号。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 用于播放视频文件时快进或快退到某一时刻。

【举例】

无。

AW_MPI_AO_ClearChnBuf

【目的】

清除 AO 通道中当前的音频数据缓存。

【语法】

ERRORTYPE AW_MPI_AO_ClearChnBuf(AUDIO_DEV AudioDevId, AO_CHN AoChn);

【参数】

参数	描述	
AudioDevId	AO 设备号。	输入
AoChn	AO 通道号。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 在 AO 通道成功启用后再调用此接口。
- 为完全清除解码回放通路上所有缓存数据，此接口还应该与 AW_MPI_ADEC_ClearChnBuf 接口配合使用。

【举例】

无。

AW_MPI_AO_QueryChnStat

【目的】

查询 AO 通道中当前的音频数据缓存状态。

【语法】

ERRORTYPE AW_MPI_AO_QueryChnStat(AUDIO_DEV AudioDevId ,AO_CHN AoChn, AO_CHN_STATE_S *pstStatus);

【参数】

参数	描述	
AudioDevId	AO 设备号。	输入
AoChn	AO 通道号。	输入
pstStatus	缓存状态结构体指针。	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 在 AO 通道成功启用后再调用此接口。

【举例】

无。

AW_MPI_AO_SetTrackMode

【目的】

设置 AO 设备声道模式。

【语法】

ERRORTYPE AW_MPI_AO_SetTrackMode(AUDIO_DEV AudioDevId, AUDIO_TRACK_MODE_E enTrackMode);

【参数】

参数	描述	
AudioDevId	AO 设备号。	输入

enTrackMode	音频设备声道模式。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 在 AO 设备成功启用后再调用此接口。
- AO 设备工作在 I2S 模式时，支持设置声道模式，PCM 模式下不支持。

【举例】

无。

AW_MPI_AO_GetTrackMode

【目的】

获取 AO 设备声道模式。

【语法】

ERRORTYPE AW_MPI_AO_GetTrackMode(AUDIO_DEV AudioDevId, AUDIO_TRACK_MODE_E *penTrackMode);

【参数】

参数	描述	
AudioDevId	AO 设备号。	输入
penTrackMode	音频设备声道模式指针。	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 在 AO 设备成功启用后再调用此接口。
- AO 设备工作在 I2S 模式时，支持获取声道模式，PCM 模式下不支持。

【举例】

无。

AW_MPI_AO_SetVolume

【目的】

设置 AO 设备音量大小。

【语法】

ERRORTYPE AW_MPI_AO_SetVolume(AUDIO_DEV AudioDevId, int s32VolumeDb);

【参数】

参数	描述	
AudioDevId	AO 设备号。	输入
s32VolumeDb	AO 设备音量大小。	

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 在 AO 设备成功启用后再调用此接口。
- 音量大小 s32VolumeDb 参数的范围为 0~100 内的整数。

【举例】

无。

AW_MPI_AO_GetVolume

【目的】

获取 AO 设备音量大小。

【语法】

ERRORTYPE AW_MPI_AO_GetVolume(AUDIO_DEV AudioDevId, int *ps32VolumeDb);

【参数】

参数	描述	
AudioDevId	AO 设备号。	输入
ps32VolumeDb	AO 设备音量大小指针。	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 在 AO 设备成功启用后再调用此接口。

【举例】

无。

AW_MPI_AO_SetMute

【目的】

设置 AO 设备静音状态。

【语法】

ERRORTYPE AW_MPI_AO_SetMute(AUDIO_DEV AudioDevId, BOOL bEnable, AUDIO_FADE_S *pstFade);

【参数】

参数	描述	
AudioDevId	AO 设备号。	输入
bEnable	音频设备是否启用静音。	输入
pstFade	淡入淡出结构体指针。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 在 AO 设备成功启用后再调用此接口。
- 调用此接口时，用户可以选择是否使用淡入淡出功能，如果不使用淡入淡出则将结构体指针赋为空即可。（暂不支持 pstFade 参数设置）
- 静音标志 bEnable 为 1 时，设置音频主通道静音。bEnable 为 0 时，取消主通道静音，即恢复声音，音量大小为设置静音前的值。

【举例】

无。

AW_MPI_AO_GetMute

【目的】

获取 AO 设备静音状态。

【语法】

ERRORTYPE AW_MPI_AO_GetMute(AUDIO_DEV AudioDevId, BOOL *pbEnable, AUDIO_FADE_S *pstFade);

【参数】

参数	描述	
AudioDevId	AO 设备号。	输入
pbEnable	音频设备静音状态指针。	输出
pstFade	淡入淡出结构体指针。	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 在 AO 设备成功启用后再调用此接口。
- 静音状态值为 1 时指示 AO 设备目前处于静音状态；反之为正常工作状态。

【举例】

无。

AW_MPI_AO_SetVqeAttr

【目的】

设置 AO 的声音质量增强功能相关属性。

【语法】

ERRORTYPE AW_MPI_AO_SetVqeAttr(AUDIO_DEV AudioDevId, AO_CHN AoChn, AO_VQE_CONFIG_S *pstVqeConfig);

【参数】

参数	描述	
AudioDevId	AO 设备号。	输入
AoChn	AO 通道号。	输入
pstVqeConfig	音频输出声音质量增强配置结构体指针。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 启用声音质量增强功能前必须先设置相对应 AO 通道的声音质量增强功能相关属性。
- 设置 AO 的声音质量增强功能相关属性前，必须先使能对应的 AO 通道。
- 相同 AO 通道的声音质量增强功能不支持动态设置属性，重新设置 AO 通道的声音质量增强功能相关属性时，需要先关闭 AO 通道的声音质量功能，再设置 AO 通道的声音质量增强功能相关属性。

- AO 声音质量增强功能包括了环境噪声抑制功能，自动增益控制功能，高通滤波功能，均衡器功能。在设置声音质量增强功能属性时，可通过配置相应的声音质量增强功能属性来选择使能其中的部分功能。

【举例】

无。

AW_MPI_AO_GetVqeAttr

【目的】

获取 AO 的声音质量增强功能相关属性。

【语法】

ERRORTYPE AW_MPI_AO_GetVqeAttr(AUDIO_DEV AudioDevId, AO_CHN AoChn, AO_VQE_CONFIG_S *pstVqeConfig);

【参数】

参数	描述	
AudioDevId	AO 设备号。	输入
AoChn	AO 通道号。	输入
pstVqeConfig	音频输出声音质量增强配置结构体指针	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 获取声音质量增强功能相关属性前必须先设置相对应 AO 通道的声音质量增强功能相关属性。

【举例】

无。

AW_MPI_AO_EnableVqe

【目的】

使能 AO 的声音质量增强功能。

【语法】

ERRORTYPE AW_MPI_AO_EnableVqe(AUDIO_DEV AudioDevId, AO_CHN AoChn);

【参数】

参数	描述	
AudioDevId	AO 设备号。	输入
AoChn	AO 通道号。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 启用声音质量增强功能前必须先启用相对应的 AO 通道。
- 多次使能相同 AO 通道的声音质量增强功能时，返回成功。
- 禁用 AO 通道后，如果重新启用 AO 通道，并使用声音质量增强功能，需调用此接口重新启用声音质量增强功能。

【举例】

无。

AW_MPI_AO_DisableVqe

【目的】

禁用 AO 的声音质量增强功能。

【语法】

ERRORTYPE AW_MPI_AO_DisableVqe(AUDIO_DEV AudioDevId, AO_CHN AoChn);

【参数】

参数	描述	
AudioDevId	AO 设备号。	输入
AoChn	AO 通道号。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 不再使用 AO 声音质量增强功能时，应该调用此接口将其禁用。
- 多次禁用相同 AO 通道的声音质量增强功能，返回成功。

【举例】

无。

AW_MPI_AO_SetStreamEof

【目的】

通知 AO 组件码流传递完毕标志。

【语法】

```
ERRORTYPE AW_MPI_AO_SetStreamEof(AUDIO_DEV AudioDevId, AO_CHN AoChn, BOOL bEofFlag);
```

【参数】

参数	描述	
AudioDevId	AO 设备号。	输入
AoChn	AO 通道号。	输入
bEofFlag	码流结束标志。	

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

- 播放完毕后，app 必须设置码流结束标志到 AO 组件中。

【举例】

无。

AW_MPI_AO_SaveFile

【目的】

设置 AO 组件 pcm 数据的文件保存信息。

【语法】

```
ERRORTYPE AW_MPI_AO_SaveFile(AUDIO_DEV AudioDevId, AO_CHN AoChn,
AUDIO_SAVE_FILE_INFO_S *pstSaveFileInfo);
```

【参数】

参数	描述	
AudioDevId	AO 设备号。	输入
AoChn	AO 通道号。	输入
pstSaveFileInfo	文件保存信息指针。	

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

【举例】

无。

AW_MPI_AO_QueryFileStatus

【目的】

查询 AO 组件文件保存状态。

【语法】

ERRORTYPE AW_MPI_AO_QueryFileStatus(AUDIO_DEV AudioDevId, AO_CHN AoChn, AUDIO_SAVE_FILE_INFO_S *pstSaveFileInfo);

【参数】

参数	描述	
AudioDevId	AO 设备号。	输入
AoChn	AO 通道号。	输入
pstSaveFileInfo	文件保存信息指针。	

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aio.h、mm_common.h

【注意】

【举例】

无。

9.4.3. 音频编码

AW_MPI_AENC_CreateChn

【目的】

创建音频编码通道。

【语法】

ERRORTYPE AW_MPI_AENC_CreateChn(AENC_CHN AeChn, const AENC_CHN_ATTR_S *pstAttr);

【参数】

参数	描述	
AeChn	音频编码通道号。	输入
pstAttr	音频编码属性指针。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aenc.h、mm_common.h

【注意】

无。

【举例】

无。

AW_MPI_AENC_DestroyChn

【目的】

销毁音频编码通道。

【语法】

ERRORTYPE AW_MPI_AENC_DestroyChn(AENC_CHN AeChn);

【参数】

参数	描述	
AeChn	音频编码通道号。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aenc.h、mm_common.h

【注意】

- 通道未创建的情况下调用此接口会返回成功。
- 如果正在获取/释放码流或者发送帧时销毁该通道，则会返回失败，用户同步处理时需要注意。

【举例】

无。

AW_MPI_AENC_SendFrame

【目的】

发送音频编码音频帧。

【语法】

ERRORTYPE AW_MPI_AENC_SendFrame(AENC_CHN AeChn,const AUDIO_FRAME_INFO_S *pFrameInfo);

【参数】

参数	描述	
AeChn	音频编码通道号。	输入
pFrameInfo	音频 pcm 帧结构体指针。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aenc.h、mm_common.h

【注意】

- 发送 pcm 信息接口是非阻塞接口，如果音频 pcm 缓存区满，则直接返回失败。
- 该接口用于用户主动发送音频帧进行编码，如果 AENC 通道已经通过系统绑定（AW_MPI_SYS_Bind）接口与 AI 绑定，不需要也不建议调此接口。
- 调用该接口发送音频编码音频帧时，必须先创建对应的编码通道。

【举例】

无。

AW_MPI_AENC_GetStream

【目的】

获取编码后码流。

【语法】

ERRORTYPE AW_MPI_AENC_GetStream(AENC_CHN AeChn, AUDIO_STREAM_S *pStream, int nMilliSec);

【参数】

参数	描述	
AeChn	音频编码通道号。	输入
pStream	音频编码属性指针。	输出
nMilliSec	获取数据的超时时间: -1 表示阻塞模式，无数据时一直等待； 0 表示非阻塞模式，无数据时则报错返回； >0 表示阻塞 nMilliSec 毫秒，超时则报错返回。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aenc.h、mm_common.h

【注意】

- 必须创建通道后才可能获取码流，否则直接返回失败，如果在获取码流过程中销毁通道则会立刻返回失败。
- nMilliSec 的值必须大于等于-1，等于-1 时采用阻塞模式获取数据，等于 0 时采用非阻塞模式获取数据，大于 0 时，阻塞 nMilliSec 毫秒后，没有数据则返回超时并报错。

【举例】

无。

AW_MPI_AENC_ReleaseStream

【目的】

释放用户占用的编码码流。

【语法】

ERRORTYPE AW_MPI_AENC_ReleaseStream(AENC_CHN AeChn, AUDIO_STREAM_S *pStream);

【参数】

参数	描述	
AeChn	音频编码通道号。	输入
pStream	音频编码属性指针。	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aenc.h、mm_common.h

【注意】

- 用户通过 AW_MPI_AENC_GetStream 接口获得编码数据后，必须尽快将该编码数据再还给编码组件，以方便释放 stream buffer 中占用的空间。
- 该接口配合 AW_MPI_AENC_GetStream 一起使用，用于 none-tunnel 方式来保存编码后的数据，通常用于 nvr 模式，将采集到的 pcm 数据送编码，然后应用去拿编码数据，再去做文件封装。
- AI 组件与 AEnc 组件通常用 tunnel 方式进行数据传递（需进行 bind 的操作）。AEnc 组件与 Muexer 组件通过 tunnel 方式数据传递时，mpp 组件内部做封装处理，保存为本地文件；通过 none-tunnel 方式数据传递时，即应用主动拿编码数据，然后自行处理，或通过网络传走，或自行封装写卡。
- 码流最好能够在使用完之后立即释放，如果不及时释放，会导致编码过程阻塞。
- 释放的码流必须是从该通道获取的码流，不得对码流信息结构体进行任何修改，否则会导致码流不能释放，使此码流 buffer 丢失，甚至导致程序异常。
- 释放码流时必须保证通道已经被创建，否则直接返回失败，如果在释放码流过程中销毁通道则会立刻返回失败。

【举例】

无。

AW_MPI_AENC_StartRecvPcm

【目的】

启动音频编码组件。

【语法】

ERRORTYPE AW_MPI_AENC_StartRecvPcm(AENC_CHN AeChn);

【参数】

参数	描述	
AeChn	音频编码通道号。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aenc.h、mm_common.h

【注意】

- 该接口常用于绑定方式（AI、AENC 组件绑定）下，控制音频编码器的启动。如果 APP 主动发送音频帧进行编码，建议使用 AW_MPI_AENC_SendFrame 接口，如果 APP 本身不进行音频数据管理，而希望系统主动处理，建议使用本接口。

【举例】

无。

AW_MPI_AENC_StopRecvPcm

【目的】

关闭音频编码组件。

【语法】

ERRORTYPE AW_MPI_AENC_StopRecvPcm(AENC_CHN AeChn);

【参数】

参数	描述	
AeChn	音频编码通道号。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aenc.h、mm_common.h

【注意】

无。

【举例】

无。

AW_MPI_AENC_ResetChn

【目的】

复位 AEnc 通道。

【语法】

ERRORTYPE AW_MPI_AENC_ResetChn(AENC_CHN AeChn);

【参数】

参数	描述	
AeChn	音频编码通道号。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aenc.h、mm_common.h

【注意】

【举例】

无。

AW_MPI_AENC_Query

【目的】

查询 AEnc 通道内部数据状态。

【语法】

ERRORTYPE AW_MPI_AENC_Query(AENC_CHN AeChn, AENC_CHN_STAT_S *pStat);

【参数】

参数	描述	

AeChn	音频编码通道号。	输入
pStat	来自 app 的状态信息指针。	

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aenc.h、mm_common.h

【注意】

【举例】

无。

AW_MPI_AENC_RegisterCallback

【目的】

向 AEnc 通道注册回调信息。

【语法】

ERRORTYPE AW_MPI_AENC_RegisterCallback(AENC_CHN AeChn, MPPCallbackInfo *pCallback);

【参数】

参数	描述	
AeChn	音频编码通道号。	输入
pCallback	来自 app 的回调信息指针。	

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aenc.h、mm_common.h

【注意】

【举例】

无。

AW_MPI_AENC_SetChnAttr

【目的】

设置 AEnc 通道属性信息。

【语法】

ERRORTYPE AW_MPI_AENC_SetChnAttr(AENC_CHN AeChn, const AENC_CHN_ATTR_S *pAttr);

【参数】

参数	描述	
AeChn	音频编码通道号。	输入
pAttr	来自 app 的通道属性信息指针。	

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aenc.h、mm_common.h

【注意】

【举例】

无。

AW_MPI_AENC_GetChnAttr

【目的】

获取 AEnc 通道属性信息。

【语法】

ERRORTYPE AW_MPI_AENC_GetChnAttr(AENC_CHN AeChn, const AENC_CHN_ATTR_S *pAttr);

【参数】

参数	描述	

AeChn	音频编码通道号。	输入
pAttr	通道属性信息指针。	

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_aenc.h、mm_common.h

【注意】

无。

【举例】

无。

AW_MPI_AENC_GetHandle

【目的】

获取 AEnc 通道句柄。

【语法】

int AW_MPI_AENC_GetHandle(AENC_CHN AeChn);

【参数】

参数	描述	
AeChn	音频编码通道号。	输入

【返回值】

返回值	描述
int	编码句柄号。

【需求】

- 头文件：mm_comm_aenc.h、mm_common.h

【注意】

无。

【举例】

无。

9.4.4. 音频解码

AW_MPI_ADEC_CreateChn

【目的】

创建音频解码通道。

【语法】

ERRORTYPE AW_MPI_ADEC_CreateChn(ADEC_CHN AdChn, ADEC_CHN_ATTR_S *pstAttr);

【参数】

参数	描述	
AdChn	音频解码通道号。	输入
pstStream	音频解码通道属性指针。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_adec.h、mm_common.h

【注意】

无。

【举例】

无。

AW_MPI_ADEC_DestroyChn

【目的】

销毁音频解码通道。

【语法】

ERRORTYPE AW_MPI_ADEC_DestroyChn(ADEC_CHN AdChn);

【参数】

参数	描述	

AdChn	音频解码通道号。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_adec.h、mm_common.h

【注意】

无。

【举例】

无。

AW_MPI_ADEC_ResetChn

【目的】

复位 ADec 通道。

【语法】

ERRORTYPE AW_MPI_ADEC_ResetChn(ADEC_CHN AdChn);

【参数】

参数	描述	
AdChn	音频解码通道号。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_adec.h、mm_common.h

【注意】

无。

【举例】

无。

AW_MPI_ADEC_RegisterCallback

【目的】

向 ADec 通道注册回调信息。

【语法】

ERRORTYPE AW_MPI_ADEC_RegisterCallback(ADEC_CHN ADecChn, MPPCallbackInfo *pCallback);

【参数】

参数	描述	
AdChn	音频解码通道号。	输入
pCallback	来自 app 层的回调信息指针。	

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_adec.h、mm_common.h

【注意】

无。

【举例】

无。

AW_MPI_ADEC_SendStream

【目的】

向音频解码通道发送码流。

【语法】

ERRORTYPE AW_MPI_ADEC_SendStream(ADEC_CHN AdChn, const AUDIO_STREAM_S *pstStream, BOOL bBlock);

【参数】

参数	描述	
AdChn	音频解码通道号。	输入
pstStream	音频码流指针。	输入
bBlock	阻塞标识。TRUE：阻塞。FALSE：非阻塞。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_adec.h、mm_common.h

【注意】

无。

【举例】

无。

AW_MPI_ADEC_ClearChnBuf

【目的】

清除 ADEC 通道中当前的音频数据缓存。

【语法】

ERRORTYPE AW_MPI_ADEC_ClearChnBuf(ADEC_CHN AdChn);

【参数】

参数	描述	
AdChn	音频解码通道号。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_adec.h、mm_common.h

【注意】

无。

【举例】

无。

AW_MPI_ADEC_GetFrame

【目的】

获取解码后音频帧。

【语法】

ERRORTYPE AW_MPI_ADEC_GetFrame(ADEC_CHN AdChn, AUDIO_FRAME_INFO_S *pstFrmInfo, BOOL bBlock);

【参数】

参数	描述	
AdChn	音频解码通道号。	输入
pstFrmInfo	音频帧指针。	输出
bBlock	阻塞标识。TRUE：阻塞。FALSE：非阻塞。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_adec.h、mm_common.h

【注意】

无。

【举例】

无。

AW_MPI_ADEC_ReleaseFrame

【目的】

释放从音频解码通道获取的音频帧。

【语法】

ERRORTYPE AW_MPI_ADEC_ReleaseFrame(ADEC_CHN AdChn, AUDIO_FRAME_INFO_S *pstFrmInfo);

【参数】

参数	描述	
AdChn	音频解码通道号。	输入
pstFrmInfo	获取的音频帧指针。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_adec.h、mm_common.h

【注意】

无。

【举例】

无。

AW_MPI_ADEC_SetStreamEof

【目的】

向解码器发送码流结束标识符，并清除码流 buffer。

【语法】

ERRORTYPE AW_MPI_ADEC_SetStreamEof(ADEC_CHN AdChn, BOOL bEofFlag);

【参数】

参数	描述	
AdChn	音频解码通道号。	输入
bEofFlag	是否立即清除解码器内部的缓存数据。 取值范围：	输入

	FALSE：延时清除。不会立即清除解码器内部的缓存数据，解码会继续进行，直到剩余 buffer 不足一帧数据时进行清除操作。 TRUE：立即清除解码器内部缓存数据。	

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_adec.h、mm_common.h

【注意】

无。

【举例】

无。

AW_MPI_ADEC_StartRecvStream

【目的】

启动 ADEC 通道。

【语法】

ERRORTYPE AW_MPI_ADEC_StartRecvStream(ADEC_CHN AdChn);

【参数】

参数	描述	
AdChn	音频解码通道号。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_adec.h、mm_common.h

【注意】

- 在启动该通道前，该 ADec 通道已经创建。否则返回错误码。
- 该 api 的调用，其内部实现中，该组件由 idle 状态过渡到 executing 状态。

【举例】

无。

AW_MPI_ADEC_StopRecvStream

【目的】

停止 ADEC 通道。

【语法】

ERRORTYPE AW_MPI_ADEC_StopRecvStream(ADEC_CHN AdChn);

【参数】

参数	描述	
AdChn	音频解码通道号。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_adec.h、mm_common.h

【注意】

- 在停止该通道前，需确保该 ADec 通道已经创建。否则返回错误码。
- 该 api 的调用，其内部实现中，该组件由 executing 状态过渡到 idle 状态。

【举例】

无。

AW_MPI_ADEC_SetChnAttr

【目的】

设置 ADEC 通道属性。

【语法】

ERRORTYPE AW_MPI_ADEC_SetChnAttr(ADEC_CHN ADecChn, const ADEC_CHN_ATTR_S *pAttr);

【参数】

参数	描述	
AdChn	音频解码通道号。	输入
pAttr	音频帧属性。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_adec.h、mm_common.h

【注意】

无。

【举例】

无。

AW_MPI_ADEC_GetChnAttr

【目的】

获取 ADEC 通道属性。

【语法】

ERRORTYPE AW_MPI_ADEC_GetChnAttr(ADEC_CHN ADecChn, ADEC_CHN_ATTR_S *pAttr);

【参数】

参数	描述	
AdChn	音频解码通道号。	输入
pAttr	音频帧属性。	输出

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_adec.h、mm_common.h

【注意】

无。

【举例】

无。

AW_MPI_ADEC_Pause

【目的】

修改 ADEC 通道中组件内部状态。

【语法】

ERRORTYPE AW_MPI_ADEC_Pause(ADEC_CHN AdChn);

【参数】

参数	描述	
AdChn	音频解码通道号。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_adec.h、mm_common.h

【注意】

- 组件只有在 idle 或 executing 状态下才能变换到 pause 状态。常用于 player 在播放时暂停。

【举例】

无。

AW_MPI_ADEC_Seek

【目的】

Player 跳播播放。

【语法】

ERRORTYPE AW_MPI_ADEC_Seek(ADEC_CHN AdChn);

【参数】

参数	描述	
AdChn	音频解码通道号。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，其值见错误码。

【需求】

- 头文件：mm_comm_adec.h、mm_common.h

【注意】

- 播放器进行 seek 跳转播放时，该接口在内部实现为刷新音频解码器内部的 pcm 和 bitstream 缓冲管理器。

【举例】

无。

9.5. 数据结构

9.5.1. 音频输入输出

AIO_ATTR_S

【说明】

定义音频输入输出设备属性结构。

【定义】

```
typedef struct AIO_ATTR_S
{
    AUDIO_SAMPLE_RATE_E   enSamplerate;
    AUDIO_BIT_WIDTH_E     enBitwidth;
    AIO_MODE_E            enWorkmode;
```

```
    AUDIO_SOUND_MODE_E   enSoundmode;
    unsigned int         u32EXFlag;
    unsigned int         u32FrmNum;
    unsigned int         u32PtNumPerFrm;
    unsigned int         u32ChnCnt;
    unsigned int         u32ClkSel;
    unsigned int         mPcmCardId;
} AIO_ATTR_S;
```

【成员】

成员名称	描述	其它说明
enSamplerate	音频采样率。	支持。
enBitwidth	音频采样精度。	支持。
enWorkmode	音频输入输出工作模式。	暂不支持。
enSoundmode	音频声道模式。	暂不支持。
u32EXFlag	扩展标志。	暂不支持。
u32FrmNum	缓存帧数目。	暂不支持。
u32PtNumPerFrm	每帧采样点个数。	暂不支持。
u32ChnCnt	支持的通道数目。	支持。
u32ClkSel	配置 AI 设备 0 是否复用 AO 设备 0 的帧同步时钟及位流时钟。	暂不支持。
mPcmCardId	配置声卡类型。	支持。

【注意事项】

在设置AIO设备公共属性(AW_MPI_AI/AO_SetPubAttr)时，其设备属性结构中三个field需填充正确的参数，这三个field分别为enSamplerate、enBitwidth、u32ChnCnt。

在设置AO设备公共属性时，还需再添加输出的声卡类型mPcmCardId：AudioCodec和SndHdmi，其对应的输出接口，一种为lineout方式，另一种为hdmi输出方式。

【相关数据类型及接口】

AW_MPI_AI_SetPubAttr。

AW_MPI_AO_SetPubAttr。

AI_CHN_PARAM_S

【说明】

定义通道参数结构体。

【定义】

```
typedef struct AI_CHN_PARAM_S
{
    unsigned int u32UsrFrmDepth;
} AI_CHN_PARAM_S
```

【成员】

成员名称	描述	其它说明
u32UsrFrmDepth	音频帧缓存深度。	暂不支持。

【注意事项】

无。

【相关数据类型及接口】

无。

AUDIO_FRAME_S

【说明】

定义音频帧结构体。

【定义】

```
typedef struct AUDIO_FRAME_S
{
    AUDIO_BIT_WIDTH_E     enBitwidth;
    AUDIO_SOUND_MODE_E    enSoundmode;
    void                  *mpAddr;
    unsigned long long    mTimeStamp;
```

```
    unsigned int        mSeq;
    unsigned int        mLen;
    unsigned int        mId;
} AUDIO_FRAME_S;
```

【成员】

成员名称	描述	其它说明
enBitwidth	音频采样精度。	支持。
enSoundmode	音频声道模式。	支持。
mpAddr	音频帧数据虚拟地址。	支持。
mTimeStamp	音频帧时间戳。以μs 为单位。	支持。
mSeq	音频帧序号。	暂不支持。
mLen	音频帧长度。以 byte 为单位。	支持。
mId	音频帧 ID。	支持。

【注意事项】

- mLen（音频帧长度）指1024个sample采样的数据长度，在位宽为16、单通道情况下，其值为2048，在位宽为16、双通道情况下，其值为4096。

【相关数据类型及接口】

无。

AEC_FRAME_S

【说明】

定义音频回音消除参考帧信息结构体。

【定义】

```
typedef struct AEC_FRAME_S
{
    AUDIO_FRAME_S    stRefFrame;
    BOOL             bValid;
    BOOL             bSysBind;
} AEC_FRAME_S;
```

【成员】

成员名称	描述	其它说明
stRefFrame	回音消除参考帧结构体。	支持。
bValid	参考帧有效标志。	支持。
bSysBind	组件间是否采用绑定方式。	支持。

【注意事项】

无。

【相关数据类型及接口】

无。

AUDIO_AGC_CONFIG_S

【说明】

定义音频自动电平控制配置信息结构体。

【定义】

```
typedef struct AUDIO_AGC_CONFIG_S
{
    BOOL bUsrMode;
    signed char s8TargetLevel;
    signed char s8NoiseFloor;
    signed char s8MaxGain;
    signed char s8AdjustSpeed;
    signed char s8ImproveSNR;
    signed char s8UseHighPassFilt;
    signed char s8OutputMode;
    short       s16NoiseSupSwitch;
    int s32Reserved;
} AUDIO_AGC_CONFIG_S;
```

【成员】

成员名称	描述
bUsrMode	是否采用用户模式：0 自动模式，1 用户模式，默认为 0
s8TargetLevel	目标电平。
s8NoiseFloor	噪声底线。
s8MaxGain	最大增益。
s8AdjustSpeed	调整速度。
s8ImproveSNR	提高信噪比开关。
s8UseHighPassFilt	打开高通滤波标志。
s8OutputMode	输出模式,低于NoiseFloor 的信号输出静音。范围：[0:关闭, 1:打开]
s16NoiseSupSwitch	噪声抑制开关；范围{0, 1}，0 表示关闭，1 表示开启。
s32Reserved	保留。

【注意事项】

无。

【相关数据类型及接口】

无。

AI_AEC_CONFIG_S

【说明】

定义音频回声抵消配置信息结构体。

【定义】

```
typedef struct AI_AEC_CONFIG_S
{
    EC_PARAMS_T prms;
} AI_AEC_CONFIG_S;

typedef struct
{
    int enable_aec;
    AEC_PARAMS_T aec_prms;
    int enable_bdc;
    BDC_PARAMS_T bdc_prms;
```

```
    Int enable_cdc;
    DRC_PARAMS_T txdrc_prms;
    int enable_rxdrc;
    DRC_PARAMS_T rxdrc_prms;
    int enable_txeq;
    EQ_PARAMS_T txeq_prms;
    int enable_rxeq;
    EQ_PARAMS_T rxeq_prms;
    int enable_ns;
    NS_PARAMS_T ns_prms;
    int enable_txfade;
} EC_PARAMS_T;
```

【成员】

成员名称	其它说明
prms	参考《V316 音效库模块说明》。

【注意事项】

无。

【相关数据类型及接口】

无。

AUDIO_ANR_CONFIG_S

【说明】

定义音频环境噪声抑制功能配置信息结构体。

【定义】

```
typedef struct AUDIO_ANR_CONFIG_S
{
    BOOL bUsrMode;
    short s16NrIntensity;
    short s16NoiseDbThr;
    signed char s8SpProSwitch;
    int s32Reserved;
} AUDIO_ANR_CONFIG_S;
```

【成员】

成员名称	描述
bUsrMode	是否采用用户模式：0 自动模式，1 用户模式，默认为 0
s16NrIntensity	降噪力度配置。
s16NoiseDbThr	噪声门限配置。
s8SpProSwitch	音乐检测开关。
s32Reserved	保留。

【注意事项】

无。

【相关数据类型及接口】

无。

AUDIO_HPF_CONFIG_S

【说明】

定义音频高通滤波功能配置信息结构体。

【定义】

```
typedef struct AUDIO_HPF_CONFIG_S
{
    BOOL bUsrMode;
    AUDIO_HPF_FREQ_E enHpfFreq;
} AUDIO_HPF_CONFIG_S;
```

【成员】

成员名称	描述	其它说明
bUsrMode	是否采用用户模式：0 自动模式，1 用户模式，默认为 0	暂不支持。
enHpfFreq	高通滤波截止频率选择。	暂不支持。

【注意事项】

无。

【相关数据类型及接口】

无。

AI_RNR_CONFIG_S

【说明】

定义音频输入高保真噪声抑制功能配置信息结构体。

【定义】

```
typedef struct AI_RNR_CONFIG_S
{
    int sMaxNoiseSuppression;
    int sOverlapPercent;
    int sNonstat;
} AI_RNR_CONFIG_S;
```

【成员】

成员名称	其它说明
sMaxNoiseSuppression	暂不支持。
sOverlapPercent	暂不支持。
sNonstat	暂不支持。

【注意事项】

无。

【相关数据类型及接口】

无。

AUDIO_EQ_CONFIG_S

【说明】

定义音频均衡器功能配置信息结构体。

【定义】

```
typedef struct AUDIO_EQ_CONFIG_S
{
    short s16GaindB[VQE_EQ_BAND_NUM];
    int s32Reserved;
} AUDIO_EQ_CONFIG_S;
```

【成员】

成员名称	描述
s8GaindB	EQ 频段增益调节。
s32Reserved	保留。

【注意事项】

无。

【相关数据类型及接口】

无。

AI_VQE_CONFIG_S

【说明】

定义音频输入声音质量增强配置信息结构体。

【定义】

```
typedef struct AI_VQE_CONFIG_S
{
    int                 bHpfOpen;
    int                 bAecOpen;
    int                 bAnrOpen;
    int                 bRnrOpen;
    int                 bAgcOpen;
    int                 bEqOpen;
    int                 bDrcOpen;
    int                 s32WorkSampleRate;
    int                 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_DRC_CONFIG_S      stDrcCfg;
} AI_VQE_CONFIG_S;
```

【成员】

成员名称	描述
bHpfOpen	高通滤波功能是否使能标志。
bAecOpen	回声抵消功能是否使能标志。
bAnrOpen	环境噪声抑制功能是否使能标志。
bRnrOpen	高保真噪声抑制功能是否使能标志。
bAgcOpen	自动电平控制功能是否使能标志
bEqOpen	均衡器功能是否使能标志
bDrcOpen	录音宽动态功能是否使能标志
s32WorkSampleRate	工作采样频率。
s32FrameSample	VQE 的帧长，即采样点数目。
enWorkstate	工作模式。
stHpfCfg	高通滤波功能相关配置信息。
stAecCfg	回声抵消功能相关配置信息。
stAnrCfg	环境噪声抑止功能相关配置信息。
stRnrCfg	高保真噪声抑制功能相关配置信息。
stAgcCfg	自动电平控制相关配置信息。
stEqCfg	均衡器相关配置信息。
stDrcCfg	录音宽动态相关配置信息。

【注意事项】

无。

【相关数据类型及接口】

无。

AO_VQE_CONFIG_S

【说明】

定义音频输出声音质量增强配置信息结构体。

【定义】

```
typedef struct AO_VQE_CONFIG_S
{
    int                 bHpfOpen;
    int                 bAnrOpen;
    int                 bAgcOpen;
    int                 bEqOpen;
    int                 bGainOpen;
    int                 s32WorkSampleRate;
    int                 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;
    AUDIO_GAIN_CONFIG_S   stGainCfg;
} AO_VQE_CONFIG_S;
```

【成员】

成员名称	描述
bHpfOpen	高通滤波功能是否使能标志。
bAnrOpen	降噪功能是否使能标志。
bAgcOpen	自动电平控制功能是否使能标志
bEqOpen	均衡器功能是否使能标志
bGainOpen	增益功能是否使能标志
s32WorkSampleRate	工作采样频率。
s32FrameSample	VQE 的帧长，即采样点数目。
enWorkstate	工作模式。
stHpfCfg	高通滤波功能相关配置信息。

stAnrCfg	环境噪声抑止功能相关配置信息。
stAgcCfg	自动电平控制相关配置信息。
stEqCfg	均衡器相关配置信息。
stGainCfg	增益相关配置信息。

【注意事项】

无。

【相关数据类型及接口】

无。

AUDIO_STREAM_S

【说明】

定义音频码流结构体。

【定义】

```
typedef struct AUDIO_STREAM_S
{
    unsigned char       *pStream;
    unsigned int        mLen;
    unsigned long long  mTimeStamp;
    unsigned int        mId;
} AUDIO_STREAM_S;
```

【成员】

成员名称	描述
pStream	音频码流数据指针。
mLen	音频码流长度。单位为 byte。
mTimeStamp	音频码流时间戳。
mId	音频码流序号。

【注意事项】

无。

【相关数据类型及接口】

无。

AO_CHN_STATE_S

【说明】

音频输出通道的数据缓存状态结构体。

【定义】

```
typedef struct AO_CHN_STATE_S
{
    unsigned int        u32ChnTotalNum;
    unsigned int        u32ChnFreeNum;
    unsigned int        u32ChnBusyNum;
} AO_CHN_STATE_S;
```

【成员】

成员名称	描述
u32ChnTotalNum	输出通道总的缓存块数。
u32ChnFreeNum	空闲缓存块数。
u32ChnBusyNum	被占用缓存块数。

【注意事项】

无。

【相关数据类型及接口】

无。

AUDIO_FADE_S

【说明】

音频输出设备淡入淡出配置结构体。

【定义】

```
typedef struct AUDIO_FADE_S
{
    BOOL                bFade;
    AUDIO_FADE_RATE_E enFadeInRate;
```

```
    AUDIO_FADE_RATE_E enFadeOutRate;
} AUDIO_FADE_S;
```

【成员】

成员名称	描述
bFade	是否开启淡入淡出功能。
enFadeInRate	音频输出设备音量淡入速度。
enFadeOutRate	音频输出设备音量淡出速度。

【注意事项】

无。

【相关数据类型及接口】

无。

AUDIO_SAMPLE_RATE_E

【说明】

定义音频采样率。

【定义】

```
typedef enum AUDIO_SAMPLE_RATE_E
{
    AUDIO_SAMPLE_RATE_8000    = 8000,    /* 8K samplerate*/
    AUDIO_SAMPLE_RATE_12000   = 12000,   /* 12K samplerate*/
    AUDIO_SAMPLE_RATE_11025   = 11025,   /* 11.025K samplerate*/
    AUDIO_SAMPLE_RATE_16000   = 16000,   /* 16K samplerate*/
    AUDIO_SAMPLE_RATE_22050   = 22050,   /* 22.050K samplerate*/
    AUDIO_SAMPLE_RATE_24000   = 24000,   /* 24K samplerate*/
    AUDIO_SAMPLE_RATE_32000   = 32000,   /* 32K samplerate*/
    AUDIO_SAMPLE_RATE_44100   = 44100,   /* 44.1K samplerate*/
    AUDIO_SAMPLE_RATE_48000   = 48000,   /* 48K samplerate*/
} AUDIO_SAMPLE_RATE_E;
```

【成员】

成员名称	描述
AUDIO_SAMPLE_RATE_8000	8kHz 采样率
AUDIO_SAMPLE_RATE_12000	12kHz 采样率
AUDIO_SAMPLE_RATE_11025	11.025kHz 采样率
AUDIO_SAMPLE_RATE_16000	16kHz 采样率
AUDIO_SAMPLE_RATE_22050	22.05kHz 采样率
AUDIO_SAMPLE_RATE_24000	24kHz 采样率
AUDIO_SAMPLE_RATE_32000	32kHz 采样率
AUDIO_SAMPLE_RATE_44100	44.1kHz 采样率
AUDIO_SAMPLE_RATE_48000	48kHz 采样率

【注意事项】

无。

【相关数据类型及接口】

无。

AUDIO_BIT_WIDTH_E

【说明】

定义音频采样精度。

【定义】

```
typedef enum AUDIO_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_32  = 3,    /* 32bit width*/
} AUDIO_BIT_WIDTH_E;
```

【成员】

成员名称	描述
AUDIO_BIT_WIDTH_8	采样精度为 8bit 位宽。
AUDIO_BIT_WIDTH	采样精度为 16bit 位宽。

_16	
AUDIO_BIT_WIDTH_24	采样精度为 24bit 位宽。
AUDIO_BIT_WIDTH_32	采样精度为 32bit 位宽。

【注意事项】

无。

【相关数据类型及接口】

无。

AIO_MODE_E

【说明】

定义音频保存文件功能配置信息结构体。

【定义】

```
typedef enum AIO_MODE_E
{
    AIO_MODE_I2S_MASTER    = 0,     /* AIO I2S master mode */
    AIO_MODE_I2S_SLAVE,             /* AIO I2S slave mode */
    AIO_MODE_PCM_SLAVE_STD,         /* AIO PCM slave standard mode */
    AIO_MODE_PCM_SLAVE_NSTD,        /* AIO PCM slave non-standard mode */
    AIO_MODE_PCM_MASTER_STD,        /* AIO PCM master standard mode */
    AIO_MODE_PCM_MASTER_NSTD,       /* AIO PCM master non-standard mode */
    AIO_MODE_BUTT
} AIO_MODE_E;
```

【成员】

成员名称	描述	其它说明
AIO_MODE_I2S_MASTER	I2S 主模式	暂不支持。
AIO_MODE_I2S_SLAVE	I2S 从模式	暂不支持。
AIO_MODE_PCM_SLAVE_STD	PCM 从模式(标准协议)	暂不支持。
AIO_MODE_PCM_SLAVE_NSTD	PCM 从模式（自定义协议）	暂不支持。
AIO_MODE_PCM_MASTER_ST	PCM 主模式(标准协议)	暂不支持。

D		
AIO_MODE_PCM_MASTER_NSTD	PCM 主模式（自定义协议）	暂不支持。

【注意事项】

无。

【相关数据类型及接口】

无。

AIO_SOUND_MODE_E

【说明】

定义音频保存文件功能配置信息结构体。

【定义】

```
typedef enum AIO_SOUND_MODE_E
{
    AUDIO_SOUND_MODE_MONO     =0, /*mono*/
    AUDIO_SOUND_MODE_STEREO   =1, /*stereo*/
} AUDIO_SOUND_MODE_E;
```

【成员】

成员名称	描述
AUDIO_SOUND_MODE_MONO	单声道
AUDIO_SOUND_MODE_STEREO	双声道

【注意事项】

无。

【相关数据类型及接口】

无。

AUDIO_HPF_FREQ_E

【说明】

定义音频保存文件功能配置信息结构体。

【定义】

```
typedef enum AUDIO_HPF_FREQ_E
{
    AUDIO_HPF_FREQ_80    = 80,    /* 80Hz */
    AUDIO_HPF_FREQ_120   = 120,   /* 120Hz */
    AUDIO_HPF_FREQ_150   = 150,   /* 150Hz */
} AUDIO_HPF_FREQ_E;
```

【成员】

成员名称	描述
AUDIO_HPF_FREQ_80	截止频率为 80Hz。
AUDIO_HPF_FREQ_120	截止频率为 120Hz。
AUDIO_HPF_FREQ_150	截止频率为 150Hz。

【注意事项】

无。

【相关数据类型及接口】

无。

AQE_WORKSTATE_E

【说明】

定义音频保存文件功能配置信息结构体。

【定义】

```
typedef enum VQE_WORKSTATE_E
{
    VQE_WORKSTATE_COMMON   = 0,
    VQE_WORKSTATE_MUSIC    = 1,
    VQE_WORKSTATE_NOISY    = 2,
} VQE_WORKSTATE_E;
```

【成员】

成员名称	描述
VQE_WORKSTATE_COMMON	一般模式。
VQE_WORKSTATE_MUSIC	音乐模式。
VQE_WORKSTATE_NOISY	噪声模式。

【注意事项】

无。

【相关数据类型及接口】

无。

AUDIO_TRACK_MODE_E

【说明】

定义音频设备声道模式类型。

【定义】

```
typedef enum AUDIO_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_MODE_E;
```

【成员】

成员名称	描述	其它说明
AUDIO_TRACK_NORMAL	正常模式，不做处理	支持。
AUDIO_TRACK_BOTH_L	两个声道全部为左声道声音	暂不支持。

EFT		
AUDIO_TRACK_BOTH_RIGHT	两个声道全部为右声道声音	暂不支持。
AUDIO_TRACK_EXCHANGE	左右声道数据互换，左声道为右声道声音，右声道为左声道声音	暂不支持。
AUDIO_TRACK_MIX	左右两个声道输出为左右声道相加（混音）	暂不支持。
AUDIO_TRACK_LEFT_MUTE	左声道静音，右声道播放原右声道声音	暂不支持。
AUDIO_TRACK_RIGHT_MUTE	右声道静音，左声道播放原左声道声音	暂不支持。
AUDIO_TRACK_BOTH_MUTE	左右声道均静音	暂不支持。

【注意事项】

无。

【相关数据类型及接口】

无。

AUDIO_FADE_RATE_E

【说明】

定义音频输出设备淡入淡出速度类型。

【定义】

```
typedef enum AUDIO_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_E;
```

【成员】

成员名称	描述
AUDIO_FADE_RATE_1	1 个采样点改变一次
AUDIO_FADE_RATE_2	2 个采样点改变一次
AUDIO_FADE_RATE_4	4 个采样点改变一次
AUDIO_FADE_RATE_8	8个采样点改变一次
AUDIO_FADE_RATE_16	16 个采样点改变一次
AUDIO_FADE_RATE_32	32 个采样点改变一次
AUDIO_FADE_RATE_64	64 个采样点改变一次
AUDIO_FADE_RATE_128	128 个采样点改变一次

【注意事项】

无。

【相关数据类型及接口】

无。

G726_BPS_E

【说明】

定义 G.726 编解码协议速率。

【定义】

```
typedef enum G726_BPS_E
{
    G726_16K = 0,          /* G726 16kbps, see RFC3551.txt    4.5.4 G726-16 */
    G726_24K,              /* G726 24kbps, see RFC3551.txt    4.5.4 G726-24 */
    G726_32K,              /* G726 32kbps, see RFC3551.txt    4.5.4 G726-32 */
```

```
    G726_40K,           /* G726 40kbps, see RFC3551.txt    4.5.4 G726-40 */
    MEDIA_G726_16K,     /* G726 16kbps for ASF ... */
    MEDIA_G726_24K,     /* G726 24kbps for ASF ... */
    MEDIA_G726_32K,     /* G726 32kbps for ASF ... */
    MEDIA_G726_40K,     /* G726 40kbps for ASF ... */
} G726_BPS_E;
```

【成员】

成员名称	描述	其它说明
G726_16K	16kbit/s G.726。	暂不支持。
G726_24K	24kbit/s G.726。	暂不支持。
G726_32K	32kbit/s G.726。	暂不支持。
G726_40K	40kbit/s G.726。	暂不支持。
MEDIA_G726_16K	G726 16kbit/s for ASF。	暂不支持。
MEDIA_G726_24K	G726 24kbit/s for ASF。	暂不支持。
MEDIA_G726_32K	G726 32kbit/s for ASF。	暂不支持。
MEDIA_G726_40K	G726 40kbit/s for ASF。	暂不支持。

【注意事项】

无。

【相关数据类型及接口】

无。

ADPCM_TYPE_E

【说明】

定义 ADPCM 编解码协议类型。

【定义】

```
typedef enum ADPCM_TYPE_E
{
    ADPCM_TYPE_DVI4 = 0,
```

```
    ADPCM_TYPE_IMA,
} ADPCM_TYPE_E;
```

【成员】

成员名称	描述	其它说明
ADPCM_TYPE_DVI4	32kbit/s ADPCM(DVI4)。	暂不支持。
ADPCM_TYPE_IMA	32kbit/s ADPCM(IMA)。	暂不支持。

【注意事项】

无。

【相关数据类型及接口】

无。

9.5.2. 音频编码

AENC_CHN_ATTR_S

【说明】

定义音频编码通道属性结构体。

【定义】

```
typedef struct AENC_CHN_ATTR_S
{
    AENC_ATTR_S AeAttr;
}AENC_CHN_ATTR_S;
typedef struct AENC_ATTR_S
{
    PAYLOAD_TYPE_E Type;
    int sampleRate;
    int channels;
    int bitRate;
    int bitsPerSample;
    int attachAACHeader;
}AENC_ATTR_S;
```

【成员】

成员名称	描述	其它说明
Type	音频解码协议类型。	支持。
sampleRate	音频数据采样率。	支持。
channels	通道数量(单或双通道)。	支持。
bitRate	编码码率。	支持。
bitsPerSample	采样位宽。	支持。
attachAACHeader	编码为AAC的数据是否添加文件头信息。	支持。

【注意事项】

- 编码通道属性至少需填充四个field：Type、sampleRate、channels、bitsPerSample。
- 在AAC编码时，当通过网络传输希望每帧音频码流带头信息时，其中需设置attachAACHeader为1，当直接送muxer而无需码流头信息时，设置attachAACHeader为0即可。
- 在G726编码时，需设置bitRate参数(16k/24k/32k/40k)，用于调整G726编码输出数据宽度(2bit/3bit/4bit/5bit)。如果设置为0，编码数据宽度自动调整为2bit。

【相关数据类型及接口】

无。

9.5.3. 音频解码

ADEC_CHN_ATTR_S

【说明】

定义音频解码通道属性结构体。

【定义】

```
typedef struct ADEC_CHN_ATTR_S
{
    PAYLOAD_TYPE_E mType;
    int sampleRate;
    int channels;
    int bitRate;            // not use
    int bitsPerSample;
    int attachAACHeader;
```

}ADEC_CHN_ATTR_S;

【成员】

成员名称	描述	其它说明
mType	音频解码协议类型。	支持。
sampleRate	音频数据采样率。	支持。
channels	通道数量(单或双通道)。	支持。
bitsPerSample	采样位宽。	支持。
attachAACHeader	用于码流类型为 AAC 的数据。	支持。

【注意事项】

无。

【相关数据类型及接口】

无。

9.5.4. 音频编解码器类型与数据格式要求

编码器类型	输入 pcm 格式	输出
aac	8k~48k 采样率,单/双通道,16 位宽	压缩率为 10 左右。8k/单通道下输出约 16kbps。
mp3	8k~48k 采样率,单/双通道,16 位宽	16k/24k/128kbps 由应用控制。默认 16kbps。
adpcm	8k 采样率,单通道,16 位宽	压缩率为 4。输出码率 32kbps。
g711a/u	8k 采样率,单通道,16 位宽	压缩率为 2。输出码率 64kbps。
g726	8k 采样率,单通道,16 位宽	压缩 16->2/3/4/5bit，分别对应输出 16k/24k/32k/40kbps，由应用控制。
pcma	8k~48k 采样率,单/双通道,16 位宽	不进行压缩。根据输入原样输出数据。

解码器类型	输入码流格式	输出
aac	8k~192k 采样率，单/双通道，16 位宽	根据输入调整输出。
mp3	8k~192k 采样率，单/双通道，16 位宽	根据输入调整输出。
adpcm	8k 采样率，单通道，16 位宽	8k 采样率、16 位宽。
g711a/u	8k 采样率，单通道，16 位宽	8k 采样率、16 位宽。

g726	8k 采样率, 单通道, 16 位宽	8k 采样率、16 位宽。
pcma	8k~192k 采样率, 单/双通道, 16 位宽	根据输入调整输出。

注意：

不同编码器对输入数据格式要求不同。acc 和 mp3 编码器对输入数据格式要求只需保证 16 位宽，对通道数量、采样率没有要求。adpcm、g711a/u、g726 编码器则对输入 pcm 数据格式有较多限制，必须同时满足 8k 采样率、单通道、16 位宽的要求(编码标准要求)。pcma 编码为非压缩编码器，在内部实现为直接将输入数据送输出。在做 ai-aenc 开发时，如果需要输出 adpcm、g711a/u、g726 格式的数据，那么在 AI 设备属性需设置 8k 采样率/单通道/16 位宽的参数。

目前，AIO 设备的数据位宽(bitwidth)只支持 16 位，8、24、32 位宽大小暂不支持。

9.6. 错误码

9.6.1. 音频输入错误码

错误码	宏定义	描述
0xA0158001	ERR_AI_INVALID_DEVID	音频输入设备号无效
0xA0158002	ERR_AI_INVALID_CHNID	音频输入通道号无效
0xA0158003	ERR_AI_ILLEGAL_PARAM	音频输入参数设置无效
0xA0158006	ERR_AI_NULL_PTR	输入参数空指针错误
0xA0158007	ERR_AI_NOT_CONFIG	音频输入设备属性未设置
0xA0158008	ERR_AI_NOT_SUPPORT	操作不支持
0xA0158009	ERR_AI_NOT_PERM	操作不允许
0xA0158005	ERR_AI_NOT_ENABLED	音频输入设备或通道没有使能
0xA015800C	ERR_AI_NOMEM	分配内存失败
0xA015800D	ERR_AI_NOBUF	音频输入缓存不足
0xA015800E	ERR_AI_BUF_EMPTY	音频输入缓存为空
0xA015800F	ERR_AI_BUF_FULL	音频输入缓存为满
0xA0158010	ERR_AI_SYS_NOTREADY	音频输入系统未初始化
0xA0158012	ERR_AI_BUSY	音频输入系统忙

9.6.2. 音频输出错误码

错误码	宏定义	描述
0xA0168001	ERR_AO_INVALID_DEVID	音频输出设备号无效

0xA0168002	ERR_AO_INVALID_CHNID	音频输出通道号无效
0xA0168003	ERR_AO_ILLEGAL_PARAM	音频输出参数设置无效
0xA0168006	ERR_AO_NULL_PTR	音频输出参数空指针错误
0xA0168007	ERR_AO_NOT_CONFIG	音频输出设备属性未设置
0xA0168008	ERR_AO_NOT_SUPPORT	操作不支持
0xA0168009	ERR_AO_NOT_PERM	操作不允许
0xA0168005	ERR_AO_NOT_ENABLED	音频输出设备或通道没有使能
0xA016800C	ERR_AO_NOMEM	系统内存不足
0xA016800D	ERR_AO_NOBUF	音频输出缓存不足
0xA016800E	ERR_AO_BUF_EMPTY	音频输出缓存为空
0xA016800F	ERR_AO_BUF_FULL	音频输出缓存为满
0xA0168010	ERR_AO_SYS_NOTREADY	音频输出系统未初始化
0xA0168012	ERR_AO_BUSY	音频输出系统忙

9.6.3. 音频编码错误码

错误码	宏定义	描述
0xA0178001	ERR_AENC_INVALID_DEVID	音频编码设备号无效
0xA0178002	ERR_AENC_INVALID_CHNID	音频编码通道号无效
0xA0178003	ERR_AENC_ILLEGAL_PARAM	音频编码参数设置无效
0xA0178004	ERR_AENC_EXIST	音频编码通道已经创建
0xA0178005	ERR_AENC_UNEXIST	音频编码通道未创建
0xA0178006	ERR_AENC_NULL_PTR	输入参数空指针错误
0xA0178007	ERR_AENC_NOT_CONFIG	编码通道未配置
0xA0178008	ERR_AENC_NOT_SUPPORT	操作不被支持
0xA0178009	ERR_AENC_NOT_PERM	操作不允许
0xA017800C	ERR_AENC_NOMEM	系统内存不足
0xA017800D	ERR_AENC_NOBUF	编码通道缓存分配失败
0xA017800E	ERR_AENC_BUF_EMPTY	编码通道缓存空
0xA017800F	ERR_AENC_BUF_FULL	编码通道缓存满
0xA0178010	ERR_AENC_SYS_NOTREADY	系统没有初始化
	ERR_AENC_ENCODER_ERR	音频编码数据错误

9.6.4. 音频解码错误码

错误码	宏定义	描述

0xA0188001	ERR_ADEC_INVALID_DEVID	音频解码设备号无效
0xA0188002	ERR_ADEC_INVALID_CHNID	音频解码通道号无效
0xA0188003	ERR_ADEC_ILLEGAL_PARAM	音频解码参数设置无效
0xA0188004	ERR_ADEC_EXIST	音频解码通道已经创建
0xA0188005	ERR_ADEC_UNEXIST	音频解码通道未创建
0xA0188006	ERR_ADEC_NULL_PTR	输入参数空指针错误
0xA0188007	ERR_ADEC_NOT_CONFIG	解码通道属性未配置
0xA0188008	ERR_ADEC_NOT_SUPPORT	操作不被支持
0xA0188009	ERR_ADEC_NOT_PERM	操作不允许
0xA018800C	ERR_ADEC_NOMEM	系统内存不足
0xA018800D	ERR_ADEC_NOBUF	解码通道缓存分配失败
0xA018800E	ERR_ADEC_BUF_EMPTY	解码通道缓存空
0xA018800F	ERR_ADEC_BUF_FULL	解码通道缓存满
0xA0188010	ERR_ADEC_SYS_NOTREADY	系统没有初始化
	ERR_ADEC_DECODER_ERR	音频解码数据错误

10. Region 模块

10.1. 概述

用户一般都需要在视频中叠加 OSD 用于显示一些特定的信息（如：通道号、时间戳等），必要时还会填充色块。这些叠加在视频上的 OSD 和遮挡在视频上的色块统称为区域。REGION 模块，用于统一管理这些区域资源。

区域管理可以实现区域的创建，并叠加到视频中或对视频进行遮挡。例如，实际应用中，用户通过创建一个区域，通过 AW_MPI_RGN_AttachToChn，将该区域叠加到某个通道（如 VENC 通道）中。在通道进行调度时，则会将 OSD 叠加在视频中。一个区域支持通过设置通道显示属性接口指定到多个通道中（如：多个 VENC 通道，多个 VideoScaler 通道，甚至多个 VENC 和 VideoScaler 通道），且支持在每个通道的显示属性（如位置、透明度等）都不同。

10.2. 功能描述

支持区域叠加(overlay)和区域遮挡(cover)两种方式。其中叠加支持位图加载、反色功能等功能，遮挡则支持纯色块的遮挡。

区域在不同通道拥有不同的通道显示属性，比如显示位置、层次和区域是否显示等属性。

10.2.1. 状态

本组件没有内部线程，所以没有状态转换。

10.3. API 参考

区域管理模块提供以下 MPI:

- AW_MPI_RGN_Create:创建区域
- AW_MPI_RGN_Destrory：销毁区域
- AW_MPI_RGN_GetAttr：获取区域属性
- AW_MPI_RGN_SetAttr：设置区域属性
- AW_MPI_RGN_SetBitMap：设置区域位图
- AW_MPI_RGN_AttachToChn：将区域叠加到通道上
- AW_MPI_RGN_DetachFrmChn：将区域从通道中撤出
- AW_MPI_RGN_SetDisplayAttr：设置区域的通道显示属性
- AW_MPI_RGN_GetDisplayAttr：获取区域的通道显示属性

AW_MPI_RGN_Create

【描述】

创建区域。

【语法】

ERRORTYPE AW_MPI_RGN_Create(RGN_HANDLE Handle, const RGN_ATTR_S *pstRegion);

【参数】

参数名称	描述	输入/输出
Handle	区域句柄号。 必须是未使用的 Handle 号。 取值范围：[0, RGN_HANDLE_MAX)。	输入
pstRegion	区域属性指针。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【需求】

- 头文件：mpi_region.h
- 库文件：libmedia_mpp.so

【注意】

- 创建 Cover 时，只需指定区域类型即可。其它的属性，如区域位置，层次等信息在调用 AW_MPI_RGN_AttachToChn 接口时指定。
- 创建区域时，本接口只进行基本的参数的检查，譬如：最小宽高，最大宽高等；当区域 attach 到通道上时，根据各通道模块支持类型的约束条件进行更加有针对性的参数检查，譬如支持的像素格式等。
- 对于准备添加到 VENC 通道的区域，其区域的宽高，位置坐标参数必须能是 16 的倍数，否则无法添加到 VENC 通道。

【举例】

无。

AW_MPI_RGN_Destroy

【描述】

销毁区域。

【语法】

ERRORTYPE AW_MPI_RGN_Destroy(RGN_HANDLE Handle);

【参数】

参数名称	描述	输入/输出
Handle	区域句柄号。 取值范围：[0,	输入

	RGN_HANDLE_MAX)。	

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【需求】

- 头文件：mpi_region.h
- 库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

AW_MPI_RGN_GetAttr

【描述】

获取区域属性。

【语法】

ERRORTYPE AW_MPI_RGN_GetAttr(RGN_HANDLE Handle, RGN_ATTR_S *pstRegion);

【参数】

参数名称	描述	输入/输出
Handle	区域句柄号。 取值范围：[0, RGN_HANDLE_MAX)。	输入
pstRegion	区域属性指针。	输出

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【需求】

- 头文件：mpi_region.h
- 库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

AW_MPI_RGN_SetAttr

【描述】

设置区域属性。

【语法】

ERRORTYPE AW_MPI_RGN_SetAttr(RGN_HANDLE Handle, const RGN_ATTR_S *pstRegion);

【参数】

参数名称	描述	输入/输出
Handle	区域句柄号。 取值范围：[0, RGN_HANDLE_MAX)。	输入
pstRegion	区域属性指针。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【需求】

- 头文件：mpi_region.h
- 库文件：libmedia_mpp.so

【注意】

- 当区域通过 AW_MPI_RGN_AttachToChn 接口绑定到通道上时，本接口不可以用于修改静态属性，但是可以修改动态属性；当区域没有 attach 到任何通道上时，本接口即可用于修改静态属性，也可用于修改动态属性。

【举例】

无。

AW_MPI_RGN_SetBitMap

【描述】

设置区域位图，即对区域进行位图填充。

【语法】

ERRORTYPE AW_MPI_RGN_SetBitMap(RGN_HANDLE Handle, const BITMAP_S *pstBitmap);

【参数】

参数名称	描述	输入/输出
Handle	区域句柄号。	输入

	取值范围：[0, RGN_HANDLE_MAX)。	
pstBitmap	位图属性指针。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【需求】

- 头文件：mpi_region.h
- 库文件：libmedia_mpp.so

【注意】

- 只对 OVERLAY 类型的区域有效。
- 位图像素格式必须与区域像素格式一致。
- 当位图大小与区域大小不一致时，区域大小将被修改为与位图大小保持一致。

【举例】

无。

AW_MPI_RGN_AttachToChn

【描述】

将区域叠加到通道上。

【语法】

ERRORTYPE AW_MPI_RGN_AttachToChn(RGN_HANDLE Handle, const MPP_CHN_S *pstChn, const RGN_CHN_ATTR_S *pstChnAttr);

【参数】

参数名称	描述	输入/输出
Handle	区域句柄号。 取值范围：[0, RGN_HANDLE_MAX)。	输入
pstChn	通道结构体指针。	输入
pstChnAttr	区域通道显示属性指针。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【需求】

- 头文件：mpi_region.h
- 库文件：libmedia_mpp.so

【注意】

- 目前为止，只能将区域叠加到 VI 和 VENC 两个通道上。

【举例】

无。

AW_MPI_RGN_DetachFromChn

【描述】

将区域从通道中撤出。

【语法】

ERRORTYPE AW_MPI_RGN_DetachFromChn(RGN_HANDLE Handle, const MPP_CHN_S *pstChn);

【参数】

参数名称	描述	输入/输出
Handle	区域句柄号。 取值范围：[0, RGN_HANDLE_MAX)。	输入
pstChn	通道结构体指针。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【需求】

- 头文件：mpi_region.h
- 库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

AW_MPI_RGN_SetDisplayAttr

【描述】

设置区域的通道显示属性。

【语法】

ERRORTYPE AW_MPI_RGN_SetDisplayAttr(RGN_HANDLE Handle, const MPP_CHN_S *pstChn, const RGN_CHN_ATTR_S *pstChnAttr);

【参数】

参数名称	描述	输入/输出
Handle	区域句柄号。 取值范围：[0, RGN_HANDLE_MAX)。	输入
pstChn	通道结构体指针。	输入
pstChnAttr	区域通道显示属性指针。	输入

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【需求】

- 头文件：mpi_region.h
- 库文件：libmedia_mpp.so

【注意】

- 静态属性不能修改，动态属性可以修改。

【举例】

无。

AW_MPI_RGN_GetDisplayAttr

【描述】

获取区域的通道显示属性。

【语法】

ERRORTYPE AW_MPI_RGN_GetDisplayAttr(RGN_HANDLE Handle, const MPP_CHN_S *pstChn, RGN_CHN_ATTR_S *pstChnAttr);

【参数】

参数名称	描述	输入/输出

Handle	区域句柄号。 取值范围：[0, RGN_HANDLE_MAX)。	输入
pstChn	通道结构体指针。	输入
pstChnAttr	区域通道显示属性指针。	输出

【返回值】

返回值	描述
0	成功
非 0	失败，参见错误码。

【需求】

- 头文件：mpi_region.h
- 库文件：libmedia_mpp.so

【注意】

无。

【举例】

无。

10.4. 数据类型

RGN_TYPE_E

【说明】

定义区域类型

【定义】

```
typedef enum RGN_TYPE_E
{
    OVERLAY_RGN   = 0,      /* video overlay region */
    COVER_RGN,
    COVEREX_RGN,
    OVERLAYEX_RGN,
    RGN_BUTT
} RGN_TYPE_E;
```

【成员】

成员名称	描述
OVERLAY_RGN	通道视频叠加区域。
COVER_RGN	通道视频遮挡区域。

COVEREX_RGN	扩展视频遮挡区域。不支持。
OVERLAYEX_RGN	扩展视频遮挡叠加区域。不支持。

【注意事项】

- 目前为止，COVEREX_RGN 和 OVERLAYEX_RGN 不可用，是保留项。

【相关数据类型及接口】

无。

RGN_AREA_TYPE_E

【说明】

定义 COVER、COVEREX_RGN 类型。

【定义】

```
typedef enum RGN_AREA_TYPE_E
{
    AREA_RECT    =  0,
    AREA_QUAD_RANGLE,
    AREA_BUTT
} RGN_AREA_TYPE_E;
```

【成员】

成员名称	描述
AREA_RECT	矩形区域。
AREA_QUAD_RANGLE	任意四边形区域。不支持。

【注意事项】

无。

【相关数据类型及接口】

无。

OVERLAY_ATTR_S

【说明】

定义通道叠加区域属性结构体。

【定义】

```
typedef struct OVERLAY_ATTR_S
{
    PIXEL_FORMAT_E          mPixelFmt;
    unsigned int            mBgColor;
```

```
    SIZE_S                  mSize;
}OVERLAY_ATTR_S;
```

【成员】

成员名称	描述
mPixelFmt	像素格式，只支持 ARGB1555 和 ARGB8888 两种格式。
mBgColor	未使用。
mSize	区域宽高大小， 宽度：[4，4096]，要求以 2 对齐。 高度：宽度：[4，4096]，要求以 2 对齐。

【注意事项】

无。

【相关数据类型及接口】

无。

OVERLAY_INVERT_COLOR_S

【说明】

定义 OSD 反色相关属性。

【定义】

```
typedef struct OVERLAY_INVERT_COLOR_S
{
SIZE_S                  stInvColArea;
unsigned int            mLumThresh;
    INVERT_COLOR_MODE_E     enChgMod;
    BOOL                    bInvColEn;
}OVERLAY_INVERT_COLOR_S;
```

【成员】

成员名称	描述
stInvColArea	单元反色区域，反色处理的基本单元。 取值范围： 高度：[16，64]，需要 16 对齐。 宽度：[16，64]，需要 16 对齐。 未使用。V316 不支持单元反色区域。
mLumThresh	亮度阀值。暂未使用。
enChgMod	OSD 反色触发模式。暂未使用。

bInvColEn	OSD 反色开关。TRUE：开启反色；FALSE：关闭反色。

【注意事项】

无。

【相关数据类型及接口】

无。

OVERLAY_CHN_ATTR_S

【说明】

定义通道叠加区域的通道显示属性。

【定义】

```
typedef struct OVERLAY_CHN_ATTR_S
{
    POINT_S                 stPoint;
    unsigned int            mFgAlpha;
    unsigned int            mBgAlpha;
    unsigned int            mLayer;
    OVERLAY_QP_INFO_S       stQpInfo;
    OVERLAY_INVERT_COLOR_S  stInvertColor;
}OVERLAY_CHN_ATTR_S;
```

【成员】

成员名称	描述
stPoint	区域位置： 水平位置 X:[0,4096],要求以 4 对齐。 垂直位置 Y:[0，4636],要求以 4 对齐。
u32FgAlpha	Alpha 位为 1 的像素点的透明度（前景 Alpha ），取值范围为[0,128]，值越小，越透明。只针对像素格式为 MM_PIXEL_FORMAT_RGB_1555 的 bmp 图有意义，设置全局 alpha。对所有 overlay 有效。所以以最后一次设置的 overlay 的通道属性为准。
u32BgAlpha	Alpha 位为 0 的像素点的透明度（背景 Alpha ），取值范围为[0,128]，值越小，越透明。未使用。
u32Layer	区域层次，取值范围为：[0,63]，值越大，层次越高。
stQpInfo	区域编码使用的 QP 值，未使用。
stInvertColor	区域反色配置信息。

【注意事项】

- 建议用户把 Overlay 的起始位置和宽高限定为 16 对齐。

【相关数据类型及接口】

无。

COVER_CHN_ATTR_S

【说明】

定义通道遮挡区域的通道显示属性。

【定义】

```
typedef struct COVER_CHN_ATTR_S
{
    RGN_AREA_TYPE_E          enCoverType;
    union
    {
        RECT_S               stRect;
        RGN_QUADRANGLE_S     stQuadRangle;
    };
    unsigned int             mColor;
unsigned int                 mLayer;
}COVER_CHN_ATTR_S;
```

【成员】

成员名称	描述
enCoverType	只支持 AREA_RECT。
stRect	区域矩形位置，宽高。
stQuadRangle	区域任意四边形位置形状。不支持。
mColor	区域颜色。ARGB 格式。
mLayer	区域层次。范围：[0,7]。

【注意事项】

无。

【相关数据类型及接口】

无。

RGN_ATTR_U

【说明】

定义区域属性联合体。

【定义】

```
typedef union RGN_ATTR_U
{
OVERLAY_ATTR_S          stOverlay;
OVERLAYEX_ATTR_S        stOverlayEx;
} RGN_ATTR_U;
```

【成员】

成员名称	描述
stOverlay	通道叠加区域属性。
stOverlayEx	扩展叠加区域属性。不支持。

【注意事项】

无。

【相关数据类型及接口】

无。

RGN_CHN_ATTR_U

【说明】

定义区域通道显示属性联合体。

【定义】

```
typedef union RGN_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;
```

【成员】

成员名称	描述
stOverlayChn	叠加区域通道显示属性。
stCoverChn	遮挡区域通道显示属性。
stCoverExChn	扩展遮挡区域通道显示属性。不支持。

stOverlayExChn	扩展叠加区域通道显示属性。不支持。

【注意事项】

无。

【相关数据类型及接口】

无。

RGN_ATTR_S

【说明】

定义区域属性结构体。

【定义】

```
typedef struct RGN_ATTR_S
{
    RGN_TYPE_E      enType;
    RGN_ATTR_U      unAttr;
} RGN_ATTR_S;
```

【成员】

成员名称	描述
enType	区域类型。
unAttr	区域属性。

【注意事项】

无。

【相关数据类型及接口】

无。

RGN_CHN_ATTR_S

【说明】

定义区域属性结构体。

【定义】

```
typedef struct RGN_CHN_ATTR_S
{
    BOOL                bShow;
    RGN_TYPE_E          enType;
    RGN_CHN_ATTR_U      unChnAttr;
} RGN_CHN_ATTR_S;
```

【成员】

成员名称	描述
bShow	区域是否显示。
enType	区域类型。
unChnAttr	区域通道显示属性。

【注意事项】

无。

【相关数据类型及接口】

无。

10.5. 错误码

错误代码	宏定义	描述
0xA0038001	ERR_RGN_INVALID_DEVID	设备 ID 超出合法范围
0xA0038002	ERR_RGN_INVALID_CHNID	通道组号错误或无效区域句柄
0xA0038003	ERR_RGN_ILLEGAL_PARAM	参数超出合法范围
0xA0038004	ERR_RGN_EXIST	重复创建已存在的设备、通道或资源
0xA0038005	ERR_RGN_UNEXIST	试图使用或者销毁不存在的设备、通道或资源
0xA0038006	ERR_RGN_NULL_PTR	函数参数中有空指针
0xA0038007	ERR_RGN_NOT_CONFIG	模块没有配置。
0xA0038008	ERR_RGN_NOT_SUPPORT	不支持的参数或者功能
0xA0038009	ERR_RGN_NOT_PERM	该操作不允许，如试图修改静态配置参数
0xA003800C	ERR_RGN_NOMEM	分配内存失败，如系统内存不足。
0xA003800D	ERR_RGN_NOBUF	分配缓存失败，如申请的数据缓冲区太大
0xA003800E	ERR_RGN_BUF_EMPTY	缓冲区中无数据

0xA003800F	ERR_RGN_BUF_FULL	缓冲区中数据满
0xA0038011	ERR_RGN_BADADDR	地址非法。
0xA0038012	ERR_RGN_BUSY	系统忙
0xA0038010	ERR_RGN_NOTREADY	系统没有初始化或没有加载相应模块

11. Proc 调试节点用户指南

11.1. ISE 模块

[调试信息]

#cat /tmp/mpp/sunxi-ise

[ISE] Version:[V2.0 Debug]!

----------ISE MAIN PARAM----------

ISE MODULE CLK:432MHz

W_IN:2048

H_IN:2048

W_OUT:1024

H_OUT:1024

FLIP:FALSE

MIRR:FALSE

STRIDE_Y_OUT:1024

STRIDE_C_OUT:1024

FMT_OUT:YUV420

----------ISE SCALE PARAM----------

SCALE_CHN0_EN:TRUE

W_OUT_CHN0:512

H_OUT_CHN0:512

FLIP_CHN0:FALSE

MIRR_CHN0:FALSE

STRIDE_Y_OUT_CHN0:512

STRIDE_C_OUT_CHN0:512

SCALE_CHN1_EN:TRUE

W_OUT_CHN1:256

H_OUT_CHN1:256

FLIP_CHN1:FALSE

MIRR_CHN1:FALSE

STRIDE_Y_OUT_CHN1:256

STRIDE_C_OUT_CHN1:256

SCALE_CHN2_EN:TRUE

W_OUT_CHN2:256

H_OUT_CHN2:256

FLIP_CHN2:FALSE

MIRR_CHN2:FALSE

STRIDE_Y_OUT_CHN2:256

STRIDE_C_OUT_CHN2:256

----------ISE STATUS----------

INTERRUPT_TIMES:105

ISE_ERR_CNT:0

TIMEOUT_ERR:NO

ROI_ERR:NO

[调试信息分析]

记录 ISE 模块当前的配置信息和状态信息

参数		描述
ISE MAIN PARAM 主通道输出参数配置	W_IN	输入图像宽度
	H_IN	输入图像高度
	FMT_IN	输入图像格式
	W_OUT	主通道输出图像宽度
	H_OUT	主通道输出图像高度
	FLIP	输出图像是否上下翻转
	MIRR	输出图像是否镜像（水平翻转）
	STRIDE_Y_OUT	YUV 图像 Y 数据 32 位对齐宽度
	STRIDE_C_OUT	YUV 图像 UV 数据 32 位对齐宽度
	FMT_OUT	输出图像格式
ISE SCALE PARAM Scale 通道输出配置	SCALE_CHNn_EN	缩放通道的状态
	W_OUT_CHNn	缩放通道 n（0-2）宽度
	H_OUT_CHNn	缩放通道 n（0-2）高度
	FLIP_CHNn	缩放通道 n（0-2）否上下翻转（相较于 main 通道输出）
	MIRR_CHNn	缩放通道 n（0-2）否镜像（相较于 main 通道输出）
	STRIDE_Y_OUT_CHNn	YUV 图像 Y 数据 32 位对齐宽度

	STRIDE_C_OUT_CHNn	YUV 图像 UV 数据 32 位对齐宽度
ISE STATUS ISE 模块状态信息	INTERRUPT_TIMES	中断次数，
	TIMEOUT_ERR	硬件处理发生超时错误标志
	ROI_ERR	硬件处理发生 ROI 溢出错误标志

表 11-1

11.2. VI 模块

[调试信息]

注意：需要手动加载 sensor 模块才可以看到 VI 节点，比如 imx317_mipi.ko 等（根据板子上的 sensor 类型选择）。

#cat /tmp/mpp/vi

```
**********************************************************************
vi3, version: 1.0.0
pipe: imx317_mipi_2 => mipi1 => csi1 => isp0 => vipp3
input_win => hoff: 0, voff: 0, w: 1280, h: 720
output_width: 1280, output_height: 720
hflip: 0, vflip: 0
input_fmt: RGGB12, output_fmt: NV21M
interface: MIPI, isp_mode: NORMAL
buf_cnt: 10 buf_size: 352256 buf_rest: 10
frame_cnt: 168, frame_internal: 40(ms)
max_internal 40(ms), min_internal 40(ms)
vi_error_cnt: 0
**********************************************************************
```

参数		描述	
vi3		vi 通道号	
pipe		当前 vi 通道的 pipe，数据流的流过的模块	
input_win	hoff	CSI 输入窗口	水平偏移
	voff		垂直偏移
	w		宽度
	h		高度
output_width		输出宽度	
output_height		输出高度	

hflip	是否水平反转
vflip	是否垂直反转
input_fmt	输入格式 YUYV8/RGGB8/BGGR10/GBRG12 等
output_fmt	输出格式 YUV420M/NV21M/NV12M 等
interface	接口类型 PARALLEL/BT656/MIPI/SUBLVDS/HISPI
isp_mode	isp 的模式
buf_cnt	Video buffer 个数
buf_size	Video buffer 大小
buf_rest	当前驱动剩余的 buffer 个数
frame_cnt	当前接收到的帧计数
frame_internal	当前帧间隔(ms)
max_internal	最大帧间隔(ms)
min_internal	最小帧间隔(ms)
vi_error_cnt	Vi 通道错误状态计数

表 11-2

11.3. VO 模块

[调试信息]

#cat /tmp/mpp/vo

screen 0:

de_rate 432000000 hz, ref_fps:176

mgr0: 240x320 fmt[rgb] cs[0x0] range[limit] unblank direct_show[false]

lcd output backlight(197) fps:178.5 240x 320

err:1 skip:16 irq:147701 vsync:0

BUF enable ch[0] lyr[0] z[0] prem[N] a[pixel 128] fmt[77] fb[720,1280; 720,1280; 720,1280] crop[0, 0, 720,1280] frame[25, 0, 190, 320] addr[70a00000,70b00000, 0] flags[0x 0] trd[0,0]

BUF enable ch[1] lyr[0] z[16] prem[N] a[pixel 150] fmt[0] fb[240, 320; 240, 320; 240, 320] crop[0, 0, 240, 320] frame[0, 0, 240, 320] addr[70b80000, 0, 0] flags[0x 0] trd[0,0]

参数		描述
screen0		屏幕设备号
de_rate		DE 模块运行频率
ref_fps		输出设备的参考刷新频率
内核设备管理信息	mgr0	分辨率
	fmt	颜色格式
	cs	颜色空间，color sapce
	range	YUV 的 value 值范围，limit：0~255
	unblank	unblank：显示；blank：不显示
	direct_show	cvbs 测试指标标志位，为 true 时测试有可能通过，为 false 时测试肯定不通过
输出设备信息	lcd output	输出设备类型，有 lcd、hdmi、vga 等
	backlight	屏幕背光值
	fps	设备实际的刷新频率
	240x 320	设备分辨率
	err	DE 缺数的次数
	skip	DE 跳帧的次数
	irq	tcon 中断次数
	vsync	已发送的 vsync 的消息的个数
图层 buf 信息	enable	图层处于使能状态
	ch[0]	图层处于 blending 通道 0
	lyr[0]	该图层处于当前 blending 通道中的图层 0
	z[0]	显示优先级，不同图层的值唯一，越大显示越靠近上层
	prem[Y]	是否预乘格式，Y 是，N 否
	a[pixel 128]	alpha 参数，类型有 globl/pixel 两种；128 是 alpha 的取值
	fmt	图层格式，值 64 以下为 RGB 格式；以上为 YUV 格式，常见的 72 为 YV12, 76 为 NV12
	fb	图层 buffer 的 size，width,height，三个分量
	crop	裁剪区域
	frame	图层在屏幕上的显示区域

	addr	YUV 分量的地址，物理地址
	flag	一般为 0， 3D SS 时 0x4, 3D TB 时为 0x1, 3D FP 时为 0x2
	trd	是否 3D 输出，3D 输出的类型（HDMI FP 输出时为 1）

表 11-3

11.4. Video Encode 模块

[调试信息]

#cat /tmp/mpp/ve

**********begin venc channel[0]: h264 enc info**********

profile:Hp, level:51, bitRate:6291456, frameRate:30

inputWxH:1280x720, outputWxH:1280x720

idr_period:30, virtual_i_period:0

bitRate mode: CBR

slice:P, qp:35, frameNum:248, lt_ref:1

the average bitRate:597Kbits, real bitRate:575Kbits

the average frameRate:31fps, real FrameRate:31fps

AFBC:0, rot_angle:0, filter3d_level:0

smart:0, intra4x4:1, intraInPFrame:1

crop_left:0, crop_top:0, crop_width:0, crop_height:0

qp_offset1:10, qp_offset2:10, qp_offset3:10, qp_offset4:10

qp_offset5:10, qp_offset6:10, qp_offset7:10, qp_offset8:10

vbv_size:12582912Bytes,UnusedBufferSize:12581184Bytes,

UsedBufferSize:1728Bytes, ValidFrameNum:1

**********end channel[0]: h264 enc info**********

[调试信息分析]

记录 VE 模块当前的配置信息

参数		描述
Ve 通道基本参数	channel	通道号
	profile	编码等级，0：baseline，1：MP，2：

		HP，3：SVC-T
	level	编码 level 10：level1.0，11：level1.1， 12：level1.2，13：level1.3， 20：level2.0，21：level2.1， 22：level2.2，30：level3.0， 31：level3.1，32：level3.2， 40：level4.0，41：level4.1， 42：level4.2，50：level5.0， 51：level5.1
	bitRate	用户设置的编码码率
	frameRate	用户设置的编码帧率
	inputWxH	输入图像大小
	outputWxH	输出图像大小
	idr_period	IDR 帧间隔
	virtual_i_period	虚拟 i 帧间隔，0：不开启虚拟 i 帧功能； 其它值：用户设定的虚拟 i 帧间隔
	bitRate mode	码流控制模式，CBR：固定码率控制； VBR：可变码率控制
	vbv_size	设置的 vbv buffer 大小
	UnusedBufferSize	vbv buffer 剩余可用的大小
	UsedBufferSize	已经使用的 vbv buffer 大小
	ValidFrameNum	在 vbv buffer 中还没被取走的帧数
	the average bitRate	编码实际平均码流
	real bitRate	统计时间内的实际码率
	the average frameRate	编码实际平均帧率
	real FrameRate	统计时间内的实际帧率
当前帧参数	slice	当前帧 slice 类型，I、P、B
	qp	当前帧级 qp 值
	frameNum	已编码帧数量
	lt_ref	当前帧在图像序列中的索引
编码设置	smart	是否开启 smart 编码功能，

		0：关闭， 1：开启
	intra4x4	是否开启 I 帧的帧内 4x4 分类功能， 0：关闭， 1：开启
	intraInPFrame	是否开启 p 帧的帧内预测编码功能， 0：关闭， 1：开启
	AFBC	是否开启 AFBC 编码， 0：关闭， 1：开启
	rot_angle	图像旋转角度， 0：不旋转 90：旋转 90 度 180：旋转 180 度 270：旋转 270 度
	filter3d_level	3D 去噪等级， 0：不开启 3D 去噪 1~6：不同的 3D 去噪等级
	crop_left 、 crop_top 、 crop_width、crop_height	Crop 图像的起始点（left,top）和 crop 图像的宽高。
	qp_offset1~qp_offse8	8 个 ROI 区域的 QP 相对偏移值。

表 11-4

11.5. Video Decode 模块

[调试信息]

#cat /tmp/mpp/ve

**********channel[0]: h264 dec info**********

profile: main, widht: 720, height: 576

RefCount: 1, FrameRate: 25fps

[调试信息分析]

记录 VE 模块当前的配置信息

参数	描述
profile	解码等级
widht	解码宽度
height	解码高度
RefCount	解码当前帧的参考帧个数
FrameRate	实际帧率

表 11-5

11.6. Audio AIO 模块

注意：需要手动加载 AIO 模块才可以看到 AIO 节点，insmod sunxi_aio.ko。

#cat /tmp/mpp/sunxi-aio

----------AI Dev ATTR(V0.1)----------

ai_enable: 0

ai_chncnt: 0

ai_cardtype: 0(0-codec;1-linein)

ai_samplerate: 0

ai_bitwidth: 0

ai_trackcnt: 0

ai_bMute: 0

ai_volume: 0

----------AO Dev ATTR(V0.1)----------

ao_enable: 0

ao_chncnt: 0

ao_cardtype: 0(0-codec;1-hdmi)

ao_samplerate: 0

ao_bitwidth: 0

ao_trackcnt: 0

ao_bMute: 0

ao_volume: 0

参数		描述	参数范围
AudioInput PARAM 参数配置	ai_enable	AI 设备使能标志	0：未使能。1：已使能
	ai_chncnt	AI 通道个数	一共 16 个通道
	ai_cardtype	AI 的 Card 类型	0-codec;1-linein
	ai_samplerate	AI 采样率	8~192kHz
	ai_bitwidth	AI 位宽	16、24、32bit
	ai_trackcnt	音轨	1:Mono,2:Stereo
	ai_bMute	静音标志	0：不静音。1：静音

	ai_volume:	声音大小	0~100
AudioOuput PARAM 参数配置	ao_enable	AO 设备使能标志	0：未使能。1：已使能
	ao_chncnt	AO 通道个数	一共 16 个通道
	ao_cardtype	AO 的 Card 类型	0-codec;1-hdmi
	ao_samplerate	AO 采样率	8~192kHz
	ao_bitwidth	AO 位宽	16、24、32bit
	ao_trackcnt	音轨	1:Mono,2:Stereo
	ao_bMute	静音标志	0：不静音。1：静音
	ao_volume:	声音大小	0~100

表 11-6

12. Declaration

This document is the original work and copyrighted property of Allwinner Technology (“Allwinner”). Reproduction in whole or in part must obtain the written approval of Allwinner and give clear acknowledgement to the copyright owner.

The information furnished by Allwinner is believed to be accurate and reliable. Allwinner reserves the right to make changes in circuit design and/or specifications at any time without notice. Allwinner does not assume any responsibility and liability for its use. Nor for any infringements of patents or other rights of the third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Allwinner. This datasheet neither states nor implies warranty of any kind, including fitness for any particular application.