Hi3507 媒体处理软件

开发参考

文档版本　01

发布日期　2011-07-12

商标声明

 、HISILICON、海思和其他海思商标均为深圳市海思半导体有限公司的商标。

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、服务或特性等应受海思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，海思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。

深圳市海思半导体有限公司

地址：　深圳市龙岗区坂田华为基地华为电气生产中心　邮编：518129

网址：　http://www.hisilicon.com

客户服务电话：　+86-755-28788858

客户服务传真：　+86-755-28357515

客户服务邮箱：　support@hisilicon.com

前 言

概述

本文为使用 Hi3507 媒体处理芯片进行开发的程序员而写，目的是供您在开发过程中查阅媒体处理软件开发包的各种参考信息，包括 API、头文件、错误码等。

本文档描述了 Hi3507 媒体处理软件的各个 API 的使用方法，以及相关的数据结构和错误码。

产品版本

与本文档相对应的产品版本如下。

产品名称	产品版本
Hi3507 芯片	V100

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

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

符号	说明
危险	表示有高度潜在危险，如果不能避免，会导致人员死亡或严重伤害。

符号	说明
 警告	表示有中度或低度潜在危险，如果不能避免，可能导致人员轻微或中等伤害。
 注意	表示有潜在风险，如果忽视这些文本，可能导致设备损坏、数据丢失、设备性能降低或不可预知的结果。
窍门	表示能帮助您解决某个问题或节省您的时间。
说明	表示是正文的附加信息，是对正文的强调和补充。

数值单位约定

数据容量、频率、数据速率等的表达方式说明如下。

类别	符号	对应的数值
数据容量（如 RAM 容量）	1K	1024
	1M	1,048,576
	1G	1,073,741,824
频率、数据速率等	1k	1000
	1M	1,000,000
	1G	1,000,000,000

地址、数据的表达方式说明如下。

符号	举例	说明
0x	0xFE04、0x18	用 16 进制表示的数据值、地址值。
0b	0b000、0b00 00000000	表示 2 进制的数据值以及 2 进制序列（寄存器描述中除外）。
X	00X、1XX	在数据的表达方式中，X 表示 0 或 1。 例如：00X 表示 000 或 001； 1XX 表示 100、101、110 或 111。

修订记录

修订记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

文档版本 01 (2011-06-16)

第 1 次正式发布。

目 录

表格目录

1 概述

1.1 API 函数参考域

本文档对 API 参考信息使用以下域来描述，具体信息如表 1-1 所示。

表1-1 API 函数参考域说明

参考域	作用
描述	描述 API 的功能。
语法	显示 API 的语法样式。
参数	列出 API 的参数、参数说明及其属性。
返回值	列出 API 的返回值及其返回值说明。
错误码	列出 API 的错误码及其错误码说明。
需求	列出本 API 要包含的头文件和 API 编译时要链接的库文件。
注意	使用 API 时应注意的事项。
举例	使用 API 的实例。
相关主题	同本 API 的其他相关信息。

1.2 数据类型参考域

本参考对数据类型使用以下域来描述，具体信息如表 1-2 所示。

表1-2 数据类型参考域说明

参考域	作用
说明	简单描述结构体所实现的功能。

参考域	作用
定义	列出结构体的定义。
成员	列出数据结构的成员及含义。
注意事项	列出使用数据类型时应注意的事项。
相关数据类型和接口	列出与本数据类型相关联的其他数据类型和接口。

2 API 参考

2.1 概述

Hi3507 媒体处理软件开发包的 API（以下简称 MPI）分为视频功能类、音频功能类、音视频复合编码功能类。

2.2 视频功能函数参考

2.2.1 系统控制

系统控制实现 MPP（Media Process Platform）系统初始化、获取 MPP 版本号、视频缓存池初始化、创建视频缓存池等功能（视频缓存池工作原理介绍请参见《Hi3507 媒体处理软件开发指南》的 2.1.3 节）。

该功能模块提供以下 MPI：

- HI_MPI_SYS_SetConf：配置系统控制参数。
- HI_MPI_SYS_GetConf：获取系统控制参数。
- HI_MPI_SYS_Init：初始化 MPP 系统。
- HI_MPI_SYS_Exit：去初始化 MPP 系统。
- HI_MPI_SYS_GetVersion：获取 MPP 的版本号。
- HI_MPI_VB_SetConf：设置 MPP 视频缓存池属性。
- HI_MPI_VB_GetConf：获取 MPP 视频缓存池属性。
- HI_MPI_VB_Init：初始化 MPP 视频缓存池。
- HI_MPI_VB_Exit：去初始化 MPP 视频缓存池。
- HI_MPI_VB_CreatePool：创建一个视频缓存池。
- HI_MPI_VB_DestroyPool：销毁一个视频缓存池。
- HI_MPI_VB_GetBlock：获取一个缓存块。
- HI_MPI_VB_ReleaseBlock：释放一个已经获取的缓存块。
- HI_MPI_VB_Handle2PhysAddr：获取一个缓存块的物理地址。
- HI_MPI_VB_Handle2PoolId：获取一个帧缓存块所在缓存池的 ID。

HI_MPI_SYS_SetConf

【描述】

配置系统控制参数。

【语法】

```
HI_S32 HI_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，或者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，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_SYS_NULL_PTR	空指针错误。
HI_ERR_SYS_NOT_PERM	操作不允许。此错误通常是在系统已经初始化后，调用此接口导致。
HI_ERR_SYS_ILLEGAL_PARAM	输入参数非法。此错误通常是系统控制参数中有非法参数导致。

【需求】

- 头文件：hi_comm_sys.h、mpi_sys.h
- 库文件：libmpi.a

【注意】

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。video buf 根据不同的应用场景需要不同的配置，关于 video buf 和场景相关配置的建议，请参见《Hi3507 媒体处理软件 开发指南》。

【举例】

```
HI_S32 s32ret;
MPP_SYS_CONF_S struSysConf;

struSysConf.u32AlignWidth = 64;

/* set config of mpp system*/
s32ret = HI_MPI_SYS_SetConf(&struSysConf);
if (HI_SUCCESS != s32ret)
{
    printf("Set mpp sys config failed!\n");
    return s32ret;
}

/* init 3507 system*/
s32ret = HI_MPI_SYS_Init();
if (HI_SUCCESS != s32ret)
{
    printf("Mpi init failed!\n");
    return s32ret;
}

/* ..... */

/* exit 3507 system*/
s32ret = HI_MPI_SYS_Exit();
if (HI_SUCCESS != s32ret)
{
    printf("Mpi exit failed!\n");
    return s32ret;
}
```

【相关主题】

HI_MPI_VB_GetConf

HI_MPI_SYS_GetConf

【描述】

获取系统控制参数。

【语法】

```
HI_S32 HI_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，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_SYS_NULL_PTR	空指针错误。
HI_ERR_SYS_NOT_PERM	操作不允许。此错误通常是由于未配置过系统控制参数导致。

【需求】

- 头文件：hi_comm_sys.h、mpi_sys.h
- 库文件：libmpi.a

【注意】

必须先调用 HI_MPI_SYS_SetConf 成功后才能获取配置。

【举例】

无。

【相关主题】

HI_MPI_SYS_SetConf

HI_MPI_SYS_Init

【描述】

初始化 MPP 系统。

【语法】

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

【参数】

无。

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_SYS_NOTREADY	系统参数未配置。
HI_ERR_SYS_BUSY	系统忙，通常是由于系统正在去初始化导致。
HI_FAILURE	MPP 系统中有模块初始化失败，通常是由于未加载 hidmac.ko 导致。

【需求】

- 头文件：hi_comm_sys.h、mpi_sys.h
- 库文件：libmpi.a

【注意】

- 必须先调用 HI_MPI_SYS_SetConf 配置 MPP 系统后才能初始化，否则初始化会失败。
- 由于 MPP 系统的正常运行依赖于缓存池，因此必须先调用 HI_MPI_VB_Init 初始化缓存池，再初始化 MPP 系统。
- 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。
- 只要有一个进程进行初始化即可，不需要所有的进程都做系统初始化的操作。

【举例】

请参见 HI_MPI_SYS_SetConf 的举例。

【相关主题】

HI_MPI_SYS_Exit

HI_MPI_SYS_Exit

【描述】

去初始化 MPP 系统。除了音频的编解码通道和复合流通道外，所有的音频输入输出、视频输入输出、视频编码、视频解码通道都会被销毁或者禁用，整个 MPP 系统恢复到系统未初始化的状态。

【语法】

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

【参数】

无。

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_SYS_BUSY	系统忙，通常是由于还有用户进程阻塞 MPI 接口导致。

【需求】

- 头文件：hi_comm_sys.h、mpi_sys.h
- 库文件：libmpi.a

【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，则可以去初始化成功。
- 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和复合流通道，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了，则这些通道随之被销毁。

【举例】

请参见 HI_MPI_SYS_SetConf 的举例。

【相关主题】

HI_MPI_SYS_Init

HI_MPI_SYS_GetVersion

【描述】

获取 MPP 的版本号。

【语法】

```
HI_S32 HI_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）。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_SYS_NULL_PTR	空指针错误。

【需求】

- 头文件：hi_comm_vb.h、mpi_vb.h
- 库文件：libmpi.a

【注意】

无。

【举例】

```
HI_S32 s32ret;
MPP_VERSION_S stVersion;

s32ret = HI_MPI_SYS_GetVersion(&stVersion);
```

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

【相关主题】

无。

HI_MPI_VB_SetConf

【描述】

设置 MPP 视频缓存池属性。

【语法】

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

【参数】

参数名称	描述	输入/输出
pstVbConf	视频缓存池属性指针。 静态属性。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VB_NULL_PTR	空指针错误。
HI_ERR_VB_BUSY	系统忙，通常是已经调用 HI_MPI_SYS_Init 对系统进行了初始化导致。

【需求】

- 头文件：hi_comm_vb.h、mpi_vb.h
- 库文件：libmpi.a

【注意】

只能在系统处于未初始化的状态下，才可以设置缓存池属性，否则会返回失败。不同的业务，缓存池的属性配置不同，具体请参见《Hi3507 媒体处理软件　开发指南》的2.1.4 节。

【举例】

```
HI_S32 s32ret;
VB_CONF_S stVbConf;

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

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

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

/* ... ... */

(void)HI_MPI_VB_Exit();
```

【相关主题】

HI_MPI_VB_GetConf

HI_MPI_VB_GetConf

【描述】

获取 MPP 视频缓存池属性。

【语法】

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

【参数】

参数名称	描述	输入/输出
pstVbConf	视频缓存池属性指针。 静态属性。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VB_NULL_PTR	空指针错误。
HI_ERR_VB_NOTREADY	缓存池属性未配置。

【需求】

- 头文件：hi_comm_vb.h、mpi_vb.h
- 库文件：libmpi.a

【注意】

必须先调用 HI_MPI_VB_SetConf 设置 MPP 视频缓存池属性，再获取属性。

【举例】

无。

【相关主题】

HI_MPI_VB_SetConf

HI_MPI_VB_Init

【描述】

初始化 MPP 视频缓存池。

【语法】

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

【参数】

无。

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VB_NOTREADY	缓存池属性未配置。
HI_FAILURE	失败，通常是保留内存不够导致。

【需求】

- 头文件：hi_comm_vb.h、mpi_vb.h
- 库文件：libmpi.a

【注意】

- 必须先调用 HI_MPI_VB_SetConf 配置缓存池属性，再初始化缓存池，否则会失败。
- 可反复初始化，不返回失败。

【举例】

请参见 HI_MPI_VB_SetConf 的举例。

【相关主题】

HI_MPI_VB_Exit

HI_MPI_VB_Exit

【描述】

去初始化 MPP 视频缓存池。

【语法】

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

【参数】

无。

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VB_BUSY	系统忙，通常是由于 MPP 系统未去初始化导致。

【需求】

- 头文件：hi_comm_vb.h、mpi_vb.h
- 库文件：libmpi.a

【注意】

- 必须先调用 HI_MPI_SYS_Exit 去初始化 MPP 系统，再去初始化缓存池，否则返回失败。
- 可以反复去初始化，不返回失败。
- 去初始化不会清除先前对缓存池的配置。

【举例】

请参见 HI_MPI_VB_SetConf 的举例。

【相关主题】

HI_MPI_VB_Init

HI_MPI_VB_CreatePool

【描述】

创建一个视频缓存池。

【语法】

```
VB_POOL HI_MPI_VB_CreatePool(HI_U32 u32BlkSize,HI_U32 u32BlkCnt);
```

【参数】

参数名称	描述	输入/输出
u32BlkSize	缓存池中每个缓存块的大小。 取值范围：（0, 2^{32}），以 byte 为单位。	输入
u32BlkCnt	缓存池中缓存块的个数。 取值范围：（0, 2^{32}）。	输入

【返回值】

返回值	描述
非 VB_INVALID_POOLID	有效的缓存池 ID 号。
VB_INVALID_POOLID	创建缓存池失败，可能是参数非法或者保留内存不够。

【错误码】

无。

【需求】

- 头文件：hi_comm_vb.h、mpi_vb.h
- 库文件：libmpi.a

【注意】

该缓存池是从保留内存中分配的，一个缓存池包含若干个大小相同的缓存块。如果该缓存池的大小超过了保留内存中的空闲空间，则创建缓存池会失败。

【举例】

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

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

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

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

/* use this address do something ...*/

/* then release the buffer block*/
(void)HI_MPI_VB_ReleaseBlock(VbBlk);

/* destroy video buffer pool */
(void)HI_MPI_VB_DestroyPool(VbPool);
```

【相关主题】

HI_MPI_VB_DestroyPool

HI_MPI_VB_DestroyPool

【描述】

销毁一个视频缓存池。

【语法】

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

【参数】

参数名称	描述	输入/输出
Pool	缓存池 ID 号。 取值范围：[0, VB_MAX_POOLS)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VB_NOTREADY	缓存池未初始化。
HI_ERR_VB_ILLEGAL_PARAM	输入参数无效，缓存池 ID 非法。
HI_ERR_VB_UNEXIST	缓存池不存在。
HI_ERR_VB_NOT_PERM	操作不允许，试图销毁一个非用户态创建的缓存池。

【需求】

- 头文件：hi_comm_vb.h、mpi_vb.h
- 库文件：libmpi.a

【注意】

- 销毁一个不存在的缓存池，则返回 HI_ERR_VB_UNEXIST。
- 在去初始化 MPP 缓存池时，所有的缓存池都将被销毁，包括用户态的缓存池。

【举例】

请参见 HI_MPI_VB_CreatePool 的举例。

【相关主题】

HI_MPI_VB_CreatePool

HI_MPI_VB_GetBlock

【描述】

用户态获取一个缓存块。

【语法】

```
VB_BLK HI_MPI_VB_GetBlock(VB_POOL Pool, HI_U32 u32BlkSize);
```

【参数】

参数名称	描述	输入/输出
Pool	缓存池 ID 号。 取值范围：[0, VB_MAX_POOLS)。	输入
u32BlkSize	缓存块大小。 取值范围：(0, 2^{32})，以 byte 为单位。	输入

【返回值】

返回值	描述
非 VB_INVALID_HANDLE	有效的缓存块句柄。
VB_INVALID_HANDLE	获取缓存块失败。

【错误码】

无。

【需求】

- 头文件：hi_comm_vb.h、mpi_vb.h
- 库文件：libmpi.a

【注意】

- 用户可以在创建一个缓存池之后，调用本接口从该缓存池中获取一个缓存块；即将第 1 个参数 Pool 设置为创建的缓存池 ID（第 2 个参数 u32BlkSize 不用设置）；获取的缓存块的大小即为用户创建缓存池时指定的缓存块大小。
- 如果用户需要从任意一个公共缓存池中获取一块指定大小的缓存块，则可以将第 1 个参数 Pool 设置为无效 ID 号（VB_INVALID_POOLID），将第 2 个参数 u32BlkSize 设置为需要的缓存块大小。
- 公共缓存池主要用来存放 VIU 的捕获图像，因此，对公共缓存池的不当操作（如占用过多的缓存块）会影响 MPP 系统的正常运行。

【举例】

请参见 HI_MPI_VB_CreatePool 的举例。

【相关主题】

HI_MPI_VB_ReleaseBlock

HI_MPI_VB_ReleaseBlock

【描述】

用户态释放一个已经获取的缓存块。

【语法】

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

【参数】

参数名称	描述	输入/输出
Block	缓存块句柄。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VB_ILLEGAL_PARAM	输入参数无效，无效的缓存块句柄。

【需求】

- 头文件：hi_comm_vb.h、mpi_vb.h
- 库文件：libmpi.a

【注意】

获取的缓存块使用完后，应该调用此接口释放缓存块。

【举例】

请参见 HI_MPI_VB_CreatePool 的举例。

【相关主题】

HI_MPI_VB_GetBlock

HI_MPI_VB_Handle2PhysAddr

【描述】

用户态获取一个缓存块的物理地址。

【语法】

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

【参数】

参数名称	描述	输入/输出
Block	缓存块句柄。	输入

【返回值】

返回值	描述
0	无效返回值，缓存块句柄非法。
非 0	有效物理地址。

【错误码】

无。

【需求】

- 头文件：hi_comm_vb.h、mpi_vb.h
- 库文件：libmpi.a

【注意】

指定的缓存块应该是从 MPP 视频缓存池中获取的有效缓存块。

【举例】

请参见 HI_MPI_VB_CreatePool 的举例。

【相关主题】

无。

HI_MPI_VB_Handle2PoolId

【描述】

用户态获取一个帧缓存块所在缓存池的 ID。

【语法】

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

【参数】

参数名称	描述	输入/输出
Block	缓存块句柄。	输入

【返回值】

返回值	描述
非负数	有效的缓存池 ID 号。
负数	无效的缓存池 ID 号。

【错误码】

无。

【需求】

- 头文件：hi_comm_vb.h、mpi_vb.h
- 库文件：libmpi.a

【注意】

指定的缓存块应该是从 MPP 视频缓存池中获取的有效缓存块。

【举例】

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

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

【相关主题】

无。

2.2.2 视频输入

视频输入（VI）实现启用视频输入设备、视频输入通道、绑定视频输入通道到某个视频输出通道等功能。

该功能模块提供以下 MPI：

- HI_MPI_VI_SetPubAttr：设置 VI 设备属性。

- HI_MPI_VI_GetPubAttr：获取 VI 设备属性。
- HI_MPI_VI_Enable：启用 VI 设备。
- HI_MPI_VI_Disable：禁用 VI 设备。
- HI_MPI_VI_SetChnAttr：设置 VI 通道属性。
- HI_MPI_VI_GetChnAttr：获取 VI 通道属性。
- HI_MPI_VI_EnableChn：启用 VI 通道。
- HI_MPI_VI_DisableChn：禁用 VI 通道。
- HI_MPI_VI_GetChnLuma：获取 VI 通道图像亮度。
- HI_MPI_VI_GetFrame：获取 VI 原始帧图像。
- HI_MPI_VI_ReleaseFrame：释放原始图像数据所占的缓存。
- HI_MPI_VI_BindOutput：绑定 VI、VO 通道。
- HI_MPI_VI_UnBindOutput：解绑定 VI、VO 通道。
- HI_MPI_VI_SetSrcFrameRate：设置视频输入通道的原始帧率。
- HI_MPI_VI_GetSrcFrameRate：获取视频输入通道的目标帧率。
- HI_MPI_VI_SetFrameRate：设置视频输入通道的目标帧率。
- HI_MPI_VI_GetFrameRate：获取视频输入通道的目标帧率。
- HI_MPI_VI_SetUserPic：设置用户图片的帧信息。
- HI_MPI_VI_EnableUserPic：启用插入用户图片。
- HI_MPI_VI_DisableUserPic：禁用插入用户图片。
- HI_MPI_VI_SetMinorChnAttr：设置 VI 通道次属性。
- HI_MPI_VI_GetMinorChnAttr：获取 VI 通道次属性。
- HI_MPI_VI_GetFd：获取 VI 通道的设备文件句柄。

HI_MPI_VI_SetPubAttr

【描述】

设置 VI 设备属性。

【语法】

```
HI_S32 HI_MPI_VI_SetPubAttr(VI_DEV ViDevId,const VI_PUB_ATTR_S
*pstPubAttr);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
pstPubAttr	VI 设备属性指针。 静态属性。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_PARA	视频输入参数无效。
HI_ERR_VI_INVALID_NULL_PTR	空指针错误。
HI_ERR_VI_FAILED_NOTDISABLE	视频输入设备未禁用。
HI_ERR_VI_NOT_SUPPORT	操作不支持，当前版本不支持此配置。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

- 在调用前要保证 VI 设备处于禁用状态。如果 VI 设备已处于使能状态，可以使用 HI_MPI_VI_Disable 来禁用设备。
- 整体捕获区域的配置目前无效。
- 当数据接收模式为 BT.656 时，需要设置工作模式和视频输入制式，工作模式支持 1D1 模式，VI 的工作模式必须和 ADC 的工作模式一致才能正常捕获视频数据。1D1 工作模式时，每个 VI 设备下支持 1 个 VI 通道；对应 ADC 的 27M 单路 D1 模式。
- 当数据接收模式为 BT.601 时，只需要设置视频制式，公共属性结构体中的其他项不用设置；只能使用 VI 设备 0，且只支持 1 个通道。
- 当数据接收模式为 digital camera 时，公共属性结构体中的其他所有项都不用设置；只能使用 VI 设备 0，且只支持 1 个通道。
- 当数据接收模式为 720P 高清模式时，公共属性结构体中的其他所有项都不用设置；只能使用 VI 设备 0，且仅只支持 1 个通道。

【举例】

```
HI_S32 s32ret;
```

```
VI_DEV ViDevId = 0;
VI_CHN ViChn = 0;
VO_CHN VoChn = 0;
VI_PUB_ATTR_S stPubAttr;
VI_CHN_ATTR_S stChnAttr;

stPubAttr.u32AdType = AD_2815;
stPubAttr.enInputMode = VI_MODE_BT656;
stPubAttr.enViNorm = VIDEO_ENCODING_MODE_PAL;
stPubAttr.enWorkMode = VI_WORK_MODE_2D1;
/* set public attribute of vi */
s32ret = HI_MPI_VI_SetPubAttr(ViDevId, &stPubAttr);
if (HI_SUCCESS != s32ret)
{
    printf("set vi pub attr err:0x%x\n", s32ret);
    return s32ret;
}

/* enable vi device*/
s32ret = HI_MPI_VI_Enable(ViDevId);
if (HI_SUCCESS != s32ret)
{
    printf("enable vi dev err:0x%x\n", s32ret);
    return s32ret;
}

stChnAttr.enCapSel = VI_CAPSEL_BOTH;
stChnAttr.bDownScale = HI_FALSE;
stChnAttr.stCapRect.s32X = 0;
stChnAttr.stCapRect.s32Y = 0;
stChnAttr.stCapRect.u32Width = 704;
stChnAttr.stCapRect.u32Height = 288;
stChnAttr.enViPixFormat = PIXEL_FORMAT_YUV_SEMIPLANAR_420;
stChnAttr.bChromaResample = HI_FALSE;
stChnAttr.bHighPri = HI_FALSE;
/* set channel attribute for vi chn */
s32ret = HI_MPI_VI_SetChnAttr(ViDevId, ViChn, &stChnAttr);
if (HI_SUCCESS != s32ret)
{
    printf("set vi chn attr err:0x%x\n", s32ret);
    return s32ret;
}

/* enable vi chn */
```

```
s32ret = HI_MPI_VI_EnableChn(ViDevId, ViChn);
if (HI_SUCCESS != s32ret)
{
    printf("enable vi chn err:0x%x\n", s32ret);
    return s32ret;
}

/* bind vi to vo */
s32ret = HI_MPI_VI_BindOutput(ViDevId, ViChn, VoChn);
if (HI_SUCCESS != s32ret)
{
    printf("bind vi to vo err:0x%x\n", s32ret);
    return s32ret;
}

/* ... ... ... */

/* unbind vi to vo */
s32ret = HI_MPI_VI_UnBindOutput(ViDevId, ViChn, VoChn);
if (HI_SUCCESS != s32ret)
{
    printf("unbind vi to vo err:0x%x\n", s32ret);
    return s32ret;
}

/* disable vi chn */
s32ret = HI_MPI_VI_DisableChn(ViDevId, ViChn);
if (HI_SUCCESS != s32ret)
{
    printf("disale vi chn err:0x%x\n", s32ret);
    return s32ret;
}
/* disable vi device*/
s32ret = HI_MPI_VI_Disable(ViDevId);
if (HI_SUCCESS != s32ret)
{
    printf("disale vi dev err:0x%x\n", s32ret);
    return s32ret;
}
```

【相关主题】

HI_MPI_VI_GetPubAttr

HI_MPI_VI_GetPubAttr

【描述】

获取 VI 设备属性。

【语法】

```
HI_S32 HI_MPI_VI_GetPubAttr(VI_DEV ViDevId,VI_PUB_ATTR_S *pstPubAttr);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
pstPubAttr	VI 设备属性指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_FAILED_NOTCONFIG	视频输入设备属性未设置。
HI_ERR_VI_INVALID_NULL_PTR	空指针错误。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

如果未设置 VI 设备属性，该接口将返回失败。

【举例】

```
HI_S32 s32ret;
VI_DEV ViDevId = 0;
VI_PUB_ATTR_S stPubAttr;

/* first set public attribute of vi and enable it*/

s32ret = HI_MPI_VI_GetPubAttr(ViDevId, &stPubAttr);
if (HI_SUCCESS != s32ret)
{
    printf("set vi pub attr err:0x%x\n", s32ret);
    return s32ret;
}
```

【相关主题】

HI_MPI_VI_SetPubAttr

HI_MPI_VI_Enable

【描述】

启用 VI 设备。

【语法】

```
HI_S32 HI_MPI_VI_Enable(VI_DEV ViDevId);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。

接口返回值	含义
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_FAILED_NOTCONFIG	视频输入设备属性未设置。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

- 启用前必须已经设置设备属性，否则返回失败。
- 可重复启用，不返回失败。

【举例】

请参见 HI_MPI_VI_SetPubAttr 的举例。

【相关主题】

HI_MPI_VI_Disable

HI_MPI_VI_Disable

【描述】

禁用 VI 设备。

【语法】

```
HI_S32 HI_MPI_VI_Disable(VI_DEV ViDevId);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_FAILED_CHNOTDISABLE	视频输入通道未禁用。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

- 必须先禁用所有 VI 通道后，再禁用 VI 设备。
- 可重复禁用，不返回失败。

【举例】

请参见 HI_MPI_VI_SetPubAttr 的举例。

【相关主题】

HI_MPI_VI_Enable

HI_MPI_VI_SetChnAttr

【描述】

设置 VI 通道属性。

【语法】

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

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入
pstAttr	VI 通道属性指针。 动态属性。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入通道号无效。
HI_ERR_VI_INVALID_PARA	视频输入参数无效。
HI_ERR_VI_INVALID_NULL_PTR	空指针错误。
HI_ERR_VI_FAILED_NOTCONFIG	视频输入设备或通道的属性未配置。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

- 必须先设置 VI 设备属性，再设置通道属性。
- VI 通道属性可以动态设置，即在 VI 通道启用或禁用时都可以调用此接口。
- 视频输入接口模式为 BT601 或 BT656 时，配置捕获区域时，可以指定是否水平 1/2 压缩、捕获一场或两场，捕获区域的宽度 u32Width 为水平压缩前宽度，捕获区域的高度 u32Height 为一场的捕获高度。
- 视频输入接口模式为 digital camera 时，配置捕获区域时，可以指定是否水平 1/2 压缩，帧场选择必须选择两场捕获，捕获区域的宽度 u32Width 为水平压缩前宽度，捕获区域的高度 u32Height 为一帧图像的捕获高度。
- 视频输入接口模式为 720P 高清模式，配置捕获区域时，水平压缩无效，帧场选择必须选择两场捕获，捕获区域宽高即为一帧图像的捕获宽高。
- 像素格式只支持 PIXEL_FORMAT_YUV_SEMIPLANAR_422 和 PIXEL_FORMAT_YUV_SEMIPLANAR_420。
- 最多只能有一个通道设为高优先级。例如：可以将 D1 的那个 VI 通道设置为高优先级，以保证其实时性。重复设置优先级时，以最后一次的设置为准。

- 色度重采样 bChromaResample，如无特殊要求一般配置为 FALSE 即可。
- 用户需自己保证配置的区域大小与实际的 ADC 输出图像大小相匹配，否则可能出现图像被裁剪或显示无效图像的现象。

【举例】

请参见 HI_MPI_VI_SetPubAttr 的举例。

【相关主题】

HI_MPI_VI_GetChnAttr

HI_MPI_VI_GetChnAttr

【描述】

获取 VI 通道属性。

【语法】

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

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入
pstAttr	VI 通道属性指针。 动态属性。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。

接口返回值	含义
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入通道号无效。
HI_ERR_VI_INVALID_NULL_PTR	空指针错误。
HI_ERR_VI_FAILED_NOTCONFIG	视频输入设备属性未配置。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

必须先设置通道属性再获取属性，否则将返回 HI_ERR_VI_FAILED_NOTCONFIG。

【举例】

```
HI_S32 s32ret;
VI_DEV ViDevId = 0;
VI_CHN ViChn = 0;
VI_CHN_ATTR_S stChnAttr;

/* first enable vi device and vi chn */

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

【相关主题】

HI_MPI_VI_SetChnAttr

HI_MPI_VI_EnableChn

【描述】

启用 VI 通道。

【语法】

```
HI_S32 HI_MPI_VI_EnableChn(VI_DEV ViDevId,VI_CHN ViChn);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM))。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入通道号无效。
HI_ERR_VI_FAILED_NOTENABLE	视频输入设备或通道未启用。
HI_ERR_VI_FAILED_NOTCONFIG	视频输入通道属性未设置。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

- 启用 VI 通道前，必须已经启用该通道所属 VI 设备；启用 VI 通道后，该 VI 通道开始捕获视频数据。
- 启用 VI 通道前，必须已经设置通道属性，否则返回失败。
- 可重复启用 VI 通道，不返回失败。

【举例】

请参见 HI_MPI_VI_SetPubAttr 的举例。

【相关主题】

HI_MPI_VI_Disable

HI_MPI_VI_DisableChn

【描述】

禁用 VI 通道。

【语法】

```
HI_S32 HI_MPI_VI_DisableChn(VI_DEV ViDevId,VI_CHN ViChn);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入通道号无效。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

可重复禁用 VI 通道，不返回失败。

【举例】

请参见 HI_MPI_VI_SetPubAttr 的举例。

【相关主题】

HI_MPI_VI_EnableChn

HI_MPI_VI_GetChnLuma

【描述】

获取 VI 通道图像的亮度接口。

【语法】

```
HI_S32 HI_MPI_VI_GetChnLuma(VI_DEV ViDevId,VI_CHN ViChn,VI_CH_LUM_S
*pstLuma);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入
pstLuma	VI 通道亮度信息指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入通道号无效。
HI_ERR_VI_INVALID_NULL_PTR	空指针错误。

接口返回值	含义
HI_ERR_VI_FAILED_NOTENABLE	视频输入设备或通道未启用。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

- 该接口需在通道已启用后才有效。
- 此接口获取的亮度值是 VI 原始捕获图像的所有像素亮度累加值。
- 通道启用后立即调用此接口，出现亮度值为 0 属正常情况，因为通道启用开始几个中断的时间硬件尚未开始捕获图像，如果是 VI 低帧率也有类似情况。

【举例】

```
HI_S32 s32ret;
VI_DEV ViDevId = 0;
VI_CHN ViChn = 0;
VI_CH_LUM_S stLuma;
s32ret = HI_MPI_VI_GetChnLuma(ViDevId, ViChn, &stLuma);
if (HI_SUCCESS != s32ret)
{
    printf("get vi luma err:0x%x\n", s32ret);
    return s32ret;
}
```

【相关主题】

无。

HI_MPI_VI_GetFrame

【描述】

获取原始图像数据。

【语法】

```
HI_S32 HI_MPI_VI_GetFrame(VI_DEV ViDevId,VI_CHN ViChn,VIDEO_FRAME_INFO_S
*pstFrame);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入
pstFrame	视频图像帧信息结构指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入通道号无效。
HI_ERR_VI_INVALID_NULL_PTR	空指针错误。
HI_ERR_VI_FAILED_NOTENABLE	视频输入设备或通道未启用。
HI_ERR_VI_BUF_EMPTY	视频输入缓存为空。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

- 此接口需在通道已启用后才有效。
- 此接口获取的 VI 视频帧为正在采集的实时图像，VI 软件内部没有缓存，如果用此接口获取连续的图像帧，可能会由于用户态处理的不及时性，导致获取的图像不连续（有丢帧）；因此在正常的预览和编码流程中，推荐使用 VI 绑定 VO 以及 VI 绑定 VENC 的操作，内核态下 VI 将实时图像数据直接发送到 VO 或 VENC 通道。

- pstFrame->stVFrame->u32PhyAddr[0]和 pstFrame->stVFrame->u32PhyAddr[1]分别指向图像的亮度分量和色度分量的物理地址。
- pstFrame 结构中图像虚拟地址无效。

【举例】

```
HI_S32 s32ret;
VI_DEV ViDevId = 0;
VI_CHN ViChn = 0;
VIDEO_FRAME_INFO_S stFrame;

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

/* deal with video frame ... */

/* release video frame */
(void)HI_MPI_VI_ReleaseFrame(ViDevId, ViChn, &stFrame);
```

【相关主题】

HI_MPI_VI_ReleaseFrame

HI_MPI_VI_ReleaseFrame

【描述】

释放原始图像数据所占的缓存。

【语法】

```
HI_S32 HI_MPI_VI_ReleaseFrame(VI_DEV ViDevId,VI_CHN ViChn,const
VIDEO_FRAME_INFO_S *pstRawFrame);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入
pstFrame	视频图像帧数据存储结构指针。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入通道号无效。
HI_ERR_VI_INVALID_NULL_PTR	空指针错误。
HI_ERR_VI_INVALID_PARA	输入参数无效。
HI_ERR_VI_FAILED_NOTENABLE	视频输入设备或通道未启用。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

- 该接口需在通道已启用后才有效。
- 要求用户每次使用完获取原始图像接口后，调用本接口释放原始图像数据（即必须与 HI_MPI_VI_GetFrame 配对使用）。

【举例】

请参见 HI_MPI_VI_GetFrame 的举例。

【相关主题】

HI_MPI_VI_GetFrame

HI_MPI_VI_BindOutput

【描述】

视频输入通道绑定到视频输出通道。

【语法】

```
HI_S32 HI_MPI_VI_BindOutput(VI_DEV ViDevId,VI_CHN ViChn,VO_CHN VoChn);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入
VoChn	VO 通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入或输出通道号无效。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

- 绑定后将在 VO 直接显示 VI 通道捕获的图像，不需作用户态获取和释放的操作即可实现预览的功能。
- 支持一个 VI 通道绑定到多个 VO 通道，但一个 VO 通道同时只能被一个 VI 通道绑定。

- 可以不解绑定，而直接绑定到另外一个 VO 通道。
- 可以重复绑定，不返回失败。

【举例】

请参见 HI_MPI_VI_SetPubAttr 的举例。

【相关主题】

HI_MPI_VI_UnBindOutput

HI_MPI_VI_UnBindOutput

【描述】

视频输入通道与视频输出通道解绑定。

【语法】

```
HI_S32 HI_MPI_VI_UnBindOutput(VI_DEV ViDevId,VI_CHN ViChn,VO_CHN VoChn);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入
VoChn	VO 通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入或输出通道号无效。

接口返回值	含义
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

无。

【举例】

请参见 HI_MPI_VI_SetPubAttr 的举例。

【相关主题】

HI_MPI_VI_BindOutput

HI_MPI_VI_SetSrcFrameRate

【描述】

设置视频输入通道的原始帧率。

【语法】

```
HI_S32 HI_MPI_VI_SetSrcFrameRate(VI_DEV ViDevId,VI_CHN ViChn,HI_U32
u32ViFramerate);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入
u32ViFramerate	原始帧率。 取值范围：大于 0。	输入

【返回值】

返回值	描述
正数值	有效返回值。

返回值	描述
非正数值	无效返回值。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入通道号无效。
HI_ERR_VI_INVALID_PARA	输入参数无效。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

VI 软件内部根据用户设置的原始帧率和目标帧率，进行视频图像帧捕获的帧率控制，原始帧率和目标帧率的初始默认值为无效值–1，只有两个帧率都设置为有效值，软件才会控制帧率，即按照原始帧率和目标帧率的比例输出视频帧图像。

【举例】

```
HI_S32 s32ret;
HI_U32 u32SrcFrmRate = 25;
HI_U32 u32FrmRate = 5;

/* set public attribute of VI device*/
/* enable VI device*/
/* set attribute of VI channel*/
/* ... ... */

/* set SRC framerate of VI channel*/
s32ret = HI_MPI_VI_SetFrameRate(0, 0, u32SrcFrmRate);
if (s32ret)
{
    return -1;
}

/* set target framerate of VI channel*/
s32ret = HI_MPI_VI_SetFrameRate(0, 0, u32FrmRate);
```

```
if (s32ret)
{
    return -1;
}
/* enable VI channel*/
/* ... ... */
/* also, you can set target framerate afer enable vi channel */
```

【相关主题】

无。

HI_MPI_VI_GetSrcFrameRate

【描述】

获取视频输入通道的目标帧率。

【语法】

```
HI_S32 HI_MPI_VI_GetSrcFrameRate(VI_DEV ViDevId,VI_CHN ViChn,HI_U32
*pu32ViFramerate);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入
pu32ViFramerate	目标帧率指针。	输出

【返回值】

返回值	描述
正数值	有效返回值。
非正数值	无效返回值。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。

接口返回值	含义
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入通道号无效。
HI_ERR_VI_FAILED_NOTCONFIG	原始帧率未配置。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

如果未设置原始帧率，则返回原始帧率未配置的错误码。

【举例】

无。

【相关主题】

HI_MPI_VI_SetFrameRate

HI_MPI_VI_SetFrameRate

【描述】

设置视频输入通道的目标帧率。

【语法】

```
HI_S32 HI_MPI_VI_SetFrameRate(VI_DEV ViDevId,VI_CHN ViChn,HI_U32
u32ViFramerate);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入
u32ViFramerate	目标帧率。 取值范围：[0, 30]	输入

【返回值】

返回值	描述
正数值	有效返回值。
非正数值	无效返回值。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入通道号无效。
HI_ERR_VI_NOT_SUPPORT	操作不支持。
HI_ERR_VI_INVALID_PARA	输入参数无效。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

- 支持视频输入通道设置指定范围内的任意目标帧率，以实现低帧率采集视频帧，降低系统性能消耗。
- 必须先设置 VI 通道的原始输入帧率，才能调用此接口设置目标帧率，且目标帧率不能大于原始帧率。
- 支持 VI 通道启用后再动态设置目标帧率。
- 如果用户未调用此接口设置 VI 通道帧率，VI 软件内部不做任何帧率控制。
- 受 VI 硬件性能限制影响，多路 D1 通道的帧率设置需要控制在一定范围之内，否则将出现丢帧以及帧采集不均匀的情况（但软件对此并不做约束和限制）。

【举例】

请参见 HI_MPI_VI_SetPubAttr 的举例。

【相关主题】

无。

HI_MPI_VI_GetFrameRate

【描述】

获取视频输入通道的目标帧率。

【语法】

```
HI_S32 HI_MPI_VI_GetFrameRate(VI_DEV ViDevId,VI_CHN ViChn,HI_U32
*pu32ViFramerate);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入
pu32ViFramerate	目标帧率指针。	输出

【返回值】

返回值	描述
正数值	有效返回值。
非正数值	无效返回值。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入通道号无效。
HI_ERR_VI_FAILED_NOTCONFIG	目标帧率未设置。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

如果未设置目标帧率，则返回目标帧率未配置的错误码。

【举例】

无。

【相关主题】

无。

HI_MPI_VI_SetUserPic

【描述】

设置用户图片的帧信息。

【语法】

```
HI_S32 HI_MPI_VI_SetUserPic(VI_DEV ViDevId,VI_CHN
ViChn,VIDEO_FRAME_INFO_S *pstVFrame);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入
pstVFrame	用户图片的帧信息结构指针。	输入

【返回值】

返回值	描述
正数值	有效返回值。
非正数值	无效返回值。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入通道号无效。
HI_ERR_VI_NOT_PERM	操作不允许。
HI_ERR_VI_INVALID_PARA	参数设置无效。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

- 目前此接口中的参数 VI 设备和通道号未被使用，即设置用户图像针对的是所有 VI 通道，而不是具体某个 VI 通道。
- 设置完用户图片后，即可调用 HI_MPI_VI_EnableUserPic 接口对指定 VI 通道的启用插入用户图片，此时 VI 通道输出的数据即为所配置的图片 YUV 数据；一般用于视频信号丢失时，VI 通道输出 NoVideo 图片。
- 用户图片的视频帧信息结构中，需要设置图片的宽度、高度、行间隔、YUV 格式以及 Y 分量和 C 分量数据的物理地址；可以从 MPP 公共缓冲池中获取一块相应大小的视频缓存块，从缓存块信息中得到存放 YUV 数据的物理地址，然后将物理地址映射到用户空间，即可对这块内存进行 YUV 数据的填充操作；注意只支持 semi-planar YUV420、semi-planarYUV422 的格式，因此填充数据时需要遵循先存 Y 分量数据，再存 UV 分量间插数据的存储顺序（小端字节序先 V 后 U）。
- VIU 模块在启用插入用户图片时，会直接使用设置的用户图片帧信息中的物理地址，因此设置完用户图片后，不应该释放或销毁其视频缓存块，除非确认不再使用。可以通过再次调用此接口设置另外一块 VideoBuffer 以修改图片信息。
- 启用 VI 通道或启用插入用户图片时，此接口都可以动态调用。

【举例】

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

/* you need get width and height of pictrue */
u32LumaSize = (u32LStride * u32Height);
u32ChrmSize = (u32CStride * u32Height) >> 2;/* 420*/
u32Size = u32LumaSize + (u32ChrmSize << 1);

/* get video buffer block form common pool */
VbBlk = HI_MPI_VB_GetBlock(VB_INVALID_POOLID, u32Size);
if (VB_INVALID_HANDLE == VbBlk)
{
```

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

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

    /* get pool id */
    pstVFrameInfo->u32PoolId = HI_MPI_VB_Handle2PoolId(VbBlk);
    if (VB_INVALID_POOLID == pstVFrameInfo->u32PoolId)
    {
        return -1;
    }

    pstVFrameInfo->stVFrame.u32PhyAddr[0] = u32PhyAddr;
    pstVFrameInfo->stVFrame.u32PhyAddr[1] = pstVFrameInfo-
>stVFrame.u32PhyAddr[0] + u32LumaSize;
    pstVFrameInfo->stVFrame.u32PhyAddr[2] = pstVFrameInfo-
>stVFrame.u32PhyAddr[1] + u32ChrmSize;

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

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

    /* now you need get YUV Semi Palnar Data ,fill them to the virtual
address */
    /* ... ... */
    /* ... ... */
```

```
/* enabel VI channel ... ... */

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

/* ... ... */

/* enable insert user pic if you need */
s32ret = HI_MPI_VI_EnableUserPic(0, 0);
if (s32ret)
{
    return -1;
}

/* ... ... */

/* disable insert user pic if you don't need */
s32ret = HI_MPI_VI_DisableUserPic(0, 0);
if (s32ret)
{
    return -1;
}
```

【相关主题】

无。

HI_MPI_VI_EnableUserPic

【描述】

启用插入用户图片。

【语法】

```
HI_S32 HI_MPI_VI_EnableUserPic(VI_DEV ViDevId,VI_CHN ViChn);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入

参数名称	描述	输入/输出
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
正数值	有效返回值。
非正数值	无效返回值。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入通道号无效。
HI_ERR_VI_NOT_PERM	操作不允许。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

- 调用插入用户图片后，VI 模块不启动 VIU 硬件采集 AD 的视频帧数据，直接将设置的用户图像帧发送到 VENC 和 VO。
- 启用插入用户图片之前，需要先设置用户图片帧信息。
- 此接口可以重复调用。

【举例】

请参见 HI_MPI_VI_SetUserPic 的举例。

【相关主题】

无。

HI_MPI_VI_DisableUserPic

【描述】

禁用插入用户图片。

【语法】

```
HI_S32 HI_MPI_VI_DisableUserPic(VI_DEV ViDevId,VI_CHN ViChn);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
正数值	有效返回值。
非正数值	无效返回值。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入通道号无效。
HI_ERR_VI_NOT_PERM	操作不允许。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

- VI 通道不再需要输出用户图片时，应该调用此接口以恢复输出 AD 的原始视频数据。
- 此接口可以重复调用。

【举例】

请参见 HI_MPI_VI_SetUserPic 的举例。

【相关主题】

无。

HI_MPI_VI_SetMinorChnAttr

【描述】

设置 VI 通道次属性。

【语法】

```
HI_S32 HI_MPI_VI_SetMinorChnAttr(VI_DEV ViDevId,VI_CHN ViChn,const
VI_CHN_ATTR_S *pstAttr);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入
pstAttr	VI 通道次属性指针。 动态属性。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。

接口返回值	含义
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入通道号无效。
HI_ERR_VI_INVALID_PARA	输入参数无效。
HI_ERR_VI_INVALID_NULL_PTR	空指针错误。
HI_ERR_VI_FAILED_NOTCONFIG	视频输入设备或通道的属性未配置。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

- 所谓“通道次属性”是相对主属性而言的，主属性在函数 HI_MPI_VI_SetChnAttr 中配置；主、次属性的概念只有在对 VI 做了帧率控制时才有效，VI 采集图像时，按配置的目标帧率采集主属性大小的图像，其他帧则按次属性大小采集。例如：设置原始帧率为 25，目标帧率为 5，主属性大小为 D1，次属性大小为 CIF，那么实际视频捕获则为 5 帧输出 D1 大小图像，剩余 20 帧输出 CIF 大小图像。
- 此接口的主要应用场景为预览实时满帧率显示，8 路 D1 编码非实时低帧率。
- 如果已配置 VI 帧率，但未配置通道次属性，则需按照目标帧率只输出主属性大小的视频图像；设置次属性后也可以通过调用本接口将属性结构体指针置为 NULL 来禁止次属性图像输出。
- 可以在 VI 通道启用后动态更改通道次属性。
- 使用此方式输出图像大小持续变化的 VI 视频图像给 VENC 和 VO 模块时，需要对 VENC 和 VO 的图像显示做特殊配置：
  - 调用 HI_MPI_VPP_SetConf 设置编码通道组缩放模式为 BOTTOM2。
  - 调用 HI_MPI_VO_SetChnField 设置 VO 显示底场。

【举例】

```
HI_S32 s32ret;
HI_U32 u32SrcFrmRate = 25;
HI_U32 u32FrmRate = 5;
VI_CHN_ATTR_S stAttr;

/* set public attribute of VI device*/
/* ... ... */

/* enable VI device*/
```

```
/* ... ... */

/* set main attribute of VI channel, size is D1 */
/* ... ... */

HI_MPI_VI_GetChnAttr(ViDevId,ViChn,&stAttr);
stAttr.enCapSel = VI_CAPSEL_BOTTOM;
stAttr.bDownScale = HI_TRUE;

/* set minor attribute of VI channel, size is CIF */
if (HI_MPI_VI_SetChnMinorAttr(ViDevId,ViChn,&stAttr))
{
   BBIT_ERR("set chn attr ex fail\n");
   return -1;
}

/* set SRC framerate of VI channel*/
s32ret = HI_MPI_VI_SetFrameRate(0, 0, u32SrcFrmRate);
if (s32ret)
{
    return -1;
}

/* set target framerate of VI channel*/
s32ret = HI_MPI_VI_SetFrameRate(0, 0, u32FrmRate);
if (s32ret)
{
    return -1;
}
/* enable VI channel*/
/* ... ... */

/* enable VO dev and VO chn */
/* ... ... */

/* set vo field bottom */
HI_MPI_VO_SetChnField(i, VO_FIELD_BOTTOM);

/* create group/venc */
/* ... ... */

/* set group scale mode VPP_SCALE_MODE_USEBOTTOM2 */
VIDEO_PREPROC_CONF_S stConf;
HI_MPI_VPP_GetConf(s32Grp, &stConf);
```

```
stConf.enScaleMode = VPP_SCALE_MODE_USEBOTTOM2;
HI_MPI_VPP_SetConf(s32Grp, &stConf);
```

【相关主题】

HI_MPI_VI_GetMinorChnAttr

HI_MPI_VI_GetMinorChnAttr

【描述】

获取 VI 通道次属性。

【语法】

```
HI_S32 HI_MPI_VI_GetMinorChnAttr(VI_DEV ViDevId,VI_CHN
ViChn,VI_CHN_ATTR_S *pstAttr);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入
pstAttr	VI 通道属性指针。 动态属性。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VI_INVALID_DEVID	视频输入设备号无效。
HI_ERR_VI_INVALID_CHNID	视频输入通道号无效。
HI_ERR_VI_INVALID_NULL_PTR	空指针错误。

接口返回值	含义
HI_ERR_VI_FAILED_NOTCONFIG	视频输入属性未配置。
HI_ERR_VI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

必须先设置通道属性再获取属性，否则将返回 HI_ERR_VI_FAILED_NOTCONFIG。

【举例】

请参见 HI_MPI_VI_SetMinorChnAttr 的举例。

【相关主题】

HI_MPI_VI_SetChnAttr

HI_MPI_VI_GetFd

【描述】

获取 VI 通道对应的设备文件句柄。

【语法】

```
HI_S32 HI_MPI_VI_GetFd(VI_DEV ViDevId,VI_CHN ViChn);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
正数值	有效返回值。
非正数值	无效返回值。

【错误码】

无。

【需求】

- 头文件：hi_comm_vi.h、mpi_vi.h
- 库文件：libmpi.a

【注意】

用户可以获取文件句柄实现多通道 Select 获取视频帧数据。

【举例】

```
VI_DEV ViDevId = 0;
VI_CHN ViChn = 0;
VIDEO_NORM_E stNorm;
HI_S32 s32ViFd = 0;

/* enable vi dev and vi chn */

/* get video frame from vi chn */
s32ViFd = HI_MPI_VI_GetFd(ViDevId, ViChn);
if (s32ViFd <= 0)
{
    return HI_FAILURE;
}
```

【相关主题】

无。

2.2.3 视频输出

视频输出（VO）实现启用视频输出设备或通道、发送视频数据到输出通道等功能。

该功能模块提供以下 MPI：

- HI_MPI_VO_Enable：启用视频输出设备。
- HI_MPI_VO_Disable：禁用视频输出设备。
- HI_MPI_VO_EnableChn：启用指定的视频输出通道。
- HI_MPI_VO_DisableChn：禁用指定的视频输出通道。
- HI_MPI_VO_SetPubAttr：配置视频输出的公共属性。
- HI_MPI_VO_GetPubAttr：获取视频输出的公共属性。
- HI_MPI_VO_SetChnAttr：配置指定视频输出通道的属性。
- HI_MPI_VO_GetChnAttr：获取指定视频输出通道的属性。
- HI_MPI_VO_SendFrame：将视频图像送入指定视频输出通道显示。
- HI_MPI_VO_EnableDeflicker：使能视频输出抗闪烁。

- HI_MPI_VO_DisableDeflicker：禁用视频输出抗闪烁。
- HI_MPI_VO_SetChnField：设置指定视频输出通道的帧场显示策略。
- HI_MPI_VO_GetChnField：获取指定视频输出通道的帧场显示策略。
- HI_MPI_VO_SetFrameRate：设置指定视频输出通道的显示帧率。
- HI_MPI_VO_GetFrameRate：获取指定视频输出通道的显示帧率。
- HI_MPI_VO_ChnPause：暂停指定的视频输出通道。
- HI_MPI_VO_ChnResume：恢复指定的视频输出通道。
- HI_MPI_VO_ChnStep：单帧播放指定的视频输出通道。
- HI_MPI_VO_CreateSyncGroup：创建视频输出同步组。
- HI_MPI_VO_DestroySyncGroup：销毁视频输出同步组。
- HI_MPI_VO_RegisterSyncGroup：注册视频输出通道到视频输出同步组。
- HI_MPI_VO_UnRegisterSyncGroup：注销视频输出通道到视频输出同步组。
- HI_MPI_VO_SyncGroupStart：启动视频输出同步组。
- HI_MPI_VO_SyncGroupStop：停止视频输出同步组。
- HI_MPI_VO_SetSyncGroupFrameRate：设置视频输出同步组帧率。
- HI_MPI_VO_GetSyncGroupFrameRate：获取视频同步组帧率。
- HI_MPI_VO_PauseSyncGroup：暂停视频输出同步组。
- HI_MPI_VO_ResumeSyncGroup：恢复视频输出同步组。
- HI_MPI_VO_StepSyncGroup：单帧播放视频输出同步组。
- HI_MPI_VO_SetSyncGroupBase：设置多通道同步组的基准时间戳。
- HI_MPI_VO_GetSyncGroupBase：获取多通道同步组的基准时间戳。
- HI_MPI_VO_SetZoomInWindow：设置视频输出局部放大窗口。
- HI_MPI_VO_GetZoomInWindow：获取视频输出局部放大窗口参数。
- HI_MPI_VO_GetChnPts：获取视频输出通道当前播放图像的时间戳。
- HI_MPI_VO_SetAttrBegin：设置属性开始。
- HI_MPI_VO_SetAttrEnd：设置属性结束。
- HI_MPI_VO_ChnShow：设置显示通道。
- HI_MPI_VO_ChnHide：设置隐藏通道。
- HI_MPI_VO_Query：查询视频输出通道状态。
- HI_MPI_VO_SetDisplayFrameRate：设置屏幕显示低帧率。
- HI_MPI_VO_GetDisplayFrameRate：获取屏幕显示帧率。
- HI_MPI_VO_GetChnFrame：获取输出通道图像数据。
- HI_MPI_VO_ReleaseChnFrame：释放输出通道图像数据。
- HI_MPI_VO_GetScreenFrame：获取输出屏幕图像数据。
- HI_MPI_VO_ReleaseScreenFrame：释放输出屏幕图像数据。

HI_MPI_VO_Enable

【描述】

启用视频输出设备。

【语法】

```
HI_S32 HI_MPI_VO_Enable(HI_VOID);
```

【参数】

无。

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_FAILED_NOTCONFIG	视频输出设备未配置。
HI_ERR_VO_BUSY	系统忙，一般表明此前未调用系统初始化函数。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 由于系统没有为 VO 公共属性设置缺省值，VO 启用前必须进行一次公共属性设置，否则直接返回错误信息。
- 允许多次重复启用视频输出设备，不返回失败。

【举例】

```
HI_S32 s32ret;
VO_PUB_ATTR_S VoAttr;
VO_CHN_ATTR_S VoChnAttr;
VO_CHN VoChn = 0;
VIDEO_FRAME_INFO_S stVFrame;
VO_DISPLAY_FIELD_E enVoField;
HI_U64 u64ChnPts = 0;
```

```
memset(&VoAttr, 0, sizeof(VO_PUB_ATTR_S));
VoAttr.stTvConfig.stComposeMode = VIDEO_ENCODING_MODE_PAL;  /*PAL Mode*/
VoAttr.u32BgColor = 0x0;
/* set public attribute of vo device */
s32ret = HI_MPI_VO_SetPubAttr(&VoAttr);
if (HI_SUCCESS != s32ret)
{
    printf("vo set pub attr failed! \n");
    return s32ret;
}

/* enable vo device */
s32ret = HI_MPI_VO_Enable();
if (HI_SUCCESS != s32ret)
{
    printf("enable vo device failed! \n");
    return s32ret;
}

VoChnAttr.bZoomEnable = 1;
VoChnAttr.u32Priority = 1;
VoChnAttr.stRect.s32X = 0;
VoChnAttr.stRect.s32Y = 0;
VoChnAttr.stRect.u32Width = 704;
VoChnAttr.stRect.u32Height = 576;
/* set attribute of vo chn*/
s32ret = HI_MPI_VO_SetChnAttr(VoChn, &VoChnAttr);
if (HI_SUCCESS != s32ret)
{
    printf("vo set chn attr failed! \n");
    return s32ret;
}
/* enable vo chn */
s32ret = HI_MPI_VO_EnableChn(VoChn);
if (HI_SUCCESS != s32ret)
{
    printf("enable vo chn failed! \n");
    return s32ret;
}

/* get video frame from vdec chn or file ...*/

/* send frame to vo chn */
s32ret = HI_MPI_VO_SendFrame(VoChn, &stVFrame);
```

```
if (HI_SUCCESS != s32ret)
{
    return s32ret;
}
/* get channel field */
s32ret = HI_MPI_VO_GetChnField(0, &enVoField);
if(HI_SUCCESS != s32ret)
{
   return s32ret;
}

/* set channel field */
s32ret = HI_MPI_VO_SetChnField(VoChn, VO_FIELD_BOTTOM);
if(HI_SUCCESS != s32ret)
{
   return s32ret;
}

/* set framrate */
s32ret = HI_MPI_VO_SetFrameRate(VoChn, 12);
if(HI_SUCCESS != s32ret)
{
   return s32ret;
}

/* enable deflicker */
s32ret = HI_MPI_VO_EnableDeflicker();
if(HI_SUCCESS != s32ret)
{
   return s32ret;
}

/* disable deflicker */
s32ret = HI_MPI_VO_DisableDeflicker();
if(HI_SUCCESS != s32ret)
{
   return s32ret;
}
/* get current channel's pts */
s32ret = HI_MPI_VO_GetChnPts (VoChn, &u64ChnPts);
if(HI_SUCCESS != s32ret)
{
    printf("get channel pts failed!");
   return s32ret;
```

```
}
else
{
    printf("%d pts is: %llu\n", VoChn, u64ChnPts);
}

(void)HI_MPI_VO_DisableChn(VoChn);

(void)HI_MPI_VO_Disable();
```

【相关主题】

HI_MPI_VO_Disable

HI_MPI_VO_Disable

【描述】

禁用视频输出设备。

【语法】

```
HI_S32 HI_MPI_VO_Disable(HI_VOID);
```

【参数】

无。

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_FAILED_CHNOTDISABLE	有还在使能状态的视频输出通道，需禁用。
HI_ERR_VO_BUSY	系统忙，一般表明此前未调用系统初始化函数。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h

- 库文件：libmpi.a

【注意】

- 此接口不支持多进程，一旦调用则其它进程的视频输出也将停止。
- 允许重复禁用 VO 设备，不返回失败。
- 必须先禁用视频输出通道，否则禁用视频设备会失败。

【举例】

请参见 HI_MPI_VO_Enable 的举例。

【相关主题】

HI_MPI_VO_Enable

HI_MPI_VO_EnableChn

【描述】

启用指定的视频输出通道。

【语法】

```
HI_S32 HI_MPI_VO_EnableChn(VO_CHN VoChn);
```

【参数】

参数名称	描述	输入/输出
VoChn	视频输出通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	视频输出通道号无效。
HI_ERR_VO_FAILED_NOTENABLE	没有使能视频输出设备。
HI_ERR_VO_FAILED_NOTCONFIG	没有配置视频输出设备。

接口返回值	含义
HI_ERR_VO_BUSY	系统忙，一般表明此前未调用系统初始化函数。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前必须先启用视频输出设备，否则将返回失败。
- 如果未启用通道将无法正常工作，所以通道使用前必须启用成功。
- 允许重复启用同一视频输出通道，不返回失败。

【举例】

请参见 HI_MPI_VO_Enable 的举例。

【相关主题】

HI_MPI_VO_DisableChn

HI_MPI_VO_DisableChn

【描述】

禁用指定的视频输出通道。

【语法】

```
HI_S32 HI_MPI_VO_DisableChn(VO_CHN VoChn);
```

【参数】

参数名称	描述	输入/输出
VoChn	视频输出通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VO_BUSY	系统忙，一般表明此前未调用系统初始化函数。
HI_ERR_VO_GRP_CHN_NOT_UNREG	同步组通道未注销

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

允许重复禁用同一视频输出通道，不返回失败。

【举例】

请参见 HI_MPI_VO_Enable 的举例。

【相关主题】

HI_MPI_VO_EnableChn

HI_MPI_VO_SetPubAttr

【描述】

配置视频输出的公共属性。

【语法】

```
HI_S32 HI_MPI_VO_SetPubAttr(VO_PUB_ATTR_S *pstAttr);
```

【参数】

参数名称	描述	输入/输出
pstAttr	视频输出公共属性结构体指针。 静态属性。	输入

【返回值】

返回值	描述
0	成功。

返回值	描述
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_FAILED_NOTDISABLE	VO 未处在禁用状态。
HI_ERR_VO_NULL_PTR	参数中有空指针。
HI_ERR_VO_ILLEGAL_PARAM	参数超出合法范围。
HI_ERR_VO_BUSY	系统忙，一般表明此前未调用系统初始化函数。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

必须在视频输出设备禁用情况下，才允许配置。

【举例】

请参见 HI_MPI_VO_Enable 的举例。

【相关主题】

HI_MPI_VO_GetPubAttr

HI_MPI_VO_GetPubAttr

【描述】

获取视频输出的公共属性。

【语法】

```
HI_S32 HI_MPI_VO_GetPubAttr(VO_PUB_ATTR_S *pstAttr);
```

【参数】

参数名称	描述	输入/输出
pstAttr	视频输出公共属性结构体指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_NULL_PTR	参数中有空指针。
HI_ERR_VO_FAILED_NOTCONFIG	没有配置视频输出设备。
HI_ERR_VO_BUSY	系统忙，一般表明此前未调用系统初始化函数。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

在设置视频输出的公共属性后才能获取属性，否则返回错误信息。

【举例】

```
VO_PUB_ATTR_S stAttr;
if (HI_MPI_VO_GetPubAttr(&stAttr) != HI_SUCCESS )
{
    printf("get vo publiy attr failed!);
}
```

【相关主题】

HI_MPI_VO_SetPubAttr

HI_MPI_VO_SetChnAttr

【描述】

配置指定视频输出通道的属性。

【语法】

```
HI_S32 HI_MPI_VO_SetChnAttr(VO_CHN VoChn, VO_CHN_ATTR_S *pstAttr);
```

【参数】

参数名称	描述	输入/输出
VoChn	视频输出通道的通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入
pstAttr	视频通道属性指针。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VO_ILLEGAL_PARAM	参数超出合法范围。
HI_ERR_VO_NULL_PTR	参数中有空指针。
HI_ERR_VO_BUSY	系统忙，一般表明此前未调用系统初始化函数。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 属性中的优先级，数值越大优先级越高，最大值为 15，最小值为 1。
  - 当多个通道有重叠的显示区域时，优先级高的通道图像将覆盖优先级低的通道。
  - 优先级相同的各通道有重叠时，默认通道号大的图像将覆盖通道号小的通道图像。
- 各通道起始横坐标加宽度的和须小于等于 720，各通道起始纵坐标加高度的和须小于等于 576（PAL 制）或 480（NTSC 制）。
- 如果缩放标识为 TRUE，则启用缩放模式。

- 该接口为动态设置接口，可在 VO 设备使能的情况下调用。
- 通道的起始位置和宽高必须能被 8 整除。

【举例】

请参见 HI_MPI_VO_Enable 的举例。

【相关主题】

HI_MPI_VO_GetChnAttr

HI_MPI_VO_GetChnAttr

【描述】

获取指定视频输出通道的属性。

【语法】

```
HI_S32 HI_MPI_VO_GetChnAttr(VO_CHN VoChn, VO_CHN_ATTR_S *pstAttr);
```

【参数】

参数名称	描述	输入/输出
VoChn	视频输出通道的通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入
pstAttr	视频通道属性指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VO_NULL_PTR	参数中有空指针。
HI_ERR_VO_BUSY	系统忙，一般表明此前未调用系统初始化函数。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

无。

【举例】

```
int vochn = 0;
VO_CHN_ATTR_S stAttr;
if (HI_MPI_VO_GetChnAttr (vochn, &stAttr) != HI_SUCCESS )
{
    printf("get vo %d chn attr failed!", vochn);
}
```

【相关主题】

HI_MPI_VO_SetChnAttr

HI_MPI_VO_SendFrame

【描述】

将视频图像送入指定输出通道显示。

【语法】

```
HI_S32 HI_MPI_VO_SendFrame(VO_CHN VoChn, VIDEO_FRAME_INFO_S *pstVFrame);
```

【参数】

参数名称	描述	输入/输出
VoChn	视频输出通道的通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入
pstVFrame	视频数据信息指针。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VO_NULL_PTR	参数中有空指针。
HI_ERR_VO_FAILED_NOTENABLE	没有使能视频输出设备。
HI_ERR_VO_ILLEGAL_PARAM	传入的参数非法。
HI_ERR_VO_BUSY	系统忙，一般表明此前未调用系统初始化函数或输出 buffer 已满。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

pstVFrame 一般为从解码端获取的数据，确保不作任何修改，直接传下来即可。

【举例】

请参见 HI_MPI_VO_Enable 的举例。

【相关主题】

无。

HI_MPI_VO_EnableDeflicker

【描述】

使能视频输出抗闪烁。

【语法】

```
HI_S32 HI_MPI_VO_EnableDeflicker();
```

【参数】

无。

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

无。

【举例】

请参见 HI_MPI_VO_Enable 的举例。

【相关主题】

无。

HI_MPI_VO_DisableDeflicker

【描述】

禁用视频输出抗闪烁。

【语法】

```
HI_S32 HI_MPI_VO_DisableDeflicker();
```

【参数】

无。

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

无。

【举例】

请参见 HI_MPI_VO_Enable 的举例。

【相关主题】

无。

HI_MPI_VO_SetChnField

【描述】

设置指定视频输出通道的帧场显示策略。

主要的应用场合有两个，一是在预览 CIF 图像时，将视频输入设置为两场的 CIF，对应 VO 通道设置成显示两场，将比只显示一场的流畅性要好。二是在低帧率预览的情况下，如在低帧率的 8 路 D1 预览时，应该将对应 VO 通道设置成只显示一场，否则会出现回顿的现象。

【语法】

```
HI_S32 HI_MPI_VO_SetChnField
            (VO_CHN VoChn,const VO_DISPLAY_FIELD_E enField)
```

【参数】

参数名称	描述	输入/输出
VoChn	视频输出通道的通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入
enField	视频通道显示帧场策略。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VO_ILLEGAL_PARAM	参数非法。
HI_ERR_VO_BUSY	系统忙，一般表明此前未调用系统初始化函数。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 如果送 VO 显示的图像为非两场间插的图像（例如单场 CIF 图像），则调用此接口对图像显示无实质影响。
- 如果不调用此接口设置，则 VO 按默认两场显示（即 VO_FIELD_BOTH）。

【举例】

```
/* 这是一个预览低帧率的例子，先将视频输入设置成8帧每秒，然后对应VO通道用底场显示 */
if (HI_SUCCESS!=HI_MPI_VI_SetFrameRate(0,0,8))
{
    printf("HI_MPI_VI_SetFrameRate failed !\n");
    return HI_FAILURE;
}
if (HI_SUCCESS!=HI_MPI_VO_SetChnField(0,VO_FIELD_BOTTOM))
{
    printf("HI_MPI_VO_SetChnField failed !\n");
    return HI_FAILURE;
}
```

【相关主题】

无。

HI_MPI_VO_GetChnField

【描述】

获取指定视频输出通道的帧场显示策略。

【语法】

```
HI_S32 HI_MPI_VO_GetChnField
            (VO_CHN VoChn,const VO_DISPLAY_FIELD_E *penField)
```

【参数】

参数名称	描述	输入/输出
VoChn	视频输出通道的通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入
penField	视频通道显示帧场策略。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VO_NULL_PTR	参数中有空指针。
HI_ERR_VO_BUSY	系统忙，一般表明此前未调用系统初始化函数。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

如未配置则返回系统默认策略 VO_FIELD_BOTH。

【举例】

请参见 HI_MPI_VO_Enable 的举例。

【相关主题】

无。

HI_MPI_VO_SetFrameRate

【描述】

设置指定视频输出通道的显示帧率。

【语法】

```
HI_S32 HI_MPI_VO_SetFrameRate (VO_CHN VoChn, HI_S32 s32VoFramerate);
```

【参数】

参数名称	描述	输入/输出
VoChn	视频输出通道的通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入
s32VoFramerate	视频通道显示帧率。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VO_ILLEGAL_PARAM	参数非法。
HI_ERR_VO_FAILED_NOTCONFIG	VO 公共属性未配置。
HI_ERR_VO_BUSY	系统忙，一般表明此前未调用系统初始化函数。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 需先设置 VO 的公共属性，才能调用此接口，否则返回失败。
- 帧率设置范围为[-64x,，0) 和（0，+64x]。负数的倍数用于倒退操作，这个时候需要用户来倒序送图像到 VO，即送时间戳递减的图像。正的倍速受限于解码器性

能，一般情况下单路 D1 最大支持 4x 全解，CIF 最大支持 16x 全解，QCIF 最大支持 64x 全解。当设置帧率大于最大解码能力，播放速度将小于实际设置帧率。

【举例】

请参见 HI_MPI_VO_Enable 的举例。

【相关主题】

无。

HI_MPI_VO_GetFrameRate

【描述】

获取指定视频输出通道的显示帧率。

【语法】

```
HI_S32 HI_MPI_VO_GetFrameRate(VO_CHN VoChn, HI_S32 *ps32VoFramerate);
```

【参数】

参数名称	描述	输入/输出
VoChn	视频输出通道的通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入
ps32VoFramerate	视频通道显示帧率。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VO_NULL_PTR	参数中有空指针。
HI_ERR_VO_BUSY	系统忙，一般表明此前未调用系统初始化函数。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

获取的为用户配置的帧率，如用户未设置则固定返回满帧。

【举例】

请参见 HI_MPI_VO_Enable 的举例。

【相关主题】

无。

HI_MPI_VO_ChnPause

【描述】

暂停指定的视频输出通道。

【语法】

```
HI_S32 HI_MPI_VO_ChnPause(VO_CHN VoChn);
```

【参数】

参数名称	描述	输入/输出
VoChn	视频输出通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	视频输出通道号无效。
HI_ERR_VO_FAILED_NOTCONFIG	没有配置视频输出设备。

接口返回值	含义
HI_ERR_VO_FAILED_NOTENABLE	没有使能视频输出设备。
HI_ERR_VO_FAILED_CHNOTENABLE	没有使能视频输出通道。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前必须先启用视频输出设备，否则将返回失败。
- 如果未启用通道将无法正常工作，所以通道使用前必须启用成功。
- VI-VO 预览时不推荐使用此接口。
- 允许重复暂停同一通道，不返回失败。

【举例】

```
HI_S32 s32ret;
VO_PUB_ATTR_S VoAttr;
VO_CHN_ATTR_S VoChnAttr;
VO_CHN VoChn = 0;
VIDEO_FRAME_INFO_S stVFrame;

memset(&VoAttr, 0, sizeof(VO_PUB_ATTR_S));
VoAttr.stTvConfig.stComposeMode = VIDEO_ENCODING_MODE_PAL;  /*PAL Mode*/
VoAttr.u32BgColor = 0x0;
/* set public attribute of vo device*/
s32ret = HI_MPI_VO_SetPubAttr(&VoAttr);
if (HI_SUCCESS != s32ret)
{
    printf("vo set pub attr failed! \n");
    return s32ret;
}

/* enable vo device*/
s32ret = HI_MPI_VO_Enable();
if (HI_SUCCESS != s32ret)
{
    printf("enable vo device failed! \n");
    return s32ret;
}

VoChnAttr.bZoomEnable = 1;
```

```
VoChnAttr.u32Priority = 1;
VoChnAttr.stRect.s32X = 0;
VoChnAttr.stRect.s32Y = 0;
VoChnAttr.stRect.u32Width = 704;
VoChnAttr.stRect.u32Height = 576;
/* set attribute of vo chn*/
s32ret = HI_MPI_VO_SetChnAttr(VoChn, &VoChnAttr)
if (HI_SUCCESS != s32ret)
{
    printf("vo set chn attr failed! \n");
    return s32ret;
}
/* enable vo chn */
s32ret = HI_MPI_VO_EnableChn(VoChn);
if (HI_SUCCESS != s32ret)
{
    printf("enable vo chn failed! \n");
    return s32ret;
}

/* pause current vo channel */
s32ret = HI_MPI_VO_ChnPause(VoChn);
if (HI_SUCCESS != s32ret)
{
    printf("pause vo chn failed! \n");
    return s32ret;
}

/* resume current vo channel */
s32ret = HI_MPI_VO_ChnResume(VoChn);
if (HI_SUCCESS != s32ret)
{
    printf("resume vo chn failed! \n");
    return s32ret;
}

while (getchar() != 'q')
{
    /* step forward current vo channel */
    s32ret = HI_MPI_VO_ChnStep(VoChn);
    if (HI_SUCCESS != s32ret)
    {
        printf("step play vo chn failed! \n");
        return s32ret;
```

```
    }
}

/* resume current vo channel */
s32ret = HI_MPI_VO_ChnResume(VoChn);
if (HI_SUCCESS != s32ret)
{
    printf("resume vo chn failed! \n");
    return s32ret;
}

/* resume current vo channel */
s32ret = HI_MPI_VO_ChnResume(VoChn);
if (HI_SUCCESS != s32ret)
{
    printf("resume vo chn failed! \n");
    return s32ret;
}

(void)HI_MPI_VO_DisableChn(VoChn);

(void)HI_MPI_VO_Disable();
```

【相关主题】

HI_MPI_VO_ChnResume

HI_MPI_VO_ChnResume

【描述】

恢复指定的视频输出通道。

【语法】

```
HI_S32 HI_MPI_VO_ChnResume(VO_CHN VoChn);
```

【参数】

参数名称	描述	输入/输出
VoChn	视频输出通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	视频输出通道号无效。
HI_ERR_VO_FAILED_NOTCONFIG	没有配置视频输出设备。
HI_ERR_VO_FAILED_NOTENABLE	没有使能视频输出设备。
HI_ERR_VO_FAILED_CHNOTENABLE	没有使能视频输出通道。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前必须先启用视频输出设备，否则将返回失败。
- 如果未启用通道将无法正常工作，所以通道使用前必须启用成功。
- 允许重复恢复同一通道，不返回失败。

【举例】

请参见 HI_MPI_VO_ChnPause 的举例。

【相关主题】

HI_MPI_VO_ChnResume

HI_MPI_VO_ChnStep

【描述】

单帧播放指定的视频输出通道。

【语法】

```
HI_S32 HI_MPI_VO_ChnStep(VO_CHN VoChn);
```

【参数】

参数名称	描述	输入/输出
VoChn	视频输出通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	视频输出通道号无效。
HI_ERR_VO_FAILED_NOTCONFIG	没有配置视频输出设备。
HI_ERR_VO_FAILED_NOTENABLE	没有使能视频输出设备。
HI_ERR_VO_FAILED_CHNOTENABLE	没有使能视频输出通道。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前必须先启用视频输出设备，否则将返回失败。
- 如果未启用通道将无法正常工作，所以通道使用前必须启用成功。
- 允许重复单帧播放同一通道，不返回失败。
- 恢复正常播放使用 HI_MPI_VO_ChnResume 接口。
- VI-VO 预览时不推荐使用此接口。

【举例】

请参见 HI_MPI_VO_ChnPause 的举例。

【相关主题】

HI_MPI_VO_ChnResume

HI_MPI_VO_CreateSyncGroup

【描述】

创建视频输出同步组。

【语法】

```
HI_S32 HI_MPI_VO_CreateSyncGroup(VO_GRP VoGroup, VO_SYNC_GROUP_ATTR_S
*pstSycGrpAttr);
```

【参数】

参数名称	描述	输入/输出
VoGroup	同步输出组组号。 取值范围：[0, VO_SYNC_MAX_GROUP)。	输入
pstSycGrpAttr	同步组属性结构体指针。 静态属性。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_GRP_INVALID_NUMBER	同步通道组组号无效。
HI_ERR_VO_NULL_PTR	同步组属性结构体指针为空。
HI_ERR_VO_FAILED_NOTENABLE	视频输出没有使能。
HI_ERR_VO_FAILED_NOTCONFIG	视频输出没有配置。
HI_ERR_VO_GRP_HAS_CREATED	指定的同步组已经创建。
HI_ERR_VO_GRP_INVALID_SYN_MODE	同步组同步模式无效。
HI_ERR_VO_GRP_INVALID_FRMRATE	同步组设置帧率无效。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前必须先启用视频输出设备，并对视频输出进行属性配置。
- 指定的同步组必须保证未被创建，否则返回已经创建错误码。

【举例】

请参见 HI_MPI_VO_SyncGroupStart 的举例。

【相关主题】

HI_MPI_VO_DestroySyncGroup

HI_MPI_VO_DestroySyncGroup

【描述】

销毁视频输出同步组。

【语法】

```
HI_S32 HI_MPI_VO_DestroySyncGroup(VO_GRP VoGroup);
```

【参数】

参数名称	描述	输入/输出
VoGroup	同步输出组组号。 取值范围：[0, VO_SYNC_MAX_GROUP)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_GRP_INVALID_NUMBER	同步通道组组号无效。
HI_ERR_VO_GRP_NOT_CREATE	同步通道组没有创建。

接口返回值	含义
HI_ERR_VO_GRP_CHN_NOT_EMPTY	同步组中的视频输出通道为注销完。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前必须保证同步通道已经创建。
- 如果同步组中的视频输出通道没有全部注销，调用会返回同步组非空的错误码。

【举例】

请参见 HI_MPI_VO_SyncGroupStart 的举例。

【相关主题】

HI_MPI_VO_CreateSyncGroup

HI_MPI_VO_RegisterSyncGroup

【描述】

注册视频输出通道到视频输出同步组。

【语法】

```
HI_S32 HI_MPI_VO_RegisterSyncGroup(VO_CHN VoChn, VO_GRP VoGroup);
```

【参数】

参数名称	描述	输入/输出
VoGroup	同步输出组组号。 取值范围：[0, VO_SYNC_MAX_GROUP)。	输入
VoChn	视频输出通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_FAILED_NOTENABLE	没有使能视频输出设备。
HI_ERR_VO_INVALID_CHNID	视频输出通道号无效。
HI_ERR_VO_GRP_INVALID_NUMBER	同步通道组组号无效。
HI_ERR_VO_GRP_NOT_CREATE	同步通道组没有创建。
HI_ERR_VO_GRP_CHN_FULL	同步组注册数已满。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前必须先启用视频输出设备，并且配置视频输出公共属性。
- 调用前必须保证视频输出通道配置并使能。
- 调用前必须保证同步组已经创建，并且非满。

【举例】

请参见 HI_MPI_VO_SyncGroupStart 的举例。

【相关主题】

HI_MPI_VO_UnRegisterSyncGroup

HI_MPI_VO_UnRegisterSyncGroup

【描述】

注销视频输出通道从视频输出同步组。

【语法】

```
HI_S32 HI_MPI_VO_UnRegisterSyncGroup(VO_CHN VoChn, VO_GRP VoGroup);
```

【参数】

参数名称	描述	输入/输出
VoGroup	同步输出组组号。 取值范围：[0, VO_SYNC_MAX_GROUP)。	输入

参数名称	描述	输入/输出
VoChn	视频输出通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_FAILED_NOTENABLE	没有使能视频输出设备。
HI_ERR_VO_INVALID_CHNID	视频输出通道号无效。
HI_ERR_VO_FAILED_CHNOTENABLE	视频输出通道未使能。
HI_ERR_VO_GRP_INVALID_NUMBER	同步通道组组号无效。
HI_ERR_VO_GRP_NOT_CREATE	同步通道组没有创建。
HI_ERR_VO_GRP_CHN_EMPTY	同步通道组中没有视频输出通道。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前必须先启用视频输出设备，并且配置视频输出公共属性。
- 调用前必须保证视频输出通道配置并使能。
- 同步通道组不能为空。

【举例】

请参见 HI_MPI_VO_SyncGroupStart 的举例。

【相关主题】

HI_MPI_VO_RegisterSyncGroup

HI_MPI_VO_SyncGroupStart

【描述】

启动视频输出同步组。

【语法】

```
HI_S32 HI_MPI_VO_SyncGroupStart(VO_GRP VoGroup);
```

【参数】

参数名称	描述	输入/输出
VoGroup	同步输出组组号。 取值范围：[0, VO_SYNC_MAX_GROUP)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_FAILED_NOTENABLE	没有使能视频输出设备。
HI_ERR_VO_FAILED_CHNOTENABLE	视频输出通道未使能。
HI_ERR_VO_GRP_INVALID_NUMBER	同步通道组组号无效。
HI_ERR_VO_GRP_NOT_CREATE	同步通道组没有创建。
HI_ERR_VO_GRP_INVALID_BASE_PTS	同步通道组基准 PTS 无效。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前必须先启用视频输出设备，并且配置视频输出公共属性。
- 调用前必须保证视频输出通道配置并使能。

- 调用前必须保证同步通道组已经创建，并且有视频输出通道注册。

【举例】

```
HI_S32 s32ret;
HI_U32 u32FrameRate = 12;
VO_CHN VoChn = 0;
VO_GRP VoGroup = 0;
VO_SYNC_GROUP_ATTR_S stSyncGrpAttr;
VO_SYNC_BASE_S stSyncBase;

memset(&stSyncGrpAttr, 0 , sizeof (VO_SYNC_GROUP_ATTR_S));

/* set group attributes and create synchronous group */
stSyncGrpAttr.u32SynFrameRate = g_u32NormFrameRate;
stSyncGrpAttr.enSyncMode = VO_SYNC_MODE_USR_CHN;
stSyncGrpAttr.u32UsrBaseChn = 3;

stSyncBase.VoChn = 0;
stSyncBase.u64BasePts = 64094025454LLU;s32ret  =
HI_MPI_VO_CreateSyncGroup(VoGroup, &stSyncGrpAttr);
if (HI_SUCCESS != s32ret)
{
    printf("vo create synchronous group failed! \n");
    return s32ret;
}

/* register a vo channel to synchronous group */
s32ret  =  HI_MPI_VO_RegisterSyncGroup(VoChn, VoGroup);
if (HI_SUCCESS != s32ret)
{
    printf("vo channel register synchronous group failed! \n");
    return s32ret;
}

/* start synchronous group */
s32ret  =  HI_MPI_VO_SyncGroupStart(VoGroup);
if (HI_SUCCESS != s32ret)
{
    printf("start synchronous group failed! \n");
    return s32ret;
}

s32ret  =  HI_MPI_VO_SetSyncGroupBase (VoGroup, &stSyncBase);
if (HI_SUCCESS != s32ret)
```

```
{
    printf("set synchronous group base pts failed! \n");
    return s32ret;
}

s32ret  =  HI_MPI_VO_GetSyncGroupBase (VoGroup, &stSyncBase);
if (HI_SUCCESS != s32ret)
{
    printf("get synchronous group base pts failed! \n");
    return s32ret;
}
else
{
    printf("current base pts = %llu\n", stSyncBase. u64BasePts);
}
/* set frame rate of synchronous group */
s32ret  =  HI_MPI_VO_SetSyncGroupFrameRate(VoGroup, u32FrameRate);
if (HI_SUCCESS != s32ret)
{
    printf("set synchronous group frame rate failed! \n");
    return s32ret;
}

/* pause group */
s32ret  =  HI_MPI_VO_PauseSyncGroup(VoGroup);
if (HI_SUCCESS != s32ret)
{
    printf("pause synchronous group failed! \n");
    return s32ret;
}

/* step group */
s32ret  =  HI_MPI_VO_StepSyncGroup(VoGroup);
if (HI_SUCCESS != s32ret)
{
    printf("step synchronous group failed! \n");
    return s32ret;
}

/* resume group */
s32ret  =  HI_MPI_VO_ResumeSyncGroup(VoGroup);
if (HI_SUCCESS != s32ret)
{
    printf("resume synchronous group failed! \n");
```

```
    return s32ret;
}


/* stop synchronous group */
s32ret  =  HI_MPI_VO_SyncGroupStop(VoGroup);
if (HI_SUCCESS != s32ret)
{
    printf("stop synchronous group failed! \n");
    return s32ret;
}

/* un-register vo channel from group */
s32ret  =  HI_MPI_VO_UnRegisterSyncGroup(VoChn, VoGroup);
if (HI_SUCCESS != s32ret)
{
    printf("un-register vo from synchronous group failed! \n");
    return s32ret;
}

/* destroy synchronous group */
s32ret  =  HI_MPI_VO_DestroySyncGroup(VoGroup);
if (HI_SUCCESS != s32ret)
{
    printf("destroy synchronous group failed! \n");
    return s32ret;
}
```

【相关主题】

HI_MPI_VO_SyncGroupStop

HI_MPI_VO_SyncGroupStop

【描述】

停止视频输出通道组。

【语法】

```
HI_S32 HI_MPI_VO_SyncGroupStop(VO_GRP VoGroup);
```

【参数】

参数名称	描述	输入/输出
VoGroup	同步输出组组号。 取值范围：[0, VO_SYNC_MAX_GROUP)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_GRP_INVALID_NUMBER	同步通道组组号无效。
HI_ERR_VO_GRP_NOT_CREATE	同步通道组没有创建。
HI_ERR_VO_GRP_NOT_START	同步通道组没有启动。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前必须先启用视频输出设备，并且配置视频输出公共属性。
- 调用前必须保证视频输出通道配置并使能。
- 调用前必须保证同步通道组已经创建，并且启动。

【举例】

请参见 HI_MPI_VO_SyncGroupStart 的举例。

【相关主题】

HI_MPI_VO_SyncGroupStart

HI_MPI_VO_SetSyncGroupFrameRate

【描述】

设置视频输出同步通道组帧率。

【语法】

```
HI_S32 HI_MPI_VO_SetSyncGroupFrameRate(VO_GRP VoGroup, HI_U32
u32VoGrpFramerate);
```

【参数】

参数名称	描述	输入/输出
VoGroup	同步输出组组号。 取值范围：[0, VO_SYNC_MAX_GROUP)。	输入
u32VoGrpFramerate	同步输出组预设帧率。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_GRP_INVALID_NUMBER	同步通道组组号无效。
HI_ERR_VO_GRP_INVALID_FRMRATE	同步通道组预设帧率无效。
HI_ERR_VO_GRP_NOT_CREATE	同步通道组没有创建。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前必须先启用视频输出设备，并且配置视频输出公共属性。
- 调用前必须保证视频输出通道配置并使能。
- 调用前必须保证同步通道组已经创建。

【举例】

请参见 HI_MPI_VO_SyncGroupStart 的举例。

【相关主题】

HI_MPI_VO_GetSyncGroupFrameRate

HI_MPI_VO_GetSyncGroupFrameRate

【描述】

获取视频输出通道组帧率。

【语法】

```
HI_S32 HI_MPI_VO_GetSyncGroupFrameRate(VO_GRP VoGroup, HI_U32
*pu32VoGrpFramerate);
```

【参数】

参数名称	描述	输入/输出
VoGroup	同步输出组组号。 取值范围：[0, VO_SYNC_MAX_GROUP)。	输入
pu32VoGrpFramerate	同步输出组帧率指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_GRP_INVALID_NUMBER	同步通道组组号无效。
HI_ERR_VO_GRP_NOT_CREATE	同步通道组没有创建。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前必须先启用视频输出设备，并且配置视频输出公共属性。
- 调用前必须保证视频输出通道配置并使能。
- 调用前必须保证同步通道组已经创建。

【举例】

请参见 HI_MPI_VO_SyncGroupStart 的举例。

【相关主题】

HI_MPI_VO_SetSyncGroupFrameRate

HI_MPI_VO_PauseSyncGroup

【描述】

暂停同步输出通道组。

【语法】

```
HI_S32 HI_MPI_VO_PauseSyncGroup(VO_GRP VoGroup);
```

【参数】

参数名称	描述	输入/输出
VoGroup	同步输出组组号。 取值范围：[0, VO_SYNC_MAX_GROUP)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_GRP_INVALID_NUMBER	同步通道组组号无效。
HI_ERR_VO_GRP_NOT_CREATE	同步通道组没有创建。
HI_ERR_VO_GRP_NOT_START	同步通道组没有启动。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前必须先启用视频输出设备，并且配置视频输出公共属性。
- 调用前必须保证视频输出通道配置并使能。
- 调用前必须保证同步通道组已经创建并启动。

【举例】

请参见 HI_MPI_VO_SyncGroupStart 的举例。

【相关主题】

HI_MPI_VO_ResumeSyncGroup

HI_MPI_VO_ResumeSyncGroup

【描述】

恢复同步输出通道组。

【语法】

```
HI_S32 HI_MPI_VO_ResumeSyncGroup(VO_GRP VoGroup);
```

【参数】

参数名称	描述	输入/输出
VoGroup	同步输出组组号。 取值范围：[0, VO_SYNC_MAX_GROUP)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_GRP_INVALID_NUMBER	同步通道组组号无效。
HI_ERR_VO_GRP_NOT_CREATE	同步通道组没有创建。
HI_ERR_VO_GRP_NOT_START	同步通道组没有启动。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前必须先启用视频输出设备，并且配置视频输出公共属性。
- 调用前必须保证视频输出通道配置并使能。
- 调用前必须保证同步通道组已经创建并启动。

【举例】

请参见 HI_MPI_VO_SyncGroupStart 的举例。

【相关主题】

HI_MPI_VO_PauseSyncGroup

HI_MPI_VO_StepSyncGroup

【描述】

单帧播放同步输出通道组。

【语法】

```
HI_S32 HI_MPI_VO_StepSyncGroup(VO_GRP VoGroup);
```

【参数】

参数名称	描述	输入/输出
VoGroup	同步输出组组号。 取值范围：[0, VO_SYNC_MAX_GROUP)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_GRP_INVALID_NUMBER	同步通道组组号无效。
HI_ERR_VO_GRP_NOT_CREATE	同步通道组没有创建。

接口返回值	含义
HI_ERR_VO_GRP_NOT_START	同步通道组没有启动。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前必须先启用视频输出设备，并且配置视频输出公共属性。
- 调用前必须保证视频输出通道配置并使能。
- 调用前必须保证同步通道组已经创建并启动。

【举例】

请参见 HI_MPI_VO_SyncGroupStart 的举例。

【相关主题】

HI_MPI_VO_PauseSyncGroup

HI_MPI_VO_SetSyncGroupBase

【描述】

设置多通道同步组的基准时间戳。

【语法】

```
HI_S32 HI_MPI_VO_SetSyncGroupBase(VO_GRP VoGroup, const VO_SYNC_BASE_S
*pstSyncBase);
```

【参数】

参数名称	描述	输入/输出
VoGroup	同步输出组组号。 取值范围：[0, VO_SYNC_MAX_GROUP)。	输入
pstSyncBase	同步基准 PTS 结构体指针。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_GRP_INVALID_NUMBER	同步通道组组号无效。
HI_ERR_VO_INVALID_CHNID	无效的 VO 通道号。
HI_ERR_VO_FAILED_NOTENABLE	视频输出设备未使能。
HI_ERR_VO_FAILED_CHNOTENABLE	视频输出通道未使能。
HI_ERR_VO_FAILED_NOTCONFIG	视频输出未配置。
HI_ERR_VO_FAILED_CHNNOTCONFIG	视频输出通道为配置。
HI_ERR_VO_GRP_NOT_CREATE	同步组未创建。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前必须先启用视频输出设备，否则将返回失败。
- 调用前必须使能视频输出通道，否则返回失败。
- 调用前必须创建同步组，否则返回失败。
- 设置时间基准可以在启动同步组前或者启动同步组后。
- 可以多次设置基准时间戳。

【举例】

请参见 HI_MPI_VO_SyncGroupStart 的举例。

【相关主题】

HI_MPI_VO_GetSyncGroupBase

HI_MPI_VO_GetSyncGroupBase

【描述】

获取多通道同步组的基准时间戳。

【语法】

```
HI_S32 HI_MPI_VO_GetSyncGroupBase(VO_GRP VoGroup, VO_SYNC_BASE_S
*pstSyncBase);
```

【参数】

参数名称	描述	输入/输出
VoGroup	同步输出组组号。 取值范围：[0, VO_SYNC_MAX_GROUP)。	输入
pstSyncBase	同步基准 PTS 结构体指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_GRP_INVALID_NUMBER	同步通道组组号无效。
HI_ERR_VO_INVALID_CHNID	无效的 VO 通道号。
HI_ERR_VO_FAILED_NOTENABLE	视频输出设备未使能。
HI_ERR_VO_FAILED_CHNOTENABLE	视频输出通道未使能。
HI_ERR_VO_FAILED_NOTCONFIG	视频输出未配置。
HI_ERR_VO_FAILED_CHNNOTCONFIG	视频输出通道未配置。
HI_ERR_VO_GRP_NOT_CREATE	同步组未创建。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前必须先启用视频输出设备，否则将返回失败。
- 调用前必须使能视频输出通道，否则返回失败。
- 调用前必须创建同步组，否则返回失败。
- 获取时间基准可以在启动同步组前或者启动同步组后。

- 可以多次获取基准时间戳。

【举例】

请参见 HI_MPI_VO_SyncGroupStart 的举例。

【相关主题】

HI_MPI_VO_SetSyncGroupBase

HI_MPI_VO_SetZoomInWindow

【描述】

设置视频输出局部放大窗口。

【语法】

```
HI_S32 HI_MPI_VO_SetZoomInWindow(VO_CHN VoChn, const RECT_S stZoomRect);
```

【参数】

参数名称	描述	输入/输出
VoChn	视频输出通道的通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入
stZoomRect	局部放大窗口矩形结构体。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	通道号无效。
HI_ERR_VO_INVALID_RECT_PARA	矩形结构体参数无效。
HI_ERR_VO_FAILED_NOTENABLE	视频输出设备未使能。
HI_ERR_VO_FAILED_NOTCONFIG	视频输出公共属性未配置。
HI_ERR_VO_FAILED_CHNOTENABLE	视频输出通道未使能。

接口返回值	含义
HI_ERR_VO_FAILED_CHNNOTCONFIG	视频输出通道属性未配置。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前必须先启用视频输出设备，并且配置视频输出公共属性。
- 调用前必须保证视频输出通道配置并使能。
- 局部放大窗口参数的起始点坐标为非负数，同时要保证该坐标是 8 象素对齐。
- 局部放大窗口的起始点和宽高都是相对于输入图像数据而言的，而不是屏幕坐标，如果需要的话，用户需要将屏幕坐标转换到输入图像坐标。

【举例】

```
/* define input parameter */
HI_S32 s32Ret;
VO_CHN VoChn = 0;
RECT_S stZoomWindow, stZoomWindowGet;

/*
 * we should enable VO and VO channel first!
 * TODO: enable operation.
 */

stZoomWindow.s32X = 128;
stZoomWindow.s32Y = 128;
stZoomWindow.u32Width = 200;
stZoomWindow.u32Height = 200;

/* set zoom window */
s32Ret = HI_MPI_VO_SetZoomInWindow(VoChn, stZoomWindow);
if (s32Ret != HI_SUCCESS)
{
    printf("Set zoom window failed, ret = %#x.\n", s32Ret);
    return HI_FAILURE;
}

/* get current zoom window parameter */
s32Ret = HI_MPI_VO_GetZoomInWindow(VoChn, &stZoomWindowGet);
if (s32Ret != HI_SUCCESS)
```

```
{
    printf("Get zoom window rect failed, ret = %#x.\n", s32Ret);
    return HI_FAILURE;
}
else
{
    printf("Current zoom window is (%d,%d,%d,%d)!\n",
        stZoomWindowGet.s32X, stZoomWindowGet.s32Y,
        stZoomWindowGet.u32Width, stZoomWindowGet.u32Height);
}

/*
 * TODO: something else ...
 */

stZoomWindow.s32X = 0;
stZoomWindow.s32Y = 0;
stZoomWindow.u32Width = 0;
stZoomWindow.u32Height = 0;

/* cancel zoom window, we use (0,0,0,0) to recover */
s32Ret = HI_MPI_VO_SetZoomInWindow(VoChn, stZoomWindow);
if (s32Ret != HI_SUCCESS)
{
    printf("Recover zoom window failed, ret = %#x.\n", s32Ret);
    return HI_FAILURE;
}
```

【相关主题】

HI_MPI_VO_GetZoomInWindow

HI_MPI_VO_GetZoomInWindow

【描述】

设置视频输出局部放大窗口。

【语法】

```
HI_S32 HI_MPI_VO_GetZoomInWindow(VO_CHN VoChn, RECT_S *pstZoomRect);
```

【参数】

参数名称	描述	输入/输出
VoChn	视频输出通道的通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入

参数名称	描述	输入/输出
pstZoomRect	局部放大窗口矩形结构体指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	通道号无效。
HI_ERR_VO_FAILED_NOTENABLE	视频输出设备未使能。
HI_ERR_VO_FAILED_NOTCONFIG	视频输出公共属性未配置。
HI_ERR_VO_FAILED_CHNOTENABLE	视频输出通道未使能。
HI_ERR_VO_FAILED_CHNNOTCONFIG	视频输出通道属性未配置。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前必须先启用视频输出设备，并且配置视频输出公共属性。
- 调用前必须保证视频输出通道配置并使能。
- 如果没有设置局部放大窗口，那么默认获取到（0, 0, 0, 0）。

【举例】

请参见 HI_MPI_VO_SetZoomInWindow 的举例。

【相关主题】

HI_MPI_VO_SetZoomInWindow

HI_MPI_VO_GetChnPts

【描述】

获取指定视频输出通道当前显示图像的时间戳。

【语法】

```
HI_S32 HI_MPI_VO_GetChnPts(VO_CHN VoChn, HI_U64 *pu64VoChnPts);
```

【参数】

参数名称	描述	输入/输出
VoChn	视频输出通道的通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入
pu64VoChnPts	通道时间戳指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VO_NULL_PTR	参数中有空指针。
HI_ERR_VO_FAILED_NOTENABLE	视频输出设备未使能。
HI_ERR_VO_FAILED_CHNOTENABLE	视频输出通道未使能。
HI_ERR_VO_FAILED_NOTCONFIG	视频输出未配置。
HI_ERR_VO_FAILED_CHNNOTCONFIG	视频输出通道为配置。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前需保证视频设备使能，视频通道使能。
- 可以重复调用获取时间戳接口。

【举例】

请参见 HI_MPI_VO_Enable 的举例。

【相关主题】

无。

HI_MPI_VO_SetAttrBegin

【描述】

设置属性开始。

【语法】

```
HI_S32 HI_MPI_VO_SetAttrBegin(HI_VOID);
```

【参数】

无。

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_FAILED_NOTENABLE	视频输出设备未使能。
HI_ERR_VO_FAILED_NOTCONFIG	视频输出未配置。
HI_ERR_VO_SETBEGIN_ALREADY	视频输出设置属性已经开始。
HI_ERR_VO_SETEND_NOTYET	视频输出设置属性还没有完成。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前需保证视频设备使能。
- 调用前需要保证所要设置的通道已经使能。
- BEGIN 和 END 接口要配对使用，否则 BEGIN 后设置的通道属性不会生效。

【举例】

```
HI_S32 s32ret;
HI_U32 i;
VO_PUB_ATTR_S VoAttr;
VO_CHN_ATTR_S VoChnAttr[4];

memset(&VoAttr, 0, sizeof(VO_PUB_ATTR_S));
VoAttr.stTvConfig.stComposeMode = VIDEO_ENCODING_MODE_PAL;  /*PAL Mode*/
VoAttr.u32BgColor = 0x0;

/* set public attribute of vo device */
s32ret = HI_MPI_VO_SetPubAttr(&VoAttr);
if (HI_SUCCESS != s32ret)
{
    printf("vo set pub attr failed! \n");
    return s32ret;
}

/* enable vo device */
s32ret = HI_MPI_VO_Enable();
if (HI_SUCCESS != s32ret)
{
    printf("enable vo device failed! \n");
    return s32ret;
}

/* configure and enable vo channel                                    */
for (i = 0; i < 4; i++)
{
   VoChnAttr[i].bZoomEnable = HI_TRUE;
   VoChnAttr[i].u32Priority = 1;
   VoChnAttr[i].stRect.s32X = 360*(i%2);
   VoChnAttr[i].stRect.s32Y = 288*(i/2);
   VoChnAttr[i].stRect.u32Width = 360;
   VoChnAttr[i].stRect.u32Height = 288;

   /* set attribute of vo chn*/
   s32ret = HI_MPI_VO_SetChnAttr(i, &VoChnAttr[i]);
```

```
    if (HI_SUCCESS != s32ret)
    {
    printf("vo set chn %d attr failed! \n", i);
    return s32ret;
    }

    /* enable vo chn */
    s32ret = HI_MPI_VO_EnableChn(i);
    if (HI_SUCCESS != s32ret)
    {
    printf("enable vo chn %d failed! \n", i);
    return s32ret;
    }
}

/* do somethng else                                                    */
sleep(20);

/* set channel attributes begin                                         */
s32ret = HI_MPI_VO_SetAttrBegin();
if (HI_SUCCESS != s32ret)
{
    printf("set attributes begin failed!\n");
    return s32ret;
}

/* reset channel attributes
 * we just show channel 0 and 1 on screen, and scale them as vertical
 * half D1
 */
for (i = 0; i < 2; i++)
{
    VoChnAttr[i].bZoomEnable = HI_TRUE;
    VoChnAttr[i].u32Priority = 1;
    VoChnAttr[i].stRect.s32X = 352*(i%2);
    VoChnAttr[i].stRect.s32Y = 288*(i/2);
    VoChnAttr[i].stRect.u32Width = 360;
    VoChnAttr[i].stRect.u32Height = 576;

    /* set attribute of vo chn*/
    s32ret = HI_MPI_VO_SetChnAttr(i, &VoChnAttr[i]);
    if (HI_SUCCESS != s32ret)
    {
    printf("vo set chn %d attr failed! \n", i);
```

```
    return s32ret;
    }
}

/* hide channel 2 and 3
 * we do this just for saving bandwidth
 */
for (i = 2; i < 4; i++)
{
    s32ret = HI_MPI_VO_ChnHide(i);
    if (s32ret != HI_SUCCESS)
    {
        printf("vo hide channel %d failed!\n", i);
        return HI_FAILURE;
    }
}

/* set channel attributes end                                          */
s32ret = HI_MPI_VO_SetAttrEnd();
if (HI_SUCCESS != s32ret)
{
    printf("set attributes end failed!\n");
    return s32ret;
}

/* do somethng else                                                  */
sleep(20);

/* set channel attributes begin                                        */
s32ret = HI_MPI_VO_SetAttrBegin();
if (HI_SUCCESS != s32ret)
{
    printf("set attributes begin failed!\n");
    return s32ret;
}

/* reset channel attributes
 * now we set them back as 4 cif on screen
 */
for (i = 0; i < 4; i++)
{
    VoChnAttr[i].bZoomEnable = HI_TRUE;
    VoChnAttr[i].u32Priority = 1;
    VoChnAttr[i].stRect.s32X = 360*(i%2);
```

```
    VoChnAttr[i].stRect.s32Y = 288*(i/2);
    VoChnAttr[i].stRect.u32Width = 360;
    VoChnAttr[i].stRect.u32Height = 288;

    /* set attribute of vo chn*/
    s32ret = HI_MPI_VO_SetChnAttr(i, &VoChnAttr[i]);
    if (HI_SUCCESS != s32ret)
    {
    printf("vo set chn %d attr failed! \n", i);
    return s32ret;
    }
}

/* show channel 2 and 3                                          */
for (i = 2; i < 4; i++)
{
    s32ret = HI_MPI_VO_ChnShow(i);
    if (s32ret != HI_SUCCESS)
    {
        printf("vo show channel %d failed!\n", i);
        return HI_FAILURE;
    }
}

/* set channel attributes end                                     */
s32ret = HI_MPI_VO_SetAttrEnd();
if (HI_SUCCESS != s32ret)
{
    printf("set attributes end failed!\n");
    return s32ret;
}

/* do somethng else                                              */
sleep(20);

/* disable all channels                                          */
for (i = 0; i < 4; i++)
{
    s32ret = HI_MPI_VO_DisableChn(i);
    if (HI_SUCCESS != s32ret)
    {
        printf("vo disable chn %d failed!\n", i);
        return s32ret;
    }
```

```
}

/* disable vou                                                    */
(void)HI_MPI_VO_Disable();
```

【相关主题】

HI_MPI_VO_SetAttrEnd

HI_MPI_VO_SetAttrEnd

【描述】

设置属性结束。

【语法】

```
HI_S32 HI_MPI_VO_SetAttrEnd(HI_VOID);
```

【参数】

无。

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_FAILED_NOTENABLE	视频输出设备未使能。
HI_ERR_VO_FAILED_NOTCONFIG	视频输出未配置。
HI_ERR_VO_SETBEGIN_NOTYET	视频输出设置属性没有开始。
HI_ERR_VO_SETEND_ALREADY	视频输出设置属性未完成。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前需保证视频设备使能。

- 调用前需要保证所要设置的通道已经使能。
- BEGIN 和 END 接口要配对使用，否则 BEGIN 后设置的通道属性不会生效。

【举例】

请参见 HI_MPI_VO_SetAttrBegin 的举例。

【相关主题】

HI_MPI_VO_SetAttrBegin

HI_MPI_VO_ChnShow

【描述】

显示指定通道。

【语法】

```
HI_S32 HI_MPI_VO_ChnShow(VO_CHN VoChn);
```

【参数】

参数名称	描述	输入/输出
VoChn	视频输出通道的通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VO_FAILED_NOTENABLE	视频输出设备未使能。
HI_ERR_VO_FAILED_CHNOTENABLE	视频输出通道未使能。
HI_ERR_VO_FAILED_NOTCONFIG	视频输出未配置。
HI_ERR_VO_FAILED_CHNNOTCONFIG	视频输出通道未配置。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前需保证视频设备使能。
- 调用前需要保证所要设置的通道已经使能。
- 可以多次调用显示或者隐藏通道接口。

【举例】

请参见 HI_MPI_VO_SetAttrBegin 的举例。

【相关主题】

HI_MPI_VO_ChnHide

HI_MPI_VO_ChnHide

【描述】

隐藏指定通道。

【语法】

```
HI_S32 HI_MPI_VO_ChnHide(VO_CHN VoChn);
```

【参数】

参数名称	描述	输入/输出
VoChn	视频输出通道的通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。

接口返回值	含义
HI_ERR_VO_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VO_FAILED_NOTENABLE	视频输出设备未使能。
HI_ERR_VO_FAILED_CHNOTENABLE	视频输出通道未使能。
HI_ERR_VO_FAILED_NOTCONFIG	视频输出未配置。
HI_ERR_VO_FAILED_CHNNOTCONFIG	视频输出通道未配置。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前需保证视频设备使能。
- 调用前需要保证所要设置的通道已经使能。
- 可以多次调用显示或者隐藏通道接口。

【举例】

请参见 HI_MPI_VO_SetAttrBegin 的举例。

【相关主题】

HI_MPI_VO_ChnShow

HI_MPI_VO_Query

【描述】

查询视频输出通道状态。

【语法】

```
HI_S32 HI_MPI_VO_Query (VO_CHN VoChn, VO_QUERY_STATUS_S *pstQueryStatus);
```

【参数】

参数名称	描述	输入/输出
VoChn	视频输出通道的通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入
pstQueryStatus	通道状态结构体指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VO_FAILED_NOTENABLE	视频输出设备未使能。
HI_ERR_VO_FAILED_CHNOTENABLE	视频输出通道未使能。
HI_ERR_VO_FAILED_NOTCONFIG	视频输出未配置。
HI_ERR_VO_FAILED_CHNNOTCONFIG	视频输出通道未配置。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前需保证视频设备使能。
- 调用前需要保证所要设置的通道已经使能。
- 可以多次调用获取通道状态接口。

【举例】

```
HI_S32 s32Ret;
VO_CHN VoChn = 0;
VO_QUERY_STATUS_S stVoQueryStatus;

/* enable vo device and vo channel */
…

s32Ret = HI_MPI_VO_Query(VoChn, & stVoQueryStatus);
if (HI_SUCCESS != s32Ret)
{
    printf("Query channel status failed with errorcode %#x!\n", s32Ret);
    return HI_FAILURE;
}
```

```
printf("Current channel %d has %d VB occupied!\n",
            VoChn, stVoQueryStatus .u32ChnBufUsed );
```

【相关主题】

无。

HI_MPI_VO_SetDisplayFrameRate

【描述】

设置屏幕显示低帧率。

【语法】

```
HI_S32 HI_MPI_VO_SetDisplayFrameRate(HI_S32 s32VoFramerate);
```

【参数】

参数名称	描述	输入/输出
s32VoFramerate	视频输出显示帧率。 取值范围：[0, 25]。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VO_FAILED_NOTENABLE	视频输出设备未使能。
HI_ERR_VO_FAILED_CHNOTENABLE	视频输出通道未使能。
HI_ERR_VO_ILLEGAL_PARAM	非法帧率参数。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h

- 库文件：libmpi.a

【注意】

- 调用前需保证视频设备使能。
- 该接口主要用于视频解码显示时的低帧率策略（在系统资源紧张的情况下节省性能），不推荐预览时使用。

【举例】

```
HI_S32 s32Ret;
HI_U32 frameRate = 5;
HI_U32 u32VoFramerate = 0;

/* enable vo device and vo channel */
…

/* start decode */
…

/* set vo display frame rate */
s32Ret = HI_MPI_VO_SetDisplayFrameRate(frameRate);
if (HI_SUCCESS != s32Ret)
{
    printf("set display frame rate failed at %#x\n", s32Ret);
    return HI_FAILURE;
}

/* get vo display frame rate */
s32Ret = HI_MPI_VO_GetDisplayFrameRate(&u32VoFramerate);
if (HI_SUCCESS != s32Ret)
{
    printf("get display frame rate failed at %#x\n", s32Ret);
    return HI_FAILURE;
}
printf("current vo display speed is: %d\n", u32VoFramerate);

/* stop decode and disable vo channel and vo device */
…
```

【相关主题】

HI_MPI_VO_GetDisplayFrameRate

HI_MPI_VO_GetDisplayFrameRate

【描述】

获取屏幕显示低帧率。

【语法】

```
HI_S32 HI_MPI_VO_GetDisplayFrameRate(HI_S32 *ps32VoFramerate);
```

【参数】

参数名称	描述	输入/输出
ps32VoFramerate	视频输出显示帧率指针。 取值范围：[0, 25]。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VO_FAILED_NOTENABLE	视频输出设备未使能。
HI_ERR_VO_FAILED_CHNOTENABLE	视频输出通道未使能。
HI_ERR_VO_NULL_PTR	视频输出显示帧率指针为空。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前需保证视频设备使能。
- 该接口主要用于视频解码显示时的低帧率策略（在系统资源紧张的情况下节省性能），不推荐预览时使用。

【举例】

请参见 HI_MPI_VO_SetAttrBegin 的举例。

【相关主题】

HI_MPI_VO_SetDisplayFrameRate

HI_MPI_VO_GetChnFrame

【描述】

获取输出通道图像数据。

【语法】

```
HI_S32 HI_MPI_VO_GetChnFrame(VO_CHN VoChn, VIDEO_FRAME_INFO_S *pstFrame);
```

【参数】

参数名称	描述	输入/输出
VoChn	视频输出通道的通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入
pstFrame	获取的输出通道图像数据信息结构体指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VO_FAILED_NOTENABLE	视频输出设备未使能。
HI_ERR_VO_FAILED_CHNOTENABLE	视频输出通道未使能。
HI_ERR_VO_FAILED_NOTCONFIG	视频输出未配置。
HI_ERR_VO_FAILED_CHNNOTCONFIG	视频输出通道未配置。
HI_ERR_VO_ILLEGAL_PARAM	视频输出参数无效。
HI_ERR_VO_FAILED_TIMEOUT	获取图像数据超时。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前需保证视频设备使能。
- 调用前需要保证所要设置的通道已经使能。
- 可以多次调用，获取后应保证及时的释放。

【举例】

```
...
VIDEO_FRAME_INFO_S *pstFrame =
(VIDEO_FRAME_INFO_S*)malloc(sizeof(VIDEO_FRAME_INFO_S));

/* enable vo chn */
s32ret = HI_MPI_VO_EnableChn(VoChn);
if (HI_SUCCESS != s32ret)
{
    printf("enable vo chn failed! \n");
    return s32ret;
}

if (HI_SUCCESS != HI_MPI_VO_GetScreenFrame(pstFrame))
{
   printf("Get screen frame failed!\n");
   return HI_FAILURE;
}

/* do some process, i.e. store file or do JPEG encodeing ... */

if (HI_SUCCESS != HI_MPI_VO_ReleaseScreenFrame(pstFrame))
{
   printf("Release screen frame failed!\n");
   return HI_FAILURE;
}

if (HI_SUCCESS != HI_MPI_VO_GetChnFrame(VoChn, pstFrame))
{
   printf("Get channel frame failed!\n");
   return HI_FAILURE;
}

/* do some process, i.e. store file or do JPEG encodeing ... */

if (HI_SUCCESS != HI_MPI_VO_ReleaseChnFrame(VoChn, pstFrame))
```

```
{
    printf("Release screen frame failed!\n");
    return HI_FAILURE;
}

(void)HI_MPI_VO_DisableChn(VoChn);
```

【相关主题】

HI_MPI_VO_ReleaseChnFrame

HI_MPI_VO_ReleaseChnFrame

【描述】

释放输出通道图像数据。

【语法】

```
HI_S32 HI_MPI_VO_ReleaseChnFrame(VO_CHN VoChn, VIDEO_FRAME_INFO_S
*pstFrame);
```

【参数】

参数名称	描述	输入/输出
VoChn	视频输出通道的通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入
pstFrame	释放的输出通道图像数据信息结构体指针。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VO_FAILED_NOTENABLE	视频输出设备未使能。
HI_ERR_VO_FAILED_CHNOTENABLE	视频输出通道未使能。

接口返回值	含义
HI_ERR_VO_FAILED_NOTCONFIG	视频输出未配置。
HI_ERR_VO_FAILED_CHNNOTCONFIG	视频输出通道未配置。
HI_ERR_VO_ILLEGAL_PARAM	视频输出参数无效。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前需保证视频设备使能。
- 调用前需要保证所要设置的通道已经使能。
- 可以多次调用，获取操作应保证与释放操作配对。

【举例】

请参见 HI_MPI_VO_GetChnFrame 的举例。

【相关主题】

HI_MPI_VO_GetChnFrame

HI_MPI_VO_GetScreenFrame

【描述】

获取输出屏幕图像数据。

【语法】

```
HI_S32 HI_MPI_VO_GetScreenFrame(VIDEO_FRAME_INFO_S *pstFrame);
```

【参数】

参数名称	描述	输入/输出
pstFrame	获取的输出屏幕图像数据信息结构体指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VO_FAILED_NOTENABLE	视频输出设备未使能。
HI_ERR_VO_FAILED_CHNOTENABLE	视频输出通道未使能。
HI_ERR_VO_FAILED_NOTCONFIG	视频输出未配置。
HI_ERR_VO_FAILED_CHNNOTCONFIG	视频输出通道未配置。
HI_ERR_VO_ILLEGAL_PARAM	视频输出参数无效。
HI_ERR_VO_FAILED_TIMEOUT	获取图像数据超时。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前需保证视频设备使能。
- 调用前需要保证所要设置的通道已经使能。
- 可以多次调用，获取后应保证及时的释放。

【举例】

请参见 HI_MPI_VO_GetChnFrame 的举例。

【相关主题】

HI_MPI_VO_ReleaseScreenFrame

HI_MPI_VO_ReleaseScreenFrame

【描述】

释放输出屏幕图像数据。

【语法】

```
HI_S32 HI_MPI_VO_ReleaseScreenFrame(VIDEO_FRAME_INFO_S *pstFrame);
```

【参数】

参数名称	描述	输入/输出
pstFrame	释放的输出屏幕图像数据信息结构体指针。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VO_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VO_FAILED_NOTENABLE	视频输出设备未使能。
HI_ERR_VO_FAILED_CHNOTENABLE	视频输出通道未使能。
HI_ERR_VO_FAILED_NOTCONFIG	视频输出未配置。
HI_ERR_VO_FAILED_CHNNOTCONFIG	视频输出通道未配置。
HI_ERR_VO_ILLEGAL_PARAM	视频输出参数无效。

【需求】

- 头文件：mpi_vo.h、hi_comm_vo.h
- 库文件：libmpi.a

【注意】

- 调用前需保证视频设备使能。
- 调用前需要保证所要设置的通道已经使能。
- 可以多次调用，获取操作应保证与释放操作配对。

【举例】

请参见 HI_MPI_VO_GetChnFrame 的举例。

【相关主题】

HI_MPI_VO_GetScreenFrame

2.2.4 视频前处理

视频前处理模块对用户输入或 VI 捕获的图像所作的处理都在编码前进行。视频前处理模块主要提供设置和获取视频前处理配置、创建和销毁 VPP 区域、控制 VPP 区域、创建和等待图像缩放任务完成功能。

该功能模块提供以下 MPI：

- HI_MPI_VPP_SetConf：设置视频前处理配置。
- HI_MPI_VPP_GetConf：获取视频前处理配置。
- HI_MPI_VPP_CreateRegion：创建 VPP 区域。
- HI_MPI_VPP_DestroyRegion：销毁 VPP 区域。
- HI_MPI_VPP_ControlRegion：控制 VPP 区域。
- HI_MPI_VPP_CreateScaleTask：创建一个图像缩放任务。
- HI_MPI_VPP_WaitScaleTask：等待一个图像缩放任务完成。

HI_MPI_VPP_SetConf

【描述】

设置视频前处理配置。

【语法】

```
HI_S32 HI_MPI_VPP_SetConf(VENC_GRP VeGroup, const VIDEO_PREPROC_CONF_S
*pstConf);
```

【参数】

参数名称	描述	输入/输出
VeGroup	编码通道组号。 取值范围：[0, VENC_MAX_GRP_NUM)。	输入
pstConf	视频前处理属性指针。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VPP_INVALID_CHNID	通道号错误，此错误通常是由于通道组号超出范围导致。
HI_ERR_VPP_NULL_PTR	空指针。
HI_ERR_VPP_BUSY	系统忙。
HI_ERR_VPP_ILLEGAL_PARAM	非法参数。
HI_ERR_VPP_NOTREADY	系统没有初始化或没有加载相应模块（指各个 MPI 对应的内核模块）。

【需求】

- 头文件：hi_comm_vpp.h、mpi_vpp.h
- 库文件：libmpi.a

【注意】

- 视频前处理的的各项配置都有默认值，一般情况下不需要用户再进行设置，目前的默认配置为：彩色转灰色关闭、时域去噪打开、滤波系数为默认系数、缩放方式为采用保留底场方式缩放。
- 如果需要更改默认配置，应该先调用 HI_MPI_VPP_GetConf 接口获取配置，更改需要配置的部分选项后，再调用此接口进行设置。
- 此接口可动态调用，即在编码时也可以调用此接口。

【举例】

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <sys/poll.h>
#include <fcntl.h>
#include <errno.h>
#include <pthread.h>
#include <math.h>
#include <unistd.h>

#include "hi_common.h"
#include "hi_comm_vpp.h"
#include "mpi_vpp.h"
```

```
HI_S32 VppSetAndGetConf(HI_VOID)
{
    HI_S32 s32Ret;
    VENC_GRP VeGroup = 0;
    VIDEO_PREPROC_CONF_S stConf;

    /* first get current vpp config */
    s32Ret = HI_MPI_VPP_GetConf(VeGroup, &stConf);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VPP_GetConf failed 0x%x!\n",s32Ret);
        return HI_FAILURE;
    }

    /* modify some config you need */
    stConf.bColorToGrey = HI_TRUE;
    stConf.bTemporalDenoise = HI_TRUE;
    stConf.enScaleMode = VPP_SCALE_MODE_USEBOTTOM;
    stConf.enFilter = VPP_SCALE_FILTER_DEFAULT;

    s32Ret = HI_MPI_VPP_SetConf(VeGroup, &stConf);
    if (HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VPP_SetConf failed 0x%x!\n",s32Ret);
        return HI_FAILURE;
    }
    return HI_SUCCESS;

}
```

【相关主题】

无。

HI_MPI_VPP_GetConf

【描述】

获取视频前处理配置。

【语法】

```
HI_S32 HI_MPI_VPP_GetConf(VENC_GRP VeGroup,
VIDEO_PREPROC_CONF_S *pstConf);
```

【参数】

参数名称	描述	输入/输出
VeGroup	编码通道组号。 取值范围：[0, VENC_MAX_GRP_NUM)。	输入
pstConf	视频前处理属性指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VPP_INVALID_CHNID	通道组号错误。
HI_ERR_VPP_NULL_PTR	空指针。
HI_ERR_VPP_BUSY	系统忙。
HI_ERR_VPP_NOTREADY	系统没有初始化或没有加载相应模块。

【需求】

- 头文件：hi_comm_vpp.h、mpi_vpp.h
- 库文件：libmpi.a

【注意】

- 如果不设置视频前处理属性，则返回系统默认的前处理配置，不会返回失败。
- 视频前处理的默认配置为：彩色转灰色关闭、时域去噪打开、滤波系数为默认系数、缩放方式为采用保留底场方式缩放。

【举例】

请参见 HI_MPI_VPP_SetConf 的举例。

【相关主题】

无。

HI_MPI_VPP_CreateRegion

【描述】

创建 VPP 区域。

【语法】

```
HI_S32 HI_MPI_VPP_CreateRegion(const REGION_ATTR_S *pstRegion,
REGION_HANDLE *pHandle);
```

【参数】

参数名称	描述	输入/输出
pstRegion	区域属性。	输入
pHandle	区域句柄指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VPP_INVALID_DEVID	错误的设备号。
HI_ERR_VPP_INVALID_CHNID	错误的通道号。
HI_ERR_VPP_NULL_PTR	空指针。
HI_ERR_VPP_BUSY	系统忙。
HI_ERR_VPP_NOMEM	分配内存失败。
HI_ERR_VPP_ILLEGAL_PARAM	非法参数。
HI_ERR_VPP_NOT_SUPPORT	不支持（若用户输入的区域类型既不是遮挡区域类型，又不是叠加区域类型，则返回不支持的错误码）。
HI_ERR_VPP_NOTREADY	系统没有初始化或没有加载相应模块。

【需求】

- 头文件：hi_comm_vpp.h、mpi_vpp.h
- 库文件：libmpi.a

【注意】

- VPP 区域创建后，默认是隐藏的，必须主动调用显示接口才会显示。
- 视频遮挡区域是在 VI 的原始视频图像叠加的单色填充块，在与此 VI 通道绑定的编码通道组以及视频输出中都会有相应的视频遮挡显示；视频遮挡的位置和大小是相对于 VI 原始视频图像（即未做丢场和水平压缩前的输入图像）而言。
- 视频叠加区域是视频编码前叠加在视频上的“前端 OSD”，在视频编码码流中显示，但在视频预览中不会显示；视频叠加区域针对的是整个编码通道组，由于同一编码通道组中的小码流由大码流缩放得到，因此小码流中显示的叠加区域的位置和大小也是等比例缩放。
- 马赛克遮挡区域是一种特殊的视频遮挡区域，其使用 TDE 功能将位图叠加在 VI 视频上，所以仅用于不使用 TDE 的板卡类似应用中，否则会有性能问题。
- 软件视频叠加区域，主要用于 PCI 级联业务中的对端预览 OSD，将位图叠加到 VI 视频上再通过 PCI 传输到对端，对本地的码流和预览图像无效。
- 表 2-1 为遮挡区域与叠加区域的对比。

表2-1 各种 VPP 区域的对比

属性	视频遮挡 COVER	视频叠加 OVERLAY
相对通道	VI 通道	Group 通道
区域	起始点坐标必须大于等于 0，高宽必须大于 0，最大为 4095 % 4095。	起始点坐标必须大于等于 0 且为偶数，X 坐标必须为 4 的倍数。高、宽都必须为偶数，以像素为单位，最大为 2047 % 2047。
公共区域	支持公共区域，即如果设置为公共区域，所有通道使用其相同属性。	支持公共区域，即如果设置为公共区域，所有通道使用其相同属性。
数量限制	包括公共区域在内，每个通道最多 4 个。	包括公共区域在内，每个通道最多 4 个。
像素格式	不支持设置像素格式。	即为背景色和填充位图的像素格式，支持 αRGB1555、αRGB4444 两种格式，同一通道的各区域像素格式必须一致。

属性	视频遮挡 COVER	视频叠加 OVERLAY
颜色	视频遮挡区域仅支持单色填充。颜色值采用 RGB888 格式，取值范围为 0x00000000～0x00FFFFFF，低 24 位有效，低 8 位为 R 值，中间 8 位为 G 值，高 8 位为 B 值。	对于视频叠加，即为背景色。两种格式的取值范围为 0x00000000～0x0000FFFF，低 16 位有效。对 αRGB1555 格式，最高 1 位为 α 位，次高 5 位为 R 值，中间 5 位为 G 值，低 5 位为 B 值；对 αRGB4444 格式，最高 4 位为 α 位，次高 4 位为 R 值，中间 4 位为 G 值，低 4 位为 B 值。
透明度	不支持设置透明度。	可设置前景透明度和背景透明度。透明度取值范围为 0～128，值越小表示越透明。0 表示完全透明，128 表示完全不透明。单色填充叠加区域时，透明度取背景透明度；αRGB1555 格式位图填充时，alpha 位为 1 的点取前景透明度，alpha 位为 0 的点取背景透明度；αRGB4444 格式位图填充时，透明度由位图像素点的前 4 位 α 值决定。
层次	支持范围为 0～100 的层次设置，通道内层次不同的区域间可以重叠。	不支持设置层次，且通道内各区域间不能有位置重叠。
位图填充	不支持填充位图。	可填充 ARGB1555 或者 ARGB4444 格式的位图。

【举例】

本举例共有两个用例，一个遮挡区域用例，一个叠加区域用例。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <sys/poll.h>
#include <fcntl.h>
#include <errno.h>
#include <pthread.h>
#include <math.h>
#include <unistd.h>
```

```
#include "hi_common.h"
#include "hi_comm_vpp.h"
#include "mpi_vpp.h"
#include "loadbmp.h"

#define VIDEVID 0
#define VICHNID 0
#define VOCHNID 0

/*cover region sample*/
HI_S32 VppCtrlCoverRegion(HI_VOID)
{
    HI_S32 i;
    HI_S32 s32Ret;
    HI_S32 s32Cnt = 0;
    REGION_CRTL_CODE_E enCtrl;
    REGION_CTRL_PARAM_U unParam;
    REGION_ATTR_S stRgnAttr;
    REGION_HANDLE handle[5];

    stRgnAttr.enType = COVER_REGION;
    stRgnAttr.unAttr.stCover.bIsPublic = HI_FALSE;
    stRgnAttr.unAttr.stCover.u32Color  = 0;
    stRgnAttr.unAttr.stCover.u32Layer  = 1;
    stRgnAttr.unAttr.stCover.stRect.s32X = 100;
    stRgnAttr.unAttr.stCover.stRect.s32Y = 200;
    stRgnAttr.unAttr.stCover.stRect.u32Height = 50;
    stRgnAttr.unAttr.stCover.stRect.u32Width  = 50;
    stRgnAttr.unAttr.stCover.ViChn = VICHNID;
    stRgnAttr.unAttr.stCover.ViDevId = VIDEVID;

    s32Ret = HI_MPI_VPP_CreateRegion(&stRgnAttr, &handle[0]);
    if(s32Ret != HI_SUCCESS)
    {
        printf("HI_MPI_VPP_CreateRegion err 0x%x\n",s32Ret);
        return HI_FAILURE;
    }

    stRgnAttr.enType = COVER_REGION;
    stRgnAttr.unAttr.stCover.bIsPublic = HI_FALSE;
    stRgnAttr.unAttr.stCover.u32Color  = 0x0000ff00;
    stRgnAttr.unAttr.stCover.u32Layer  = 2;
    stRgnAttr.unAttr.stCover.stRect.s32X = 200;
    stRgnAttr.unAttr.stCover.stRect.s32Y = 200;
```

```
stRgnAttr.unAttr.stCover.stRect.u32Height = 50;
stRgnAttr.unAttr.stCover.stRect.u32Width  = 50;
stRgnAttr.unAttr.stCover.ViChn = VICHNID;
stRgnAttr.unAttr.stCover.ViDevId = VIDEVID;

s32Ret = HI_MPI_VPP_CreateRegion(&stRgnAttr, &handle[1]);
if(s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VPP_CreateRegion err 0x%x\n",s32Ret);
    return HI_FAILURE;
}

/*public cover region*/
stRgnAttr.enType = COVER_REGION;
stRgnAttr.unAttr.stCover.bIsPublic = HI_TRUE;
stRgnAttr.unAttr.stCover.u32Color  = 0x00ff0000;
stRgnAttr.unAttr.stCover.u32Layer  = 3;
stRgnAttr.unAttr.stCover.stRect.s32X = 300;
stRgnAttr.unAttr.stCover.stRect.s32Y = 200;
stRgnAttr.unAttr.stCover.stRect.u32Height = 50;
stRgnAttr.unAttr.stCover.stRect.u32Width  = 50;

s32Ret = HI_MPI_VPP_CreateRegion(&stRgnAttr, &handle[2]);
if(s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VPP_CreateRegion err 0x%x\n",s32Ret);
    return HI_FAILURE;
}

stRgnAttr.enType = COVER_REGION;
stRgnAttr.unAttr.stCover.bIsPublic = HI_FALSE;
stRgnAttr.unAttr.stCover.u32Color  = 0x00ff;
stRgnAttr.unAttr.stCover.u32Layer  = 4;
stRgnAttr.unAttr.stCover.stRect.s32X = 400;
stRgnAttr.unAttr.stCover.stRect.s32Y = 200;
stRgnAttr.unAttr.stCover.stRect.u32Height = 50;
stRgnAttr.unAttr.stCover.stRect.u32Width  = 50;
stRgnAttr.unAttr.stCover.ViChn = VICHNID;
stRgnAttr.unAttr.stCover.ViDevId = VIDEVID;

s32Ret = HI_MPI_VPP_CreateRegion(&stRgnAttr, &handle[3]);
if(s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VPP_CreateRegion err 0x%x\n",s32Ret);
```

```
    return HI_FAILURE;
}

/*show region*/
enCtrl = REGION_SHOW;
s32Ret = HI_MPI_VPP_ControlRegion(handle[0],enCtrl,&unParam);
if(s32Ret != HI_SUCCESS)
{
    printf("show faild 0x%x!!!\n",s32Ret);
    return HI_FAILURE;
}

s32Ret = HI_MPI_VPP_ControlRegion(handle[1],enCtrl,&unParam);
if(s32Ret != HI_SUCCESS)
{
    printf("show faild 0x%x!!!\n",s32Ret);
    return HI_FAILURE;
}

s32Ret = HI_MPI_VPP_ControlRegion(handle[2],enCtrl,&unParam);
if(s32Ret != HI_SUCCESS)
{
    printf("show faild 0x%x!!!\n",s32Ret);
    return HI_FAILURE;
}

s32Ret = HI_MPI_VPP_ControlRegion(handle[3],enCtrl,&unParam);
if(s32Ret != HI_SUCCESS)
{
    printf("show faild 0x%x!!!\n",s32Ret);
    return HI_FAILURE;
}
/*ctrl the region*/
while(1)
{
    sleep(1);
    s32Cnt++;

    if(s32Cnt <= 10)
    {
        /*change color*/
        enCtrl = REGION_SET_COLOR;

        if(0 == s32Cnt % 2)
```

```
        {
            unParam.u32Color = 0;
        }
        else
        {
            unParam.u32Color = 0xff;
        }

       s32Ret = HI_MPI_VPP_ControlRegion(handle[0], enCtrl, &unParam);

        if(s32Ret != HI_SUCCESS)
        {
            printf("set region position faild 0x%x!!!\n",s32Ret);
            return HI_FAILURE;
        }
    }
    else if(s32Cnt <= 20)
    {
        /*change position*/
        enCtrl = REGION_SET_POSTION;
        unParam.stPoint.s32X = 200 + ((s32Cnt - 10) * 2);
        unParam.stPoint.s32Y = 200 + ((s32Cnt - 10) * 5);

        s32Ret = HI_MPI_VPP_ControlRegion(handle[1] ,enCtrl,&unParam);

        if(s32Ret != HI_SUCCESS)
        {
            printf("set region position faild 0x%x!!!\n",s32Ret);
            return HI_FAILURE;
        }
    }
    else if(s32Cnt <= 30)
    {
        /*change size*/
        enCtrl = REGION_SET_SIZE;
        unParam.stDimension.s32Height = 50 + ((s32Cnt - 20) * 8);
        unParam.stDimension.s32Width  = 50 + ((s32Cnt - 20) * 8);

        s32Ret = HI_MPI_VPP_ControlRegion(handle[2] ,enCtrl,&unParam);

        if(s32Ret != HI_SUCCESS)
        {
            printf("set region position faild 0x%x!!!\n",s32Ret);
            return HI_FAILURE;
```

```
            }
        }
        else if(s32Cnt <= 40)
        {
            /*change layer*/
            enCtrl = REGION_SET_LAYER;
            unParam.u32Layer = s32Cnt;

            if(0 == s32Cnt % 2)
            {
                handle[4] = handle[3];
            }
            else
            {
                handle[4] = handle[2];
            }

           s32Ret = HI_MPI_VPP_ControlRegion(handle[4] ,enCtrl, &unParam);

            if(s32Ret != HI_SUCCESS)
            {
                printf("set region position faild 0x%x!!!\n",s32Ret);
                return HI_FAILURE;
            }
        }
        else
        {
            for(i=0; i<4; i++)
            {
                s32Ret = HI_MPI_VPP_DestroyRegion(handle[i]);
        if(s32Ret != HI_SUCCESS)
                {
                    printf("HI_MPI_VPP_DestroyRegion err 0x%x!\n",s32Ret);
                    return HI_FAILURE;
                }
            }

            break;
        }

    }

    return HI_SUCCESS;
}
```

```
/*overlay region sample*/
HI_S32 VppCtrlOverlayRegion(HI_VOID)
{
    HI_S32 i = 0;
    HI_S32 s32Ret;
    HI_S32 s32Cnt = 0;
    char *pFilename;

    REGION_ATTR_S stRgnAttr;
    REGION_CRTL_CODE_E enCtrl;
    REGION_CTRL_PARAM_U unParam;
    REGION_HANDLE handle[4];

    stRgnAttr.enType = OVERLAY_REGION;
    stRgnAttr.unAttr.stOverlay.bPublic = HI_FALSE;
    stRgnAttr.unAttr.stOverlay.enPixelFmt = PIXEL_FORMAT_RGB_1555;
    stRgnAttr.unAttr.stOverlay.stRect.s32X= 100;
    stRgnAttr.unAttr.stOverlay.stRect.s32Y= 100;
    stRgnAttr.unAttr.stOverlay.stRect.u32Width = 180;
    stRgnAttr.unAttr.stOverlay.stRect.u32Height = 144;
    stRgnAttr.unAttr.stOverlay.u32BgAlpha = 128;
    stRgnAttr.unAttr.stOverlay.u32FgAlpha = 128;
    stRgnAttr.unAttr.stOverlay.u32BgColor = 0;
    stRgnAttr.unAttr.stOverlay.VeGroup = 0;
    s32Ret = HI_MPI_VPP_CreateRegion(&stRgnAttr, &handle[0]);
    if(s32Ret != HI_SUCCESS)
    {
        printf("HI_MPI_VPP_CreateRegion err 0x%x!\n",s32Ret);
        return HI_FAILURE;
    }

    stRgnAttr.enType = OVERLAY_REGION;
    stRgnAttr.unAttr.stOverlay.bPublic = HI_FALSE;
    stRgnAttr.unAttr.stOverlay.enPixelFmt = PIXEL_FORMAT_RGB_1555;
    stRgnAttr.unAttr.stOverlay.stRect.s32X= 300;
    stRgnAttr.unAttr.stOverlay.stRect.s32Y= 100;
    stRgnAttr.unAttr.stOverlay.stRect.u32Width = 180;
    stRgnAttr.unAttr.stOverlay.stRect.u32Height = 144;
    stRgnAttr.unAttr.stOverlay.u32BgAlpha = 128;
    stRgnAttr.unAttr.stOverlay.u32FgAlpha = 128;
    stRgnAttr.unAttr.stOverlay.u32BgColor = 0x1f;
    stRgnAttr.unAttr.stOverlay.VeGroup = 0;
    s32Ret = HI_MPI_VPP_CreateRegion(&stRgnAttr, &handle[1]);
```

```
if(s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VPP_CreateRegion err 0x%x!\n",s32Ret);
    return HI_FAILURE;
}

stRgnAttr.enType = OVERLAY_REGION;
stRgnAttr.unAttr.stOverlay.bPublic = HI_FALSE;
stRgnAttr.unAttr.stOverlay.enPixelFmt = PIXEL_FORMAT_RGB_1555;
stRgnAttr.unAttr.stOverlay.stRect.s32X= 100;
stRgnAttr.unAttr.stOverlay.stRect.s32Y= 300;
stRgnAttr.unAttr.stOverlay.stRect.u32Width = 48;
stRgnAttr.unAttr.stOverlay.stRect.u32Height = 48;
stRgnAttr.unAttr.stOverlay.u32BgAlpha = 70;
stRgnAttr.unAttr.stOverlay.u32FgAlpha = 70;
stRgnAttr.unAttr.stOverlay.u32BgColor = 0x3e0;
stRgnAttr.unAttr.stOverlay.VeGroup = 0;
s32Ret = HI_MPI_VPP_CreateRegion(&stRgnAttr, &handle[2]);
if(s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VPP_CreateRegion err 0x%x!\n",s32Ret);
    return HI_FAILURE;
}

/*public overlay region*/
stRgnAttr.enType = OVERLAY_REGION;
stRgnAttr.unAttr.stOverlay.bPublic = HI_TRUE;
stRgnAttr.unAttr.stOverlay.enPixelFmt = PIXEL_FORMAT_RGB_1555;
stRgnAttr.unAttr.stOverlay.stRect.s32X= 300;
stRgnAttr.unAttr.stOverlay.stRect.s32Y= 300;
stRgnAttr.unAttr.stOverlay.stRect.u32Width = 48;
stRgnAttr.unAttr.stOverlay.stRect.u32Height = 48;
stRgnAttr.unAttr.stOverlay.u32BgAlpha = 30;
stRgnAttr.unAttr.stOverlay.u32FgAlpha = 30;
stRgnAttr.unAttr.stOverlay.u32BgColor = 0x7c00;

s32Ret = HI_MPI_VPP_CreateRegion(&stRgnAttr, &handle[3]);
if(s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VPP_CreateRegion err 0x%x!\n",s32Ret);
    return HI_FAILURE;
}

/*show all region*/
```

```
    enCtrl = REGION_SHOW;

    for(i=0; i<4; i++)
    {
        s32Ret = HI_MPI_VPP_ControlRegion(handle[i], enCtrl, &unParam);

        if(s32Ret != HI_SUCCESS)
        {
            printf("show faild 0x%x!\n",s32Ret);
            return HI_FAILURE;
        }
    }

    /*ctrl the region*/
    while(1)
    {
        sleep(1);
        s32Cnt++;

        if(s32Cnt <= 10)
        {
            /*change bitmap*/
            if(10 == s32Cnt)
            {
                memset(&unParam, 0, sizeof(REGION_CTRL_PARAM_U));
                pFilename = "mm.bmp";

                /*change bitmap to ARGB1555*/
                SampleLoadBmp(pFilename, &unParam);

                enCtrl = REGION_SET_BITMAP;

                s32Ret = HI_MPI_VPP_ControlRegion(handle[0], enCtrl,
&unParam);

                if(s32Ret != HI_SUCCESS)
                {
                    printf("set region bitmap faild 0x%x!\n",s32Ret);
                    return HI_FAILURE;
                }

                free(unParam.stBitmap.pData);
                unParam.stBitmap.pData = NULL;
            }
```

```
            }
            else if(s32Cnt <= 20)
            {
                /*change bitmap*/
                if(20 == s32Cnt)
                {
                    pFilename = "huawei.bmp";
                    memset(&unParam, 0, sizeof(REGION_CTRL_PARAM_U));
                    SampleLoadBmp(pFilename, &unParam);
                    enCtrl = REGION_SET_BITMAP;

                    s32Ret = HI_MPI_VPP_ControlRegion(handle[1], enCtrl,
&unParam);

                    if(s32Ret != HI_SUCCESS)
                    {
                        printf("REGION_SET_BITMAP 0x%x!\n",s32Ret);
                        return HI_FAILURE;
                    }

                    free(unParam.stBitmap.pData);
                    unParam.stBitmap.pData = NULL;

                }

            }
            else if(s32Cnt <= 30)
            {
                /*change position*/
                enCtrl = REGION_SET_POSTION;
                unParam.stPoint.s32X = 300 + (s32Cnt - 20)*4;
                unParam.stPoint.s32Y = 300 + (s32Cnt - 20)*4;

               s32Ret = HI_MPI_VPP_ControlRegion(handle[3], enCtrl, &unParam);

                if(s32Ret != HI_SUCCESS)
                {
                    printf("REGION_SET_POSTION faild 0x%x!\n",s32Ret);
                    return HI_FAILURE;
                }
            }
            else if(s32Cnt <=40)
            {
```

```
                enCtrl = REGION_SET_ALPHA0;
                unParam.u32Alpha = 128 - (s32Cnt - 30)*8;

               s32Ret = HI_MPI_VPP_ControlRegion(handle[0], enCtrl, &unParam);

                if(s32Ret != HI_SUCCESS)
                {
                    printf("REGION_SET_ALPHA0 0x%x!\n",s32Ret);
                    return HI_FAILURE;
                }

                enCtrl = REGION_SET_ALPHA1;
                unParam.u32Alpha = 128 - (s32Cnt - 30)*8;
               s32Ret = HI_MPI_VPP_ControlRegion(handle[0], enCtrl, &unParam);

                if(s32Ret != HI_SUCCESS)
                {
                    printf("REGION_SET_ALPHA1 faild 0x%x!\n",s32Ret);
                    return HI_FAILURE;
                }
            }
            else
            {
                break;
            }
        }

        for(i=0; i<4; i++)
        {
            s32Ret = HI_MPI_VPP_DestroyRegion(handle[i]);
    if(s32Ret != HI_SUCCESS)
            {
                printf("HI_MPI_VPP_DestroyRegion err 0x%x!\n",s32Ret);
                return HI_FAILURE;
            }
        }

        return HI_SUCCESS;
    }
```

【相关主题】

无。

HI_MPI_VPP_DestroyRegion

【描述】

销毁 VPP 区域。

【语法】

```
HI_S32 HI_MPI_VPP_DestroyRegion(REGION_HANDLE Handle);
```

【参数】

参数名称	描述	输入/输出
Handle	区域句柄。 取值范围：创建区域时返回的有效句柄。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VPP_INVALID_CHNID	无效区域句柄。
HI_ERR_VPP_BUSY	系统忙。
HI_ERR_VPP_UNEXIST	区域不存在。
HI_ERR_VPP_NOTREADY	系统没有初始化或没有加载相应模块。

【需求】

- 头文件：hi_comm_vpp.h、mpi_vpp.h
- 库文件：libmpi.a

【注意】

无。

【举例】

请参见 HI_MPI_VPP_CreateRegion 的举例。

【相关主题】

无。

HI_MPI_VPP_ControlRegion

【描述】

控制 VPP 区域。

【语法】

```
HI_S32 HI_MPI_VPP_ControlRegion(REGION_HANDLE Handle,
REGION_CRTL_CODE_E enCtrl, REGION_CTRL_PARAM_U *punParam);
```

【参数】

参数名称	描述	输入/输出
Handle	区域句柄。 取值范围：创建区域时返回的有效句柄。	输入
enCtrl	控制命令。	输入
punParam	控制参数指针。	输入/输出

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VPP_INVALID_CHNID	无效区域句柄。
HI_ERR_VPP_BUSY	系统忙。
HI_ERR_VPP_NOT_SUPPORT	不支持。
HI_ERR_VPP_UNEXIST	区域不存在。
HI_ERR_VPP_ILLEGAL_PARAM	非法参数。
HI_ERR_VPP_NOT_PERM	操作不允许。

接口返回值	含义
HI_ERR_VPP_NOTREADY	系统没有初始化或没有加载相应模块。

【需求】

- 头文件：hi_comm_vpp.h、mpi_vpp.h
- 库文件：libmpi.a

【注意】

- 对于遮挡区域，可进行的控制包括显示隐藏区域、设置位置、设置尺寸、设置层次、设置颜色、获取该区域的属性，其他操作不允许。
- 对于叠加区域，可进行的控制包括显示隐藏区域、设置位置、设置透明度、设置颜色、设置填充位图、获取该区域的属性，其他操作不允许。
- 填充位图时，位图的格式在创建时就已经确定，若位图的格式为 αRGB1555，其中 alpha 位为 0 时，表示该像素使用背景透明度；alpha 为 1 时，表示该像素使用前景透明度。若位图格式为 αRGB4444，则高 4 位表示该像素的透明度。
- 改变叠加区域的位图时，区域原有位图或背景色会被新的位图覆盖。若位图大于区域大小时，会根据创建区域时的大小按照由左到右、由上到下进行裁减。
- 获取所有区域属性的时候，参数 Handle 将被忽略，并通过 punParam 返回所有区域的属性。
- 对于不同的控制，参数的输入或输出类型也是不同的。在获取单个区域属性和所有区域属性时，punParam 是输出参数；在其他的控制操作时都是输入参数。具体的参数类型对应关系，请参见 REGION_CTRL_PARAM_U。

【举例】

请参见 HI_MPI_VPP_CreateRegion 的举例。

【相关主题】

无。

HI_MPI_VPP_CreateScaleTask

【描述】

创建一个图像缩放任务。

【语法】

```
HI_S32 HI_MPI_VPP_CreateScaleTask(PIC_SCALE_TASK_S *pstTask,
VPP_SCALE_CONF_S *pstConf);
```

【参数】

参数名称	描述	输入/输出
pstTask	缩放任务指针。	输入

参数名称	描述	输入/输出
pstConf	缩放开关和系数指针。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VPP_NULL_PTR	空指针。
HI_ERR_VPP_BUSY	系统忙。
HI_ERR_VPP_ILLEGAL_PARAM	非法参数。
HI_ERR_VPP_NOBUF	无缩放任务缓存。
HI_ERR_VPP_NOTREADY	系统没有初始化或没有加载相应模块。

【需求】

- 头文件：hi_comm_vpp.h、mpi_vpp.h
- 库文件：libmpi.a

【注意】

- 可以进行 8 倍以内的任意比例的缩小和放大。
- 原始图像和目标图像的格式支持 semi-planar YUV422 和 semi-planar YUV420 两种，原始图像和目标图像的格式可以不同。
- 无论缩小或放大，原始图像和目标图像的最大宽高均为 4096 像素。
- 原始图像与目标图像帧场格式应相同。

【举例】

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
```

```
#include <sys/ioctl.h>
#include <sys/poll.h>
#include <fcntl.h>
#include <errno.h>
#include <pthread.h>
#include <math.h>
#include <unistd.h>

#include "hi_common.h"
#include "hi_comm_vpp.h"
#include "mpi_vpp.h"
#include "mpi_vb.h"
#include "hi_comm_vi.h"
#include "mpi_vi.h"

HI_S32 VPP_MST_007_001(HI_VOID)
{
    HI_S32 s32Ret;
    HI_U32 u32TaskId= 0;
    VIDEO_FRAME_INFO_S stFrame;
    VIDEO_FRAME_S stVideoOut;
    PIC_SCALE_TASK_S stTask;
    VPP_SCALE_CONF_S stConf;
    HI_U32 u32BlockFlag = HI_IO_BLOCK;
    HI_U32 u32Timeout = 0;

    VB_POOL VbPool;
    VB_BLK VbBlk;
    HI_U32 u32Addr;

    /*create the scale task ,must be notice how to use the VB*/
    VbPool = HI_MPI_VB_CreatePool(384*288*2, 1);
    if (VB_INVALID_POOLID == VbPool)
    {
        printf("HI_MPI_VB_CreatePool err!");
        return HI_FAILURE;
    }

    /* get blk from a pool */
    VbBlk = HI_MPI_VB_GetBlock(VbPool, 0);
    if (VB_INVALID_HANDLE == VbBlk)
    {
        printf("HI_MPI_VB_GetBlock err!");
        HI_MPI_VB_DestroyPool(VbPool);
```

```
    return HI_FAILURE;
}

/* blk handle to physaddr */
u32Addr = HI_MPI_VB_Handle2PhysAddr(VbBlk);
if (0 == u32Addr)
{
    printf("HI_MPI_VB_Handle2PhysAddr err!");
    HI_MPI_VB_DestroyPool(VbPool);
    return HI_FAILURE;
}

stVideoOut.enPixelFormat = PIXEL_FORMAT_YUV_SEMIPLANAR_422;
stVideoOut.u32Width = 352;
stVideoOut.u32Height = 288;
stVideoOut.u32Field = VIDEO_FIELD_FRAME;

/*light physics addr and stride*/
stVideoOut.u32PhyAddr[0] = u32Addr;
stVideoOut.u32Stride[0]  = 384;

/*chroma physics addr and stride*/
stVideoOut.u32PhyAddr[1] = u32Addr + 384*288;
stVideoOut.u32Stride[1]  = 384;

stTask.stDesPic.stVFrame  = stVideoOut;
stTask.stDesPic.u32PoolId = VbPool;

do
{
    /*get D1 frame from VI*/
    if(HI_MPI_VI_GetFrame(0, 0, &stFrame) < 0)
    {
        printf("HI_MPI_VI_GetFrame err!\n");
        break;
    }

    memcpy(&stTask.stSrcPic,&stFrame,sizeof(VIDEO_FRAME_INFO_S));

    stTask.u32TaskId = u32TaskId;
    stTask.stDesPic.stVFrame.u32Field = stFrame.stVFrame.u32Field;

    stConf.bColorToGrey = HI_FALSE;
    stConf.bDeInterlace = HI_FALSE;
```

```
            stConf.bTemporalDenoise = HI_FALSE;
            stConf.enChoice = VPP_SCALE;
            stConf.enCE = VPP_CE_DISABLE;
            stConf.enLumaStr = VPP_LUMA_STR_DISABLE;
            stConf.enFilter = VPP_SCALE_FILTER_DEFAULT;
            stConf.enSpatialDenoise = VPP_DENOISE_ONLYEDAGE;

            s32Ret = HI_MPI_VPP_CreateScaleTask(&stTask,&stConf);
            if(s32Ret != HI_SUCCESS)
            {
                printf("HI_MPI_VPP_CreateScaleTask err 0x%x!\n",s32Ret);
                break;
            }

            s32Ret = HI_MPI_VPP_WaitScaleTask(&stTask, u32BlockFlag,
u32Timeout);
            if(s32Ret != HI_SUCCESS)
            {
                printf("HI_MPI_VPP_WaitScaleTask err 0x%x!\n",s32Ret);
                break;
            }

            u32TaskId ++;
            printf("task id is %d\n",stTask.u32TaskId);

        }while(u32TaskId < 0xff);

        s32Ret = HI_MPI_VB_ReleaseBlock(VbBlk);
        if(s32Ret != HI_SUCCESS)
        {
            printf("HI_MPI_VB_ReleaseBlock err 0x%x!\n",s32Ret);
            return HI_FAILURE;
        }

        s32Ret = HI_MPI_VB_DestroyPool(VbPool);
        if(s32Ret != HI_SUCCESS)
        {
            printf("HI_MPI_VB_DestroyPool err 0x%x!\n",s32Ret);
            return HI_FAILURE;
        }

        return HI_SUCCESS;
    }
```

【相关主题】

无。

HI_MPI_VPP_WaitScaleTask

【描述】

等待一个图像缩放任务完成。

【语法】

```
HI_S32 HI_MPI_VPP_WaitScaleTask(PIC_SCALE_TASK_S *pstTask, HI_U32
u32BlockFlag, HI_U32 u32Timeout);
```

【参数】

参数名称	描述	输入/输出
pstTask	缩放任务结构指针。	输出
u32BlockFlag	阻塞标志。 取值范围： • HI_IO_BLOCK：阻塞。 • HI_IO_NOBLOCK：非阻塞。	输入
u32Timeout	等待时间。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VPP_NULL_PTR	空指针。
HI_ERR_VPP_BUSY	系统忙。
HI_ERR_VPP_BUF_EMPTY	无缩放任务完成。
HI_ERR_VPP_NOTREADY	系统没有初始化或没有加载相应模块。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 阻塞情况下，等待缩放任务完成还需输入等待的时间，单位为 ms，0 表示无限期等待。
- 如果连续创建多个任务，则需要多次调用等待接口，每次返回一个完成的任务。

【举例】

请参见 HI_MPI_VPP_CreateScaleTask 的举例。

【相关主题】

无。

2.2.5 视频编码

视频编码功能实际包含 VENC 和 GROUP 两个重要的部分，主要提供视频编码通道组的创建和销毁、通道组 GROUP 与视频输入通道的绑定和解绑定、视频编码通道的创建和销毁、注册和反注册到通道组、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、设置和获取数字水印属性、启用和禁用数字水印、视频编码通道属性的设置和查询等功能。

该功能模块提供以下 MPI：

- HI_MPI_VENC_CreateGroup：创建编码通道组。
- HI_MPI_VENC_DestroyGroup：销毁编码通道组。
- HI_MPI_VENC_BindInput：绑定 VI 到通道组。
- HI_MPI_VENC_UnbindInput：解绑定 VI 到通道组。
- HI_MPI_VENC_CreateChn：创建编码通道。
- HI_MPI_VENC_DestroyChn：销毁编码通道。
- HI_MPI_VENC_RegisterChn：注册编码通道到通道组。
- HI_MPI_VENC_UnRegisterChn：反注册编码通道到通道组。
- HI_MPI_VENC_StartRecvPic：开启编码通道接收输入图像。
- HI_MPI_VENC_StopRecvPic：停止编码通道接收输入图像。
- HI_MPI_VENC_Query：查询编码通道状态。
- HI_MPI_VENC_SetChnAttr：设置编码通道的属性。
- HI_MPI_VENC_GetChnAttr：获取编码通道的属性。
- HI_MPI_VENC_GetStream：获取编码码流。
- HI_MPI_VENC_ReleaseStream：释放码流缓存。
- HI_MPI_VENC_RequestIDR：请求 I 帧。
- HI_MPI_VENC_InsertUserData：插入用户数据。
- HI_MPI_VENC_GetCapability：获取视频编码能力集。

- HI_MPI_VENC_SendFrame：支持用户发送原始图像进行编码。
- HI_MPI_VENC_SetWmAttr：设置编码数字水印的属性。
- HI_MPI_VENC_GetWmAttr：获取编码数字水印的属性。
- HI_MPI_VENC_EnableWm：启用编码数字水印。
- HI_MPI_VENC_DisableWm：禁用编码数字水印。
- HI_MPI_VENC_SetMaxStreamCnt：设置码流缓存帧数。
- HI_MPI_VENC_GetMaxStreamCnt：获取码流缓存帧数。
- HI_MPI_VENC_SetH264eRcPara：设置 H.264 编码的量化系数。
- HI_MPI_VENC_GetH264eRcPara：获取 H.264 编码的量化系数。
- HI_MPI_VENC_GetFd：获取编码通道对应的设备文件句柄。
- HI_MPI_VENC_CfgMestPara：设置编码通道运动估计参数。
- HI_MPI_VENC_SetH264eNaluPara：设置 H.264 编码的 nalu 划分参数。
- HI_MPI_VENC_GetH264eNaluPara：获取 H.264 编码的 nalu 划分参数。

HI_MPI_VENC_CreateGroup

【描述】

创建编码通道组。

【语法】

```
HI_S32 HI_MPI_VENC_CreateGroup(VENC_GRP VeGroup);
```

【参数】

参数名称	描述	输入/输出
VeGroup	编码通道组号。 取值范围：[0, VENC_MAX_GRP_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	编码通道组创建成功。

接口返回值	含义
HI_ERR_VENC_EXIST	编码通道组重复创建。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_VENC_INVALID_CHNID	无效的编码通道组号。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 本文档中含有通道组号的接口的通道组号的取值范围为[0, VENC_MAX_GRP_NUM)，否则返回 HI_ERR_VENC_INVALID_CHNID。
- 编码通道组是指芯片能够同时处理的编码通道的集合，一个通道组最多可同时包含一路主码流（H.264/MJPEG）和一路次码流（H.264/MJPEG），或者包含一路 JPEG 抓拍，或者仅包含一路 MPEG4 通道。
- 如果指定的通道组已经存在，则返回错误码 HI_ERR_VENC_EXIST。

【举例】

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <sys/poll.h>
#include <fcntl.h>
#include <errno.h>
#include <pthread.h>
#include <math.h>
#include <unistd.h>

#include "hi_common.h"
#include "hi_comm_video.h"
#include "hi_comm_sys.h"
#include "hi_comm_vi.h"
#include "hi_comm_venc.h"
#include "mpi_vb.h"
#include "mpi_sys.h"
#include "mpi_venc.h"

HI_S32 VencStartVenc1D1(HI_VOID)
```

```
{
    HI_S32 s32Ret;
    HI_S32 s32Cnt = 0;
    VI_DEV ViDev = 0;
    VI_CHN ViChn = 0;
    VENC_GRP VeGroup = 0;
    VENC_CHN VeChn = 0;
    VENC_CHN_ATTR_S stAttr;
    VENC_ATTR_H264_S stH264Attr;
    HI_U8 au8UserData[] = "LIUSHUGUANG64467";

    stH264Attr.u32PicWidth = 720;
    stH264Attr.u32PicHeight = 576;
    stH264Attr.bMainStream = HI_TRUE;
    stH264Attr.bByFrame = HI_TRUE;
    stH264Attr.bCBR = HI_TRUE;
    stH264Attr.bField = HI_FALSE;
    stH264Attr.bVIField = HI_TRUE;
    stH264Attr.u32Bitrate = 4000;
    stH264Attr.u32ViFramerate = 25;
    stH264Attr.u32TargetFramerate = 24;
    stH264Attr.u32BufSize = 720*576*2*2;
    stH264Attr.u32Gop = 20;
    stH264Attr.u32MaxDelay = 10;
    stH264Attr.u32PicLevel = 3;

    stAttr.enType = PT_H264;
    stAttr.pValue = (HI_VOID *)&stH264Attr;

    s32Ret = HI_MPI_VENC_CreateGroup(VeGroup);
    if (s32Ret != HI_SUCCESS)
    {
        printf("HI_MPI_VENC_CreateGroup err 0x%x\n",s32Ret);
        return HI_FAILURE;
    }

    s32Ret = HI_MPI_VENC_BindInput(VeGroup, ViDev, ViChn);
    if (s32Ret != HI_SUCCESS)
    {
        printf("HI_MPI_VENC_BindInput err 0x%x\n",s32Ret);
        return HI_FAILURE;
    }

    s32Ret = HI_MPI_VENC_CreateChn(VeChn, &stAttr, HI_NULL);
```

```
if (s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VENC_CreateChn err 0x%x\n",s32Ret);
    return HI_FAILURE;
}

stH264Attr.u32Gop = 25;
stAttr.enType = PT_H264;
stAttr.pValue = (HI_VOID *)&stH264Attr;
s32Ret = HI_MPI_VENC_SetChnAttr(VeChn, &stAttr);
if (s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VENC_SetChnAttr err 0x%x\n",s32Ret);
    return HI_FAILURE;
}
memset(&stAttr,0,sizeof(VENC_CHN_ATTR_S));
s32Ret = HI_MPI_VENC_GetChnAttr(VeChn, &stAttr);
if (s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VENC_GetChnAttr err 0x%x\n",s32Ret);
    return HI_FAILURE;
}

s32Ret = HI_MPI_VENC_RegisterChn(VeGroup, VeChn);
if (s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VENC_RegisterChn err 0x%x\n",s32Ret);
    return HI_FAILURE;
}

s32Ret = HI_MPI_VENC_StartRecvPic(VeChn);
if (s32Ret != HI_SUCCESS)
{
    printf("HI_MPI_VENC_StartRecvPic err 0x%x\n",s32Ret);
    return HI_FAILURE;
}

while(s32Cnt < 0x5)
{
    sleep(2);
    s32Cnt++;

    s32Ret = HI_MPI_VENC_InsertUserData(VeChn, au8UserData,
                                        sizeof(au8UserData));
```

```
            if(HI_SUCCESS != s32Ret)
            {
                printf("HI_MPI_VENC_InsertUserData err 0x%xn",s32Ret);
                continue;
            }

            s32Ret = HI_MPI_VENC_RequestIDR(VeChn);
            if(HI_SUCCESS != s32Ret)
            {
                printf("HI_MPI_VENC_RequestIDR err 0x%xn",s32Ret);
                continue;
            }
        }

        return HI_SUCCESS;
}
```

【相关主题】

无。

HI_MPI_VENC_DestroyGroup

【描述】

销毁编码通道组。

【语法】

```
HI_S32 HI_MPI_VENC_DestroyGroup(VENC_GRP VeGroup);
```

【参数】

参数名称	描述	输入/输出
VeGroup	编码通道组号。 取值范围：[0, VENC_MAX_GRP_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	无效的编码通道组号。
HI_ERR_VENC_UNEXIST	通道组不存在。
HI_ERR_VENC_NOT_PERM	操作不允许。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 销毁通道组时，必须保证通道组为空，即没有任何通道在通道组中注册，否则会返回错误码 HI_ERR_VENC_NOT_PERM。
- 销毁并不存在的通道组，返回错误码 HI_ERR_VENC_UNEXIST。

【举例】

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <sys/poll.h>
#include <fcntl.h>
#include <errno.h>
#include <pthread.h>
#include <math.h>
#include <unistd.h>

#include "hi_common.h"
#include "hi_comm_video.h"
#include "hi_comm_sys.h"
#include "hi_comm_vi.h"
#include "hi_comm_venc.h"
#include "mpi_vb.h"
#include "mpi_sys.h"
#include "mpi_venc.h"

HI_S32 VencStopVenc1D1(HI_VOID)
```

```
{
    HI_S32 s32Ret;
    VENC_CHN VeChn = 0;
    VENC_GRP VeGroup = 0;

    s32Ret =HI_MPI_VENC_StopRecvPic(VeChn);
    if (s32Ret != HI_SUCCESS)
    {
        printf("HI_MPI_VENC_StopRecvPic err 0x%x\n",s32Ret);
        return HI_FAILURE;
    }

    s32Ret = HI_MPI_VENC_UnRegisterChn(VeChn);
    if (s32Ret != HI_SUCCESS)
    {
        printf("HI_MPI_VENC_UnRegisterChn err 0x%x\n",s32Ret);
        return HI_FAILURE;
    }

    s32Ret = HI_MPI_VENC_DestroyChn(VeChn);
    if (s32Ret != HI_SUCCESS)
    {
        printf("HI_MPI_VENC_DestroyChn err 0x%x\n",s32Ret);
        return HI_FAILURE;
    }

    s32Ret = HI_MPI_VENC_UnbindInput(VeGroup);
    if (s32Ret != HI_SUCCESS)
    {
        printf("HI_MPI_VENC_UnbindInput err 0x%x\n",s32Ret);
        return HI_FAILURE;
    }

    s32Ret =HI_MPI_VENC_DestroyGroup(VeGroup);
    if (s32Ret != HI_SUCCESS)
    {
        printf("HI_MPI_VENC_DestroyGroup err 0x%x\n",s32Ret);
        return HI_FAILURE;
    }

    return HI_SUCCESS;
}
```

【相关主题】

无。

HI_MPI_VENC_BindInput

【描述】

绑定 VI 到通道组。

【语法】

```
HI_S32 HI_MPI_VENC_BindInput(VENC_GRP VeGroup, VI_DEV ViDevId, VI_CHN ViChn);
```

【参数】

参数名称	描述	输入/输出
VeGroup	编码通道组号。 取值范围：[0, VENC_MAX_GRP_NUM)。	输入
ViDevId	VI 设备号。 取值范围：[0, VI_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VI_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_RET_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	无效的通道号。
HI_ERR_VENC_INVALID_DEVID	无效的设备号。
HI_ERR_VENC_UNEXIST	通道组不存在。
HI_ERR_VENC_NOT_PERM	操作不允许。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 绑定并不存在的通道组，则返回错误码 HI_ERR_VENC_UNEXIST。
- 如果 VI 设备或者 VI 通道超出范围，则返回 HI_ERR_VENC_INVALID_DEVID 或者 HI_ERR_VENC_INVALID_CHNID。
- 此接口并不判断 VI 的状态，ViDevId 和 ViChn 可以对应实际的 VI 设备，也可以对应虚拟的 VI 设备，对应虚拟的 VI 设备主要用于用户手动发送图像编码，HI_MPI_VENC_SendFrame 会对此作出详细的说明。
- 如果通道组已经绑定了某个 VI 通道，则返回错误码 HI_ERR_VENC_NOT_PERM。
- 一个通道组只能绑定一个 VI 通道，但一个 VI 通道可以被多个通道组绑定。
- 在编码过程中，可以动态解绑定和绑定 VI，达到编码不同图像源的目的。

【举例】

请参见 HI_MPI_VENC_CreateGroup 的举例。

【相关主题】

无。

HI_MPI_VENC_UnbindInput

【描述】

解绑定 VI 到通道组。

【语法】

```
HI_S32 HI_MPI_VENC_UnbindInput(VENC_GRP VeGroup);
```

【参数】

参数名称	描述	输入/输出
VeGroup	编码通道组号。 取值范围：[0, VENC_MAX_GRP_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	无效的通道组号。
HI_ERR_VENC_UNEXIST	通道组不存在。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 解绑定并不存在的通道组，返回错误码 HI_ERR_VENC_UNEXIST。
- 解绑定之后，VI 通道如果满足条件，可以再绑定到其他任意通道组。
- 可以重复解绑定，返回 HI_SUCCESS。

【举例】

请参见 HI_MPI_VENC_DestroyGroup 的举例。

【相关主题】

无。

HI_MPI_VENC_CreateChn

【描述】

创建编码通道。

【语法】

```
HI_S32 HI_MPI_VENC_CreateChn(VENC_CHN VeChn, const VENC_CHN_ATTR_S
*pstAttr, const VENC_WM_ATTR_S *pstWm);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
pstAttr	编码通道属性指针。	输入

参数名称	描述	输入/输出
pstWm	数字水印属性指针。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	无效的编码通道号。
HI_ERR_VENC_NULL_PTR	空指针。
HI_ERR_VENC_NOT_SUPPORT	不支持。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_VENC_ILLEGAL_PARAM	非法参数。
HI_ERR_VENC_EXIST	编码通道重复创建。
HI_ERR_VENC_NOBUF	内存不足导致 buffer 分配失败。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- Hi3507 支持对主次码流进行编码。在创建编码通道的时候必须指定该通道是主码流还是次码流。
- 在创建编码通道的时候，编码通道属性除需要输入各个协议的特有的编码属性之外，一般还需要输入主次码流（MPEG4 编码协议无此属性）、编码协议、编码的帧场模式、输入图像的帧场模式、获取码流的方式（按帧还是按包获取码流）、编码图像大小属性，这些属性受表 2-2 约束，并且这些属性都为静态属性，不允许动态设置。

表2-2 编码通道的部分属性的约束

编码协议	大小码流	编码方式	编码图像大小
H.264	大码流	Frame、Field	160 % 112 以上，2048 % 1536（3M）像素以下，高度和宽度均 16 像素对齐。
MJPEG	大码流	Frame	160 % 112 以上，2048 % 1536（3M）像素以下，高度和宽度均 16 像素对齐。
H.264 、MJPEG	小码流	Frame	CIF • PAL：352 % 288 • NTSC：352 % 240 QCIF • PAL：176 % 144 • NTSC：176 % 112 QVGA：320 % 240 QQVGA：160 % 112
MPEG4	无	Frame	QCIF • PAL：176 % 144 • NTSC：176 % 112
JPEG 抓拍	无	Frame	80 % 64 以上，2048 % 1536（3M）像素以下，高度和宽度均 16 像素对齐。

- 若输入图像与大码流的宽高相差 16 像素以内（含 16 像素），则大码流编码图像通过输入图像做切边得到。
- 推荐的大码流编码宽高为：2048 % 1536（3M 像素）、1280 % 1024（1.3M 像素）、1280 % 720（720P）、704 % 576、704 % 480、352 % 288、352 % 240。
- 对于 H.264 主码流，编码图像大小不为 D1 时，其编码方式不推荐为 Field。
- 当参数 pstWm 为空时，表示该编码通道不需要使用水印，否则认为需要使用数字水印。如果创建成功，数字水印默认使能。目前只有 H.264 编码的大码流可以设置数字水印，其他的情况设置数字水印时均返回错误码 HI_ERR_VENC_NOT_SUPPORT。

【举例】

请参见 HI_MPI_VENC_CreateGroup 的举例。

【相关主题】

无。

HI_MPI_VENC_DestroyChn

【描述】

销毁编码通道。

【语法】

```
HI_S32 HI_MPI_VENC_DestroyChn(VENC_CHN VeChn);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NOT_PERM	操作不允许。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 销毁并不存在的通道，返回错误码 HI_ERR_VENC_UNEXIST。
- 销毁前必须保证通道已经从通道组注销，否则返回错误码 HI_ERR_VENC_NOT_PERM。

【举例】

请参见 HI_MPI_VENC_DestroyGroup 的举例。

【相关主题】

无。

HI_MPI_VENC_RegisterChn

【描述】

注册编码通道到通道组。

【语法】

```
HI_S32 HI_MPI_VENC_RegisterChn(VENC_GRP VeGroup,VENC_CHN VeChn);
```

【参数】

参数名称	描述	输入/输出
VeGroup	通道组号。 取值范围：[0, VENC_MAX_GRP_NUM)。	输入
VeChn	通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号/通道组号错误。
HI_ERR_VENC_UNEXIST	通道/通道组不存在。
HI_ERR_VENC_NOT_PERM	操作不允许。
HI_ERR_VENC_NOMEM	内存分配失败。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 注册并不存在的通道，返回错误码 HI_ERR_VENC_UNEXIST。
- 注册通道到不存在的通道组，返回错误码 HI_ERR_VENC_UNEXIST。
- 同一个编码通道只能注册到一个通道组，如果该通道已经注册到某个通道组，则返回 HI_ERR_VENC_NOT_PERM。
- 主次码流注册的时候需要判定以下约束关系：
  - 主码流要先于次码流注册，编码在对应的 md 进行操作前注册，否则 MD 启用返回 HI_ERR_VENC_NOT_PERM。
  - 如果编码通道已经注册，则在反注册前不能再进行注册，否则返回 HI_ERR_VENC_NOT_PERM。
- 同组的主次码流若为 1:1 的关系，则编码方式必须同为帧编码，否则返回 HI_ERR_VENC_NOT_PERM。
- 主次码流不能同时为 MJPEG 编码，否则返回 HI_ERR_VENC_NOT_PERM。
- 同组的主次码流宽高必须符合如下约束：D/s – d = R（主次码流宽或高分别为 D 和 d，s 为 1、2 或者 4，R 为 0 到 16）。

【举例】

请参见 HI_MPI_VENC_CreateGroup 的举例。

【相关主题】

无。

HI_MPI_VENC_UnRegisterChn

【描述】

反注册编码通道到通道组。

【语法】

```
HI_S32 HI_MPI_VENC_UnRegisterChn(VENC_CHN VeChn);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NOT_PERM	操作不允许。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 反注册未创建的通道，则返回错误码 HI_ERR_VENC_UNEXIST。
- 如果通道未注册，则返回错误码 HI_ERR_VENC_NOT_PERM。
- 如果编码通道未停止接收图像编码（HI_MPI_VENC_StopRecvPic 可停止接收），则返回错误码 HI_ERR_VENC_NOT_PERM。
- 反注册后会将编码通道复位，如果用户还在使用未及时释放的码流 buffer，将不能保证此 buffer 数据的正确性。用户可以使用 HI_MPI_VENC_Query 接口来查询状态，确认自己所有的操作都完成之后再反注册通道。

【举例】

请参见 HI_MPI_VENC_DestroyGroup 的举例。

【相关主题】

无。

HI_MPI_VENC_StartRecvPic

【描述】

开启编码通道接收输入图像。

【语法】

```
HI_S32 HI_MPI_VENC_StartRecvPic(VENC_CHN VeChn);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NOT_PERM	操作不允许。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 如果通道未创建，则返回 HI_ERR_VENC_UNEXIST。
- 如果通道没有注册到通道组，则返回 HI_ERR_VENC_NOT_PERM。
- 此接口不判断当前是否已经开启接收，直接将状态设置为开启接收。
- 此接口用于开启编码通道接收图像来编码，请注意它和绑定通道组的区别。
- 开始接收输入是针对通道的，只有开启接收之后编码器才开始接收图像编码。

【举例】

请参见 HI_MPI_VENC_CreateGroup 的举例。

【相关主题】

无。

HI_MPI_VENC_StopRecvPic

【描述】

停止编码通道接收输入图像。

【语法】

```
HI_S32 HI_MPI_VENC_StopRecvPic(VENC_CHN VeChn);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NOT_PERM	操作不允许。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 如果通道未创建，则返回 HI_ERR_VENC_UNEXIST。
- 如果通道没有注册到通道组，则返回 HI_ERR_VENC_NOT_PERM。
- 此接口并不判断当前是否停止接收，直接将状态设置为停止接收。

- 此接口用于编码通道停止接收图像来编码，在编码通道反注册前必须停止接收图像。
- 调用此接口仅停止接收原始数据编码，码流 Buffer 并不会被清除。

【举例】

请参见 HI_MPI_VENC_DestroyGroup 的举例。

【相关主题】

无。

HI_MPI_VENC_Query

【描述】

查询编码通道状态。

【语法】

```
HI_S32 HI_MPI_VENC_Query(VENC_CHN VeChn, VENC_CHN_STAT_S *pstStat);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
pstStat	编码通道的状态指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NOT_SUPPORT	不支持。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 如果通道未创建，则返回 HI_ERR_VENC_UNEXIST。
- 此接口用于查询此函数调用时刻的编码器状态，pstStat 包含三个主要的信息：
  - 在编码通道状态结构体中，u32LeftPics 表示待编码的帧个数。

    在反注册通道前，可以通过查询是否还有图像待编码来决定注销时机，防止注销时将可能需要编码的帧清理出去。
  - 在编码通道状态结构体中，u32LeftStreamBytes 表示码流 buffer 中剩余的 byte 数目。

    在反注册通道前，可以通过查询是否还有码流没有被处理来决定注销时机，防止注销时将可能需要的码流清理出去。
  - 在编码通道状态结构体中，u32CurPacks 表示当前帧的码流包个数。

    在按包获取时当前帧可能不是一个完整帧（被取走一部分），按帧获取时表示当前一个完整帧的包个数（如果没有一帧数据则为 0）。用户在需要按帧获取码流时，需要查询一个完整帧的包个数，在这种情况下，通常可以在 select 成功之后执行 query 操作，此时 u32CurPacks 是当前完整帧中包的个数。

【举例】

请参见 HI_MPI_VENC_GetStream 的举例。

【相关主题】

无。

HI_MPI_VENC_SetChnAttr

【描述】

设置编码通道属性。

【语法】

```
HI_S32 HI_MPI_VENC_SetChnAttr(VENC_CHN VeChn, const VENC_CHN_ATTR_S
*pstAttr);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
pstAttr	编码通道属性指针。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NULL_PTR	空指针。
HI_ERR_VENC_NOT_PERM	操作不允许。
HI_ERR_VENC_ILLEGAL_PARAM	非法参数。
HI_ERR_VENC_NOT_SUPPORT	不支持。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 设置未创建的通道的属性，则返回 HI_ERR_VENC_UNEXIST。
- 如果 pstAttr 为空，则返回 HI_ERR_VENC_NULL_PTR。
- 此接口只能设置动态属性，如果设置静态属性，则返回 HI_ERR_VENC_NOT_PERM。主次码流（MPEG4 编码协议无此属性）、编码协议、编码的帧场模式、输入图像的帧场模式、获取码流的方式（按帧还是按包获取码流）、编码图像大小属性均为静态属性。另外，各个编码协议的静态属性由各个协议模块指定，具体请参见 VENC_CHN_ATTR_S。

【举例】

请参见 HI_MPI_VENC_CreateGroup 的举例。

【相关主题】

无。

HI_MPI_VENC_GetChnAttr

【描述】

获取编码通道属性。

【语法】

```
HI_S32 HI_MPI_VENC_GetChnAttr(VENC_CHN VeChn, VENC_CHN_ATTR_S *pstAttr);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
pstAttr	编码通道属性指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NULL_PTR	空指针。
HI_ERR_VENC_NOT_SUPPORT	不支持。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 获取未创建的通道的属性，返回 HI_ERR_VENC_UNEXIST。

- 输入的类型必须和 pstAttr->pValue 指向的结构体类型匹配，否则可能出现无法预料的问题。
- pstAttr->enType 为不支持的类型，则返回 HI_ERR_VENC_NOT_SUPPORT。
- 如果 pstAttr 为空，则返回 HI_ERR_VENC_NULL_PTR。

【举例】

请参见 HI_MPI_VENC_CreateGroup 的举例。

【相关主题】

无。

HI_MPI_VENC_GetStream

【描述】

获取编码的码流。

【语法】

```
HI_S32 HI_MPI_VENC_GetStream(VENC_CHN VeChn, VENC_STREAM_S *pstStream,
HI_U32 u32BlockFlag);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
pstStream	码流结构体指针。	输出
u32BlockFlag	阻塞方式。 取值范围： • HI_IO_BLOCK：阻塞。 • HI_IO_NOBLOCK：非阻塞。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_ILLEGAL_PARAM	非法参数。
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_VENC_NULL_PTR	空指针。
HI_ERR_VENC_BUF_EMPTY	缓冲区中无数据。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 如果通道未创建，返回错误码 HI_ERR_VENC_UNEXIST。
- 如果 pstStream 为空，返回错误码 HI_ERR_VENC_NULL_PTR。
- 支持阻塞或非阻塞两种方式获取。同时可支持 select/poll 系统调用。非阻塞获取时，如果缓冲无数据，则返回 HI_ERR_VENC_BUF_EMPTY；阻塞时，如果缓冲无数据，则会等待有数据时才返回 HI_SUCCESS。
- 支持按包或按帧方式获取码流。如果按包获取，对于 H.264 编码协议，每次获取的是一个 NAL 单元；对于 JPEG 编码协议（包括 JPEG 抓拍和 MJPEG），每次获取的是一个 ECS 或图像参数码流包；对于 MPEG4 编码协议，每次获取的是一帧一个包，因此按帧获取或者按包获取，结果相同。
- 码流结构体 VENC_STREAM_S 包含 3 个部分：
  - 码流包信息指针 pstPack

    指向一组 VENC_PACK_S 的内存空间，该空间由调用者分配。如果是按包获取，则此空间不小于 sizeof（VENC_PACK_S）的大小；如果按帧获取，则此空间不小于 N % sizeof（VENC_PACK_S）的大小，其中 N 代表当前帧之中的包的个数，可以在 select 之后通过查询接口获得。
  - 码流包个数 u32PackCount

    在输入时，此值指定 pstPack 中 VENC_PACK_S 的个数。按包获取时，u32PackCount 必须不小于 1；按帧获取时，u32PackCount 必须不小于当前帧的包个数。在函数调用成功后，u32PackCount 返回实际填充 pstPack 的包的个数。
  - 序列号 u32Seq

    按帧获取时是帧序列号；按包获取时为包序列号。

- 如果用户长时间不获取码流，那么码流缓冲区就会满。一个编码通道如果发生码流缓冲区满，就会停止该编码通道编码，等有足够的码流缓冲可以用来编码时，才开始继续编码，这种情况对于主次码流编码通道来说，相互不受影响。
- 用户应该及时获取码流，防止由于码流 buffer 阻塞导致编码器停止工作。

【举例】

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <sys/poll.h>
#include <fcntl.h>
#include <errno.h>
#include <pthread.h>
#include <math.h>
#include <unistd.h>

#include "hi_common.h"
#include "hi_comm_video.h"
#include "hi_comm_sys.h"
#include "hi_comm_vi.h"
#include "hi_comm_venc.h"
#include "mpi_vb.h"
#include "mpi_sys.h"
#include "mpi_venc.h"

HI_S32 VencGetH264Stream(HI_VOID)
{
    HI_S32 i;
    HI_S32 s32Ret;
    HI_S32 s32VencFd;
    HI_U32 u32FrameIdx = 0;
    VENC_CHN VeChn = 0;
    VENC_CHN_STAT_S stStat;
    VENC_STREAM_S stStream;
    fd_set read_fds;
    FILE *pFile  = NULL;

    pFile = fopen("stream.h264","wb");

    if(pFile == NULL)
    {
```

```
            HI_ASSERT(0);
            return HI_FAILURE;
        }

        s32VencFd = HI_MPI_VENC_GetFd(VeChn);

        do{
            FD_ZERO(&read_fds);
            FD_SET(s32VencFd,&read_fds);

            s32Ret = select(s32VencFd+1, &read_fds, NULL, NULL, NULL);

            if (s32Ret < 0)
            {
                printf("select err\n");
                return HI_FAILURE;
            }
            else if (0 == s32Ret)
            {
                printf("time out\n");
                return HI_FAILURE;
            }
            else
            {
                if (FD_ISSET(s32VencFd, &read_fds))
                {
                    s32Ret = HI_MPI_VENC_Query(VeChn, &stStat);

                    if (s32Ret != HI_SUCCESS)
                    {
                        printf("HI_MPI_VENC_Query:0x%x err\n",s32Ret);
                        fflush(stdout);
                        return HI_FAILURE;
                    }

stStream.pstPack =
(VENC_PACK_S*)malloc(sizeof(VENC_PACK_S)*stStat.u32CurPacks);

                    if (NULL == stStream.pstPack)
                    {
                        printf("malloc memory err!\n");
                        return HI_FAILURE;
                    }
```

```
            stStream.u32PackCount = stStat.u32CurPacks;

            s32Ret = HI_MPI_VENC_GetStream(VeChn, &stStream, HI_TRUE);

            if (HI_SUCCESS != s32Ret)
            {
                printf("HI_MPI_VENC_GetStream:0x%x err!\n",s32Ret);
                fflush(stdout);
                free(stStream.pstPack);
                stStream.pstPack = NULL;
                return HI_FAILURE;
            }

            for (i=0; i< stStream.u32PackCount; i++)
            {
                fwrite(stStream.pstPack[i].pu8Addr[0],
                        stStream.pstPack[i].u32Len[0], 1, pFile);

                fflush(pFile);

                if (stStream.pstPack[i].u32Len[1] > 0)
                {

                    fwrite(stStream.pstPack[i].pu8Addr[1],
                            stStream.pstPack[i].u32Len[1], 1, pFile);

                    fflush(pFile);
                }
            }

            s32Ret = HI_MPI_VENC_ReleaseStream(VeChn,&stStream);
            if (s32Ret)
            {
                printf("HI_MPI_VENC_ReleaseStream:0x%x\n",s32Ret);
                free(stStream.pstPack);
                stStream.pstPack = NULL;
                return HI_FAILURE;
            }

            free(stStream.pstPack);
            stStream.pstPack = NULL;
        }

    }
```

```
        u32FrameIdx ++;
    }while (u32FrameIdx < 0xff);

    fclose(pFile);
    return HI_SUCCESS;
}
```

【相关主题】

无。

HI_MPI_VENC_ReleaseStream

【描述】

释放码流缓存。

【语法】

```
HI_S32 HI_MPI_VENC_ReleaseStream(VENC_CHN VeChn, VENC_STREAM_S
*pstStream);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
pstStream	码流结构体指针。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。

接口返回值	含义
HI_ERR_VENC_NULL_PTR	空指针。
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_ILLEGAL_PARAM	非法参数。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 如果通道未创建，则返回错误码 HI_ERR_VENC_UNEXIST。
- 如果 pstStream 为空，则返回错误码 HI_ERR_VENC_NULL_PTR。
- 此接口应当和 HI_MPI_VENC_GetStream 联合起来使用，用户获取码流后必须及时释放已经获取的码流缓存，否则可能会导致码流 buffer 满，影响编码器编码，并且用户必须按先获取先释放的顺序释放已经获取的码流缓存。
- 在编码通道注销以后，所有未释放的码流包均无效，不能再使用或者释放这部分无效的码流缓存。
- 释放无效的码流会返回 HI_ERR_VENC_ILLEGAL_PARAM。

【举例】

请参见 HI_MPI_VENC_GetStream 的举例。

【相关主题】

无。

HI_MPI_VENC_RequestIDR

【描述】

请求 I 帧。

【语法】

```
HI_S32 HI_MPI_VENC_RequestIDR( VENC_CHN VeChn );
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NOT_SUPPORT	不支持。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 如果通道未创建，则返回 HI_ERR_VENC_UNEXIST。
- 接受 IDR 帧或 I 帧请求后，在尽可能短的时间内编出 IDR 帧或 I 帧。
- I 帧请求，只支持 H.264 和 MPEG4 编码协议，JPEG 和 MJPEG 编码请求会返回 HI_ERR_VENC_NOT_SUPPORT。

【举例】

请参见 HI_MPI_VENC_CreateGroup 的举例。

【相关主题】

无。

HI_MPI_VENC_InsertUserData

【描述】

插入用户数据。

【语法】

```
HI_S32 HI_MPI_VENC_InsertUserData(VENC_CHN VeChn, HI_U8 *pu8Data, HI_U32 u32Len);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
pu8Data	用户数据指针。	输入
u32Len	用户数据长度。 取值范围：[0, 1024]，以 byte 为单位。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_VENC_NULL_PTR	空指针。
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_ILLEGAL_PARAM	非法参数。
HI_ERR_VENC_NOT_SUPPORT	不支持。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 如果通道未创建，则返回错误码 HI_ERR_VENC_UNEXIST。
- 如果 pu8Data 为空，则返回错误码 HI_ERR_VENC_NULL_PTR。
- 用户数据大于 1KB，则返回错误码 HI_ERR_VENC_ILLEGAL_PARAM。
- 插入用户数据，只支持 H.264 和 MJPEG/JPEG 编码协议，MPEG4 编码请求会返回 HI_ERR_VENC_NOT_SUPPORT。

【举例】

请参见 HI_MPI_VENC_CreateGroup 的举例。

【相关主题】

无。

HI_MPI_VENC_GetCapability

【描述】

获取视频编码能力集。

【语法】

```
HI_S32 HI_MPI_VENC_GetCapability(VENC_CAPABILITY_S *pstCap);
```

【参数】

参数名称	描述	输入/输出
pstCap	编码能力集指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_VENC_NULL_PTR	空指针。
HI_ERR_VENC_NOT_SUPPORT	不支持。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 无需创建通道即可获取编码能力集。
- 各个协议编码能力集不同，请参照各个协议的编码能力集描述数据结构。pstCap->pCapability 指向各个协议编码能力集空间，如：H.264 指向 H264_VENC_CAPABILITY_S 空间，JPEG 或者 MJPEG 指向 JPEG_VENC_CAPABILITY_S 空间。

【举例】

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <sys/poll.h>
#include <fcntl.h>
#include <errno.h>
#include <pthread.h>
#include <math.h>
#include <unistd.h>

#include "hi_common.h"
#include "hi_comm_video.h"
#include "hi_comm_sys.h"
#include "hi_comm_vi.h"
#include "hi_comm_venc.h"
#include "mpi_vb.h"
#include "mpi_sys.h"
#include "mpi_venc.h"

HI_S32 VencGetCapability(HI_VOID)
{
    HI_S32 s32Ret;
    VENC_CAPABILITY_S stCapAbility;
    H264_VENC_CAPABILITY_S stH264CapAbility;
    JPEG_VENC_CAPABILITY_S stJpegCapAbility;

    stCapAbility.enType = PT_H264;
    stCapAbility.pCapability = (HI_VOID *)&stH264CapAbility;

    s32Ret = HI_MPI_VENC_GetCapability(&stCapAbility);
    if(HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_GetCapability err 0x%xn",s32Ret);
        return HI_FAILURE;
```

```
    }

    stCapAbility.enType = PT_MJPEG;
    stCapAbility.pCapability = (HI_VOID *)&stJpegCapAbility;

    s32Ret = HI_MPI_VENC_GetCapability(&stCapAbility);
    if(HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_VENC_GetCapability err 0x%xn",s32Ret);
        return HI_FAILURE;
    }
    return HI_SUCCESS;

}
```

【相关主题】

无。

HI_MPI_VENC_SendFrame

【描述】

支持用户发送原始图像进行编码。

【语法】

```
HI_S32 HI_MPI_VENC_SendFrame(VI_DEV ViDevId, VI_CHN ViChn, HI_U32
u32PoolId, VIDEO_FRAME_S *pstFrame, HI_BOOL bIDR);
```

【参数】

参数名称	描述	输入/输出
ViDevId	VI 设备号。 取值范围：[0, VI_MAX_DEV_NUM)。	输入
ViChn	VI 通道号。 取值范围：[0, VI_MAX_CHN_NUM)。	输入
u32PoolId	内存池句柄。 取值范围：[0, VB_MAX_POOLS)。	输入
pstFrame	原始图像信息结构指针。	输入
bIDR	是否编为 IDR 或者 I 帧（当前版本会忽略此标志）。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_VENC_NULL_PTR	空指针。
HI_ERR_VENC_INVALID_CHNID	无效的通道组号。
HI_ERR_VENC_INVALID_DEVID	无效的设备号。
HI_ERR_VENC_ILLEGAL_PARAM	非法参数。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 用户发送原始图像必须为 Semi-planar YUV 4:2:0 格式，如果是场图像，必须是 2 场 Interlaced 的一帧图像。
- ViDevId 和 ViChn 指定一个虚拟的 VI 通道，在编码器接收原始图像进行编码前，用户需要把对应的通道组绑定到该虚拟 VI，VI 设备或者通道超出会返回 HI_ERR_VENC_INVALID_DEVID 或者 HI_ERR_VENC_INVALID_CHNID。
- 当开启 Deinterlace 功能时，只有输入的原始图像超过 2 帧才能进行编码。如果开启时域滤波功能，则需要超过 3 帧才能进行编码。因此对视频输入的原始图像开启 Deinterlace 功能时，编码器不会对最后一帧原始图像进行编码，而开启时域滤波后，第一帧和最后一帧都不会进行编码。
- 视频输入的原始图像大小必须大于或者等于与该视频输入通道绑定的编码通道的编码图像大小，否则编码器不会编码。
- 此版本 bIDR 无效，设置为 HI_TRUE 或者 HI_FALSE 结果一样。

【举例】

请参见 HI_MPI_VENC_CreateGroup 的举例。

【相关主题】

无。

HI_MPI_VENC_SetWmAttr

【描述】

设置编码数字水印的属性。

【语法】

```
HI_S32 HI_MPI_VENC_SetWmAttr(VENC_CHN VeChn, VENC_WM_ATTR_S *pstWm);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
pstWm	编码数字水印属性指针。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_VENC_NULL_PTR	空指针。
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NOT_PERM	操作不允许。
HI_ERR_VENC_NOT_SUPPORT	不支持。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 如果通道未创建，则返回 HI_ERR_VENC_UNEXIST。
- 如果 pstWm 为空，则返回 HI_ERR_VENC_NULL_PTR。
- 必须在水印禁用的情况下才能设置水印属性，否则返回 HI_ERR_VENC_NOT_PERM。

【举例】

无。

【相关主题】

无。

HI_MPI_VENC_GetWmAttr

【描述】

获取编码数字水印的属性。

【语法】

```
HI_S32 HI_MPI_VENC_GetWmAttr(VENC_CHN VeChn, VENC_WM_ATTR_S *pstAttr);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
pstWm	编码数字水印属性指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_VENC_NULL_PTR	空指针。

接口返回值	含义
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NOT_SUPPORT	不支持。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 如果通道未创建，则返回 HI_ERR_VENC_UNEXIST。
- 如果 pstWm 为空，则返回 HI_ERR_VENC_NULL_PTR。

【举例】

无。

【相关主题】

无。

HI_MPI_VENC_EnableWm

【描述】

启用编码数字水印。

【语法】

```
HI_S32 HI_MPI_VENC_EnableWm(VENC_CHN VeChn);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NOT_SUPPORT	不支持。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 如果通道未创建，则返回 HI_ERR_VENC_UNEXIST。
- 仅支持大码流 H.264 编码数字水印功能，否则返回 HI_ERR_VENC_NOT_SUPPORT。
- 可以在存储码流中间启用水印。如果开启水印，从开启时到码流文件结束前，一旦发现水印信息错误或者没有水印，解码时均会反馈水印被篡改。因此在需要检测水印的码流存储结束前，不要禁用水印，否则会引起误报。

【举例】

无。

【相关主题】

无。

HI_MPI_VENC_DisableWm

【描述】

禁用编码数字水印。

【语法】

```
HI_S32 HI_MPI_VENC_DisableWm(VENC_CHN VeChn);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NOT_SUPPORT	不支持。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

无。

【举例】

无。

【相关主题】

无。

HI_MPI_VENC_SetMaxStreamCnt

【描述】

设置码流缓存帧数。

【语法】

```
HI_S32 HI_MPI_VENC_SetMaxStreamCnt(VENC_CHN VeChn,HI_U32 u32MaxStrmCnt);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
u32MaxStrmCnt	缓存帧的个数。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NOT_SUPPORT	不支持。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

无。

【举例】

无。

【相关主题】

无。

HI_MPI_VENC_GetMaxStreamCnt

【描述】

获取码流缓存帧数。

【语法】

```
HI_S32 HI_MPI_VENC_GetMaxStreamCnt(VENC_CHN VeChn,HI_U32 *pu32MaxStrmCnt);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
pu32MaxStrmCnt	码流帧数指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NOT_SUPPORT	不支持。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

无。

【举例】

无。

【相关主题】

无。

HI_MPI_VENC_SetH264eRcPara

【描述】

设置 H.264 编码最小量化系数。

【语法】

```
HI_S32 HI_MPI_VENC_SetH264eRcPara(VENC_CHN VeChn, VENC_ATTR_H264_RC_S
*pstAttr);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
pstAttr	H264 编码的最小量化系数指针	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_NULL_PTR	空指针。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_VENC_ILLEGAL_PARAM	参数错误
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NOT_SUPPORT	不支持。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

无。

【举例】

无。

【相关主题】

无。

HI_MPI_VENC_GetH264eRcPara

【描述】

获取 H.264 编码的量化系数。

【语法】

```
HI_S32 HI_MPI_VENC_GetH264eRcPara(VENC_CHN VeChn, VENC_ATTR_H264_RC_S
*pstAttr);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
pstAttr	H264 编码的最小量化系数指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_NULL_PTR	空指针。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。

接口返回值	含义
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NOT_SUPPORT	不支持。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

无

【举例】

无。

【相关主题】

无。

HI_MPI_VENC_GetFd

【描述】

获取编码通道对应的设备文件句柄。

【语法】

```
HI_S32 HI_MPI_VENC_GetFd(VENC_CHN VeChn);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
正数值	有效返回值。
非正数值	无效返回值。

【错误码】

接口返回值	含义
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

无。

【举例】

请参见 HI_MPI_VENC_GetStream 的举例。

【相关主题】

无。

HI_MPI_VENC_CfgMestPara

【描述】

设置编码通道运动估计参数。

【语法】

```
HI_S32 HI_MPI_VENC_CfgMestPara(VENC_CHN VeChn, VENC_ATTR_MEPARA_S *
pstParam );
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入

参数名称	描述	输入/输出
pstParam	运动估计参数。 水平搜索窗设置使用参数 s32HWSize，取值范围[0, 2]。 0：[-16, +15]。 1：[-32, +31]。 2：[-64, +63]。 垂直搜索窗设置使用参数 s32VWSize，取值范围[0, 1]。 0：[-16, +15]。 1：[-32, 31]。 注意： • 此接口只对 H.264 协议有效，是高级功能接口，一般情况下无需使用。 • s32HWSize 和 s32VWSize 的默认值为 1。 • 不推荐将 s32HWSize 和 s32VWSize 值设置为 0。 • 加大搜索窗可能增加额外的带宽。 • 接口中其他参数保留，暂不使用。	输入

【返回值】

返回值	描述
正数值	有效返回值。
非正数值	无效返回值。

【错误码】

接口返回值	含义
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NULL_PTR	空指针。
HI_ERR_VENC_NOT_PERM	操作不允许。
HI_ERR_VENC_ILLEGAL_PARAM	非法参数。
HI_ERR_VENC_NOT_SUPPORT	不支持。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

无。

【举例】

无。

【相关主题】

无。

HI_MPI_VENC_SetH264eNaluPara

【描述】

设置 H.264 编码的 nalu 划分参数。

【语法】

```
HI_S32 HI_MPI_VENC_SetH264eNaluPara( VENC_CHN VeChn,
VENC_ATTR_H264_NALU_S *pstH264eNalu);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
pstH264eNalu	H.264 编码的 nalu 划分指针。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。

接口返回值	含义
HI_ERR_VENC_NULL_PTR	空指针。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_VENC_ILLEGAL_PARAM	参数错误。
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NOT_SUPPORT	不支持。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

无。

【举例】

无。

【相关主题】

无。

HI_MPI_VENC_GetH264eNaluPara

【描述】

获取 H.264 编码的 nalu 划分参数。

【语法】

```
HI_S32 HI_MPI_VENC_GetH264eNaluPara( VENC_CHN VeChn,
VENC_ATTR_H264_NALU_S *pstH264eNalu);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
pstH264eNalu	H.264 编码的 nalu 划分指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VENC_INVALID_CHNID	通道号错误。
HI_ERR_VENC_NULL_PTR	空指针。
HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_VENC_UNEXIST	通道不存在。
HI_ERR_VENC_NOT_SUPPORT	不支持。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

无.

【举例】

无。

【相关主题】

无。

2.2.6 运动侦测

运动侦测是检测某一正在视频编码的图像是否发生亮度变化以及相应的运动向量。运动侦测通道就是视频编码通道，最大支持 32 路的运动侦测（编码通道号[0, 31]）。最大支持的图像为 D1。

Hi3507 提供的运动侦测功能以宏块为最小单位，计算指定图像的宏块在指定图像间隔内的亮度变化和运动向量。用户要获取运动侦测的结果，需要启用某一视频编码通道的运动侦测功能。运动侦测的结果包括宏块的 SAD 值、宏块运动向量 MV、宏块报警信息、宏块报警像素的个数。

目前支持 H.264 编码和 MJPEG 编码进行运动侦测。而且，对于大小码流编码通道，只支持其中一个码流编码通道进行运动侦测。

该功能模块提供以下 MPI：

- HI_MPI_MD_EnableChn：启用某一路视频编码通道的运动侦测功能。
- HI_MPI_MD_DisableChn：禁用某一路视频编码通道的运动侦测功能。
- HI_MPI_MD_SetChnAttr：设置运动侦测的属性。
- HI_MPI_MD_GetChnAttr：获取运动侦测的属性。
- HI_MPI_MD_SetRefFrame：设置运动侦测的参考图像属性。
- HI_MPI_MD_GetRefFrame：获取运动侦测的参考图像属性。
- HI_MPI_MD_GetData：获取运动侦测结果。
- HI_MPI_MD_ReleaseData：释放运动侦测结果。
- HI_MPI_MD_GetFd：获取运动侦测通道对应的设备文件句柄。

HI_MPI_MD_EnableChn

【描述】

启用某一路视频编码通道的运动侦测功能。

【语法】

```
HI_S32 HI_MPI_MD_EnableChn(VENC_CHN VeChn);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_MD_INVALID_CHNID	通道号错误。
HI_ERR_MD_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_MD_UNEXIST	通道不存在。

接口返回值	含义
HI_ERR_MD_NOT_CONFIG	没有配置属性。
HI_ERR_MD_NOMEM	模块内部分配内存失败。
HI_ERR_MD_NOT_SUPPORT	不支持。
HI_ERR_MD_BUSY	系统忙。

【需求】

- 头文件：hi_comm_md.h、mpi_md.h
- 库文件：libmpi.a

【注意】

- 在启用该编码通道的运动侦测前，相对应的编码通道必须已经创建，且注册到编码通道组 GROUP 中，否则启用失败。运动侦测与编码同时实现，若对应的编码通道没有启动编码，无论运动侦测是否启用，都不会进行运动侦测。
- 在启用前，必须设置该视频编码通道的运动侦测属性。
- 如果需要获取宏块的 SAD 值、宏块报警信息、宏块报警像素的个数，还必须设置参考图像属性。
- 多次启用某一视频编码通道的运动侦测功能，和启用一次效果相同，都会返回成功。
- 目前支持 H.264 编码和 MJPEG 编码时，进行运动侦测。对于大小码流编码通道，只支持其中一个码流编码通道进行运动侦测。

【举例】

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <sys/poll.h>
#include <fcntl.h>
#include <errno.h>
#include <pthread.h>
#include <math.h>
#include <unistd.h>

#include "hi_common.h"
#include "hi_comm_sys.h"
#include "hi_comm_md.h"
#include "mpi_md.h"
```

```
HI_S32 MdStart(HI_VOID)
{
    HI_S32 i;
    HI_S32 s32Ret;
    VENC_CHN VeChn = 0;
    MD_CHN_ATTR_S stMdAttr;
    MD_REF_ATTR_S  stRefAttr;

    /*set MD attribute*/
    stMdAttr.stMBMode.bMBSADMode =HI_TRUE;
    stMdAttr.stMBMode.bMBMVMode = HI_FALSE;
    stMdAttr.stMBMode.bMBPelNumMode = HI_FALSE;
    stMdAttr.stMBMode.bMBALARMMode = HI_FALSE;
    stMdAttr.u16MBALSADTh = 1000;
    stMdAttr.u8MBPelALTh = 20;
    stMdAttr.u8MBPerALNumTh = 20;
    stMdAttr.enSADBits = MD_SAD_8BIT;
    stMdAttr.stDlight.bEnable = HI_FALSE;
    stMdAttr.u32MDInternal = 0;
    stMdAttr.u32MDBufNum = 16;

    /*set MD frame*/
    stRefAttr.enRefFrameMode = MD_REF_AUTO;
    stRefAttr.enRefFrameStat = MD_REF_DYNAMIC;

    s32Ret =  HI_MPI_MD_SetChnAttr(VeChn, &stMdAttr);
    if(s32Ret != HI_SUCCESS)
    {
        printf("HI_MPI_MD_SetChnAttr Err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }

    s32Ret = HI_MPI_MD_SetRefFrame(VeChn, &stRefAttr);
    if(s32Ret != HI_SUCCESS)
    {
        printf("HI_MPI_MD_SetRefFrame Err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }

    memset(&stMdAttr,0,sizeof(MD_CHN_ATTR_S));
    s32Ret = HI_MPI_MD_GetChnAttr(VeChn, &stMdAttr);
    if(s32Ret != HI_SUCCESS)
    {
```

```
        printf("HI_MPI_MD_GetChnAttr Err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }

    memset(&stRefAttr,0,sizeof(MD_REF_ATTR_S));
    s32Ret = HI_MPI_MD_GetRefFrame(VeChn, &stRefAttr);
    if(s32Ret != HI_SUCCESS)
    {
        printf("HI_MPI_MD_GetRefFrame Err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }

    s32Ret = HI_MPI_MD_EnableChn(VeChn);
    if(s32Ret != HI_SUCCESS)
    {
        printf("HI_MPI_MD_EnableChn Err 0x%x\n", s32Ret);
        return HI_FAILURE;
    }

    return HI_SUCCESS;
}
```

【相关主题】

无。

HI_MPI_MD_DisableChn

【描述】

禁用某一路视频编码通道的运动侦测功能。

【语法】

```
HI_S32 HI_MPI_MD_DisableChn(VENC_CHN VeChn);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。

返回值	描述
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_MD_INVALID_CHNID	通道号错误。
HI_ERR_MD_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_MD_UNEXIST	通道不存在。
HI_ERR_MD_BUSY	系统忙。

【需求】

- 头文件：hi_comm_md.h、mpi_md.h
- 库文件：libmpi.a

【注意】

无。

【举例】

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <sys/poll.h>
#include <fcntl.h>
#include <errno.h>
#include <pthread.h>
#include <math.h>
#include <unistd.h>

#include "hi_common.h"
#include "hi_comm_sys.h"
#include "hi_comm_md.h"
#include "mpi_md.h"

HI_S32 MdStop(HI_VOID)
```

```
{
    HI_S32 s32Ret;
    VENC_CHN VeChn = 0;

    s32Ret = HI_MPI_MD_DisableChn(VeChn);

    if(HI_SUCCESS != s32Ret)
    {
        printf("HI_MPI_MD_DisableChn Err 0x%x\n",s32Ret);
        return HI_FAILURE;
    }

    return HI_SUCCESS;
}
```

【相关主题】

无。

HI_MPI_MD_SetChnAttr

【描述】

设置运动侦测的属性。

【语法】

```
HI_S32 HI_MPI_MD_SetChnAttr(VENC_CHN VeChn, const MD_CHN_ATTR_S *pstAttr);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
pstAttr	运动侦测属性指针。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_MD_INVALID_CHNID	通道号错误。
HI_ERR_MD_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_MD_NULL_PTR	空指针。
HI_ERR_MD_UNEXIST	通道不存在。
HI_ERR_MD_NOT_SUPPORT	不支持。
HI_ERR_MD_ILLEGAL_PARAM	参数错误。
HI_ERR_MD_BUSY	系统忙。

【需求】

- 头文件：hi_comm_venc.h、mpi_venc.h
- 库文件：libmpi.a

【注意】

- 在启用运动侦测功能前，需要设置某一路视频编码通道的运动侦测属性。
- 在设置运动侦测的属性前，必须保证运动侦测处于禁用状态。
- 如果要获取宏块报警信息，在去光照效应不打开时，需设置宏块报警 SAD 阈值；在去光照效应打开时，需要设置三个阈值：宏块的 SAD 阈值、宏块内像素报警阈值和宏块内像素报警个数阈值。当发现宏块阈值的变化超过设定的阈值，就认为该宏块是运动的，然后给出宏块的报警信息。
- 如果要获取宏块内像素报警个数时，需设置宏块内像素报警阈值。

【举例】

请参见 HI_MPI_MD_EnableChn 的举例。

【相关主题】

无。

HI_MPI_MD_GetChnAttr

【描述】

获取运动侦测的属性。

【语法】

```
HI_S32 HI_MPI_MD_GetChnAttr(VENC_CHN VeChn, MD_CHN_ATTR_S *pstAttr);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
pstAttr	运动侦测属性指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_MD_INVALID_CHNID	通道号错误。
HI_ERR_MD_NULL_PTR	空指针。
HI_ERR_MD_UNEXIST	通道不存在。
HI_ERR_MD_NOT_CONFIG	没有配置。

【需求】

- 头文件：hi_comm_md.h、mpi_md.h
- 库文件：libmpi.a

【注意】

若该编码通道的运动侦测的属性没有设置，则获取属性失败。

【举例】

请参见 HI_MPI_MD_EnableChn 的举例。

【相关主题】

无。

HI_MPI_MD_SetRefFrame

【描述】

设置运动侦测的参考图像属性。

【语法】

```
HI_S32 HI_MPI_MD_SetRefFrame(VENC_CHN VeChn, const MD_REF_ATTR_S
*pstAttr);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
pstAttr	参考图像属性指针。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_MD_INVALID_CHNID	通道号错误。
HI_ERR_MD_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_MD_NULL_PTR	空指针。
HI_ERR_MD_UNEXIST	通道不存在。
HI_ERR_MD_NOT_SUPPORT	不支持。
HI_ERR_MD_ILLEGAL_PARAM	非法参数。
HI_ERR_MD_NOMEM	分配内存失败（系统内存不足）。
HI_ERR_MD_BUSY	系统忙。

【需求】

- 头文件：hi_comm_md.h、mpi_md.h
- 库文件：libmpi.a

【注意】

- 如果配置的某一视频编码通道运动侦测属性的运动侦测模式中包含宏块的 SAD 值、宏块报警信息、宏块像素报警个数模式中的其中一项或几项，那么必须设置该视频编码通道运动侦测参考图像属性信息。
- 自动设置参考图像时，用户可以根据需要，设置参考图像是否更新。
  - 如果不更新，就按照启用运动侦测后第一帧的编码图像作为参考。
  - 如果更新，就按照运动侦测属性中的运动侦测间隔进行更新。
- 在设置运动侦测的参考图像属性之前必须保证运动侦测处于禁用状态，如果运动侦测为启用状态，则必须首先禁用运动侦测。

【举例】

请参见 HI_MPI_MD_EnableChn 的举例。

【相关主题】

无。

HI_MPI_MD_GetRefFrame

【描述】

获取运动侦测的参考图像属性。

【语法】

```
HI_S32 HI_MPI_MD_GetRefFrame(VENC_CHN VeChn, MD_REF_ATTR_S *pstAttr);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
pstAttr	参考图像属性指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_MD_INVALID_CHNID	通道号错误。
HI_ERR_MD_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_MD_NULL_PTR	空指针。
HI_ERR_MD_UNEXIST	通道不存在。
HI_ERR_MD_NOT_CONFIG	没有配置。
HI_ERR_MD_NOT_SUPPORT	不支持。
HI_ERR_MD_BUSY	系统忙。

【需求】

- 头文件：hi_comm_md.h、mpi_md.h
- 库文件：libmpi.a

【注意】

如果该通道的运动侦测参考图像属性没有配置，那么返回失败。

【举例】

请参见 HI_MPI_MD_EnableChn 的举例。

【相关主题】

无。

HI_MPI_MD_GetData

【描述】

获取运动侦测结果。

【语法】

```
HI_S32 HI_MPI_MD_GetData(VENC_CHN VeChn, MD_DATA_S *pstMdData, HI_U32
u32BlockFlag);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
pstMdData	运动侦测结果指针。	输出

参数名称	描述	输入/输出
u32BlockFlag	阻塞标志。 取值范围： • HI_IO_BLOCK：阻塞。 • HI_IO_NOBLOCK：非阻塞。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_MD_INVALID_CHNID	通道号错误。
HI_ERR_MD_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_MD_NULL_PTR	空指针。
HI_ERR_MD_UNEXIST	通道不存在。
HI_ERR_MD_ILLEGAL_PARAM	操作不允许。
HI_ERR_MD_NOT_SUPPORT	不支持。
HI_ERR_MD_BUSY	系统忙。

【需求】

- 头文件：hi_comm_md.h、mpi_md.h
- 库文件：libmpi.a

【注意】

- 如果运动侦测已经禁用，返回失败。
- 支持阻塞和非阻塞接口。

【举例】

```
#include <stdio.h>
#include <stdlib.h>
```

```
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <sys/poll.h>
#include <fcntl.h>
#include <errno.h>
#include <pthread.h>
#include <math.h>
#include <unistd.h>

#include "hi_common.h"
#include "hi_comm_sys.h"
#include "hi_comm_md.h"
#include "mpi_md.h"
HI_S32 MdGetData(HI_VOID)
{
    HI_S32 i;
    HI_S32 j;
    HI_S32 s32Ret;
    HI_S32 s32MdFd;
    HI_S32 s32Cnt = 0;
    HI_U16* pTmp = NULL;
    MD_DATA_S stMdData;
    VENC_CHN VeChn = 0;
    fd_set read_fds;

    s32MdFd = HI_MPI_MD_GetFd(VeChn);

    do{
        FD_ZERO(&read_fds);
        FD_SET(s32MdFd,&read_fds);

        s32Ret = select(s32MdFd+1, &read_fds, NULL, NULL, NULL);

        if (s32Ret < 0)
        {
            printf("select err\n");
            return HI_FAILURE;
        }
        else if (0 == s32Ret)
        {
            printf("time out\n");
            return HI_FAILURE;
```

```
            }
            else
            {
                sleep(1);
                memset(&stMdData, 0, sizeof(MD_DATA_S));

                if (FD_ISSET(s32MdFd, &read_fds))
                {
                    s32Ret = HI_MPI_MD_GetData(VeChn, &stMdData, HI_IO_BLOCK);
                    if(s32Ret != HI_SUCCESS)
                    {
                        printf("HI_MPI_MD_GetData err 0x%x\n",s32Ret);
                        return HI_FAILURE;
                    }
                }

                s32Cnt++;

                /*get MD SAD data*/
                if(stMdData.stMBMode.bMBSADMode)
                {
                    HI_U16* pTmp = NULL;
                    for(i=0; i<stMdData.u16MBHeight; i++)
                    {
                        pTmp = (HI_U16 *)((HI_U32)stMdData.stMBSAD.pu32Addr+
                                          i*stMdData.stMBSAD.u32Stride);
                        for(j=0; j<stMdData.u16MBWidth; j++)
                        {
                            printf("%2d",*pTmp);
                            pTmp++;
                        }

                        printf("\n");
                    }
}

                s32Ret = HI_MPI_MD_ReleaseData(VeChn, &stMdData);
                if(s32Ret != HI_SUCCESS)
                {
                    printf("Md Release Data Err 0x%x\n",s32Ret);
                    return HI_FAILURE;
                }

            }
```

```
    }while (s32Cnt < 0x1f);

    return HI_SUCCESS;
}
```

【相关主题】

无。

HI_MPI_MD_ReleaseData

【描述】

释放运动侦测缓存。

【语法】

```
HI_S32 HI_MPI_MD_ReleaseData(VENC_CHN VeChn, const MD_DATA_S *pstMdData);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM]。	输入
pstMdData	运动侦测数据指针。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_MD_INVALID_CHNID	通道号错误。
HI_ERR_MD_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_MD_NULL_PTR	空指针。
HI_ERR_MD_UNEXIST	通道不存在。
HI_ERR_MD_NOT_SUPPORT	不支持。

接口返回值	含义
HI_ERR_MD_ILLEGAL_PARAM	非法参数。
HI_ERR_MD_BUSY	系统忙。

【需求】

- 头文件：hi_comm_md.h、mpi_md.h
- 库文件：libmpi.a

【注意】

- 此接口需与 HI_MPI_MD_GetData 配对使用。
- 运动侦测结果需要在使用完之后立即释放，如果不及时释放会导致无运动侦测结果缓存而不能进行运动侦测。
- 释放的运动侦测结果必须是从该通道获取的运动侦测结果，不得对运动侦测结果结构体进行任何修改，也不允许释放从别的通道获取的运动侦测结果，否则会导致运动侦测结果不能释放，使此运动侦测结果缓存丢失，甚至导致程序异常。
- 如果在释放运动侦测结果缓存过程中销毁通道，则立刻返回失败。

【举例】

请参见 HI_MPI_MD_GetData 的举例。

【相关主题】

无。

HI_MPI_MD_GetFd

【描述】

获取运动侦测通道对应的设备文件句柄。

【语法】

```
HI_S32  HI_MPI_MD_GetFd(VENC_CHN VeChn);
```

【参数】

参数名称	描述	输入/输出
VeChn	编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM]。	输入

【返回值】

返回值	描述
正数值	有效返回值。
非正数值	无效返回值。

【错误码】

接口返回值	含义
HI_ERR_MD_INVALID_CHNID	通道号错误。
HI_ERR_MD_SYS_NOTREADY	系统没有初始化或没有加载相应模块。
HI_ERR_MD_BUSY	系统忙。

【需求】

- 头文件：hi_comm_md.h、mpi_md.h
- 库文件：libmpi.a

【注意】

无。

【举例】

请参见 HI_MPI_MD_GetData 的举例。

【相关主题】

无。

2.2.7 视频解码

视频解码模块实现创建解码通道、绑定到视频输出、发送视频码流、获取解码后图像等功能。

该功能模块提供以下 MPI：

- HI_MPI_VDEC_CreateChn：创建视频解码通道。
- HI_MPI_VDEC_DestroyChn：销毁视频解码通道。
- HI_MPI_VDEC_StartRecvStream：解码器开始接收用户发送的码流。
- HI_MPI_VDEC_StopRecvStream：解码器停止接收用户发送的码流。
- HI_MPI_VDEC_Query：查询解码通道状态。
- HI_MPI_VDEC_BindOutput：绑定视频解码通道到视频输出。
- HI_MPI_VDEC_UnbindOutput：解绑定视频解码通道到视频输出。
- HI_MPI_VDEC_SetChnAttr：设置视频解码通道属性。
- HI_MPI_VDEC_GetChnAttr：获取视频解码通道属性。

- HI_MPI_VDEC_SendStream：向视频解码通道发送码流数据。
- HI_MPI_VDEC_GetData：获取视频解码通道的解码数据。
- HI_MPI_VDEC_ReleaseData：释放视频解码通道的图像缓存。
- HI_MPI_VDEC_GetCapability：获取视频解码能力集。
- HI_MPI_VDEC_GetFd：获取视频解码通道的设备文件句柄。
- HI_MPI_VDEC_ResetChn：复位解码通道。

HI_MPI_VDEC_CreateChn

【描述】

创建视频解码通道。

【语法】

```
HI_S32 HI_MPI_VDEC_CreateChn(VDEC_CHN VdChn, const VDEC_CHN_ATTR_S
*pstAttr, const VDEC_WM_ATTR_S *pstWm);
```

【参数】

参数名称	描述	输入/输出
VdChn	视频解码通道号。 取值范围：[0, VDEC_MAX_CHN_NUM)。	输入
pstAttr	解码属性指针。	输入
pstWm	数字水印属性指针（暂不支持该功能，必须置为空指针）。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VDEC_INVALID_CHNID	通道 ID 超出合法范围。

接口返回值	含义
HI_ERR_VDEC_SYS_NOTREADY	系统没有初始化或者相关依赖的模块没有加载（如进行 jpeg 解码时 Hi3511_jpegd.ko 未加载）。
HI_ERR_VDEC_NULL_PTR	参数中有空指针。
HI_ERR_VDEC_ILLEGAL_PARAM	参数错或超出合法范围。
HI_ERR_VDEC_NOT_SUPPORT	不支持的参数或者功能。解码协议不支持，或者数字水印不支持。
HI_ERR_VDEC_EXIST	试图申请或者创建已经存在的设备、通道或者资源。
HI_ERR_VDEC_NOMEM	分配内存失败（如系统内存不足）。
HI_ERR_VDEC_NOBUF	分配缓存失败（如申请的数据缓冲区太大）。
HI_ERR_VDEC_BUSY	系统忙。

【需求】

- 头文件：mpi_vdec.h、hi_comm_vdec.h
- 库文件：libmpi.a

【注意】

- 如果参数 pstAttr 错误或者为空会返回错误码 HI_ERR_VDEC_ILLEGAL_PARAM 或 HI_ERR_VDEC_NULL_PTR。参数 pstAttr 主要包括如下内容：
  - 协议类型
  - 码流 buffer 大小

    码流 buffer 大小不能小于设定的解码图像宽高乘积的 1.5 倍（YUV420 格式大小）。
  - 各个协议属性

    各个协议的属性由各个协议指定，目前 H.264 和 JPEG 协议属性包括优先级、最大解码图像大小、参考图像个数等，H.264 还需要设置码流发送方式（按帧发送或流式发送）。
- 在创建视频解码通道之前必须保证通道未创建（或者已经销毁），否则会直接返回通道已存在错误。
- 目前版本不支持获取数字水印解码，因此参数 pstWm 必须置为空，否则返回错误码 HI_ERR_VDEC_NOT_SUPPORT。

【举例】

```
HI_S32 s32ret;
VDEC_CHN VdChn = 0;
VDEC_ATTR_H264_S    stH264Attr;
```

```
VDEC_CHN_ATTR_S     stAttr;
VDEC_STREAM_S       stStream;
VDEC_DATA_S         stVedcData;

stH264Attr.u32Priority = 0;
stH264Attr.u32PicHeight = 576;
stH264Attr.u32PicWidth = 720;
stH264Attr.u32RefFrameNum = 16;
stH264Attr.enMode = H264D_MODE_STREAM;

stAttr.enType = PT_H264;
stAttr.u32BufSize = (((stH264Attr.u32PicWidth) *
(stH264Attr.u32PicHeight))<<1);
stAttr.pValue = (void*)&stH264Attr;

/* create vdec chn*/
s32ret = HI_MPI_VDEC_CreateChn(VdChn, &stAttr, NULL);
if (HI_SUCCESS != s32ret)
{
    printf("create vdec chn failed, errno 0x%x \n", s32ret);
    return s32ret;
}
/*start recv stream*/
s32ret = HI_MPI_VDEC_StartRecvStream(VdChn);
if (HI_SUCCESS != s32ret)
{
    printf("start recv stream failed, errno 0x%x \n", s32ret);
    return s32ret;
}

/* send stream to vdec chn*/
s32ret = HI_MPI_VDEC_SendStream(VdChn, &stStream, HI_IO_BLOCK);
if (HI_SUCCESS != s32ret)
{
    printf("send stream to vdec chn fail,errno 0x%x \n", s32ret);
    return s32ret;
}

/* get video frame from vdec chn*/
s32ret = HI_MPI_VDEC_GetData(VdChn, &stVedcData, HI_IO_BLOCK);
if (HI_SUCCESS != s32ret)
{
    printf("get video frame from vdec chn fail,errno 0x%x \n", s32ret);
```

```
    return s32ret;
}

/* deal with video frame ,send to vo chn for example ... */

/* release video frame*/
s32ret = HI_MPI_VDEC_ReleaseData(VdChn, &stVedcData);
if (HI_SUCCESS != s32ret)
{
    return s32ret;
}
/* ...*/

/*stop recv stream*/
s32ret = HI_MPI_VDEC_StopRecvStream(VdChn);
if (HI_SUCCESS != s32ret){
    printf("stop recv stream failed errno 0x%x \n", s32ret);
    return s32ret;
}

/* destroy vdec chn*/
s32ret = HI_MPI_VDEC_DestroyChn(VdChn);
if (HI_SUCCESS != s32ret)
{
    printf("destroy vdec chn failed errno 0x%x \n", s32ret);
    return s32ret;
}
```

【相关主题】

无。

HI_MPI_VDEC_DestroyChn

【描述】

销毁视频解码通道。

【语法】

```
HI_S32 HI_MPI_VDEC_DestroyChn(VDEC_CHN VdChn);
```

【参数】

参数名称	描述	输入/输出
VdChn	视频解码通道号。 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，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VDEC_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VDEC_NOT_PERM	操作不允许，在进行此操作前必须停止接收码流。

【需求】

- 头文件：mpi_vdec.h、hi_comm_vdec.h
- 库文件：libmpi.a

【注意】

- 销毁前必须保证通道已创建，否则会返回通道未创建的错误。
- 销毁前必须停止接收码流（或者尚未开始接收码流），否则返回错误码 HI_ERR_VDEC_NOT_PERM。

【举例】

请参见 HI_MPI_VDEC_CreateChn 的举例。

【相关主题】

无。

HI_MPI_VDEC_StartRecvStream

【描述】

解码器开始接收用户发送的码流。

【语法】

```
HI_S32 HI_MPI_VDEC_StartRecvStream(VDEC_CHN VdChn);
```

【参数】

参数名称	描述	输入/输出
VdChn	视频解码通道号。 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，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VDEC_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VDEC_UNEXIST	通道尚未创建或者不存在。
HI_ERR_VDEC_NOT_PERM	操作不允许，重复开始接收码流。
HI_ERR_VDEC_SYS_NOTREADY	系统没有初始化或者已经去初始化。

【需求】

- 头文件：mpi_vdec.h、hi_comm_vdec.h
- 库文件：libmpi.a

【注意】

- 启动接收码流前必须保证通道已创建，否则会返回通道未创建的错误。
- 不允许重复调用此接口，否则返回操作不允许的错误。
- 启动接收码流之后，才能调用 HI_MPI_VDEC_SendStream 发送码流成功。

【举例】

请参见 HI_MPI_VDEC_CreateChn 的举例。

【相关主题】

无。

HI_MPI_VDEC_StopRecvStream

【描述】

解码器停止接收用户发送的码流。

【语法】

```
HI_S32 HI_MPI_VDEC_StopRecvStream(VDEC_CHN VdChn);
```

【参数】

参数名称	描述	输入/输出
VdChn	视频解码通道号。 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，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VDEC_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VDEC_UNEXIST	通道尚未创建或者不存在。
HI_ERR_VDEC_NOT_PERM	操作不允许，重复停止接收码流。
HI_ERR_VDEC_SYS_NOTREADY	系统没有初始化或者已经去初始化。

【需求】

- 头文件：mpi_vdec.h、hi_comm_vdec.h
- 库文件：libmpi.a

【注意】

- 停止接收码流前必须保证通道已创建，否则会返回通道未创建的错误。
- 不允许重复调用此接口，否则返回操作不允许的错误。
- 调用此接口后，调用发送码流接口 HI_MPI_VDEC_SendStream 会返回失败。

【举例】

请参见 HI_MPI_VDEC_CreateChn 的举例。

【相关主题】

无。

HI_MPI_VDEC_Query

【描述】

查询解码通道状态。

【语法】

```
HI_S32 HI_MPI_VDEC_Query(VDEC_CHN VdChn,VDEC_CHN_STAT_S *pstStat);
```

【参数】

参数名称	描述	输入/输出
VdChn	视频解码通道号。 取值范围：[0, VDEC_MAX_CHN_NUM)。	输入
pstStat	视频解码通道状态结构体指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VDEC_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VDEC_UNEXIST	通道尚未创建或者不存在。
HI_ERR_VDEC_SYS_NOTREADY	系统没有初始化或者已经去初始化。

【需求】

- 头文件：mpi_vdec.h、hi_comm_vdec.h
- 库文件：libmpi.a

【注意】

查询通道状态前必须保证通道已创建，否则会返回通道未创建的错误。

【举例】

请参见 HI_MPI_VDEC_CreateChn 的举例。

【相关主题】

无。

HI_MPI_VDEC_BindOutput

【描述】

绑定视频解码通道到视频输出通道。

【语法】

```
HI_S32 HI_MPI_VDEC_BindOutput(VDEC_CHN VdChn, VO_CHN VoChn);
```

【参数】

参数名称	描述	输入/输出
VdChn	视频解码通道号。 取值范围：[0, VDEC_MAX_CHN_NUM)。	输入
VoChn	视频输出 VO 通道号。 取值范围：[0, VO_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VDEC_INVALID_CHNID	解码通道 ID 或者 VO 通道 ID 超出合法范围。
HI_ERR_VDEC_SYS_NOTREADY	系统没有初始化或者已经去初始化。

【需求】

- 头文件：mpi_vdec.h、hi_comm_vdec.h
- 库文件：libmpi.a

【注意】

- 绑定指将视频解码通道和显示输出通道做关联，并不关心实际的解码或者显示输出通道是否已经创建；允许重复绑定，不会返回失败。
- 绑定之后，相应的视频解码通道的数据将直接通过相应的显示输出通道输出，输出帧率由 VO 决定，请参见 HI_MPI_VO_SetFrameRate。
- 绑定并不对视频解码通道和显示输出通道的映射关系作规定，映射关系的策略由业务决定。

【举例】

```
HI_S32 s32ret;
VDEC_CHN VdChn = 0;
VO_CHN VoChn = 0;

/* first create vdec chn*/

/* bind to vo chn */
s32ret = HI_MPI_VDEC_BindOutput(VdChn, VoChn);
if (HI_SUCCESS != s32ret)
{
    printf("bind vdec output to vo failed!");
    return s32ret;
}

/* ... ... */

/* unbind to vo chn*/
s32ret = HI_MPI_VDEC_UnbindOutput(VdChn);
if (HI_SUCCESS != s32ret)
{
    printf("unbind vdec output to vo failed!");
    return s32ret;
}
```

【相关主题】

无。

HI_MPI_VDEC_UnbindOutput

【描述】

解绑定视频解码通道。

【语法】

```
HI_S32 HI_MPI_VDEC_UnbindOutput(VDEC_CHN VdChn);
```

【参数】

参数名称	描述	输入/输出
VdChn	视频解码通道号。 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，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VDEC_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VDEC_SYS_NOTREADY	系统没有初始化或者已经去初始化。

【需求】

- 头文件：mpi_vdec.h、hi_comm_vdec.h
- 库文件：libmpi.a

【注意】

可以重复解绑定，不返回错误。

【举例】

请参见 HI_MPI_VDEC_BindOutput 的举例。

【相关主题】

无。

HI_MPI_VDEC_SetChnAttr

【描述】

设置视频解码通道属性。

【语法】

```
HI_S32 HI_MPI_VDEC_SetChnAttr(VDEC_CHN VdChn, const VDEC_CHN_ATTR_S
*pstAttr);
```

【参数】

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

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VDEC_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VDEC_SYS_NOTREADY	系统没有初始化。
HI_ERR_VDEC_NULL_PTR	参数中有空指针。
HI_ERR_VDEC_NOT_PERM	该操作不允许（如试图修改静态配置参数）。
HI_ERR_VDEC_NOT_SUPPORT	不支持的参数或者功能。
HI_ERR_VDEC_BUSY	系统忙。

【需求】

- 头文件：mpi_vdec.h、hi_comm_vdec.h
- 库文件：libmpi.a

【注意】

- 先创建完通道，再设置属性。如果通道未被创建，则返回失败。

- 不支持静态属性的修改，由于目前的版本没有动态属性，所以此接口无实际意义，仅为了兼容可能存在的动态属性。

【举例】

```
HI_S32              s32ret;
VDEC_CHN            VdChn = 0;
VDEC_ATTR_H264_S    stH264Attr;
VDEC_CHN_ATTR_S     stAttr;

stH264Attr.u32Priority = 0;
stH264Attr.u32PicHeight = 576;
stH264Attr.u32PicWidth = 720;
stH264Attr.u32RefFrameNum = 16;

stAttr.enType = PT_H264;
stAttr.u32BufSize = (((stH264Attr.u32PicWidth) *
(stH264Attr.u32PicHeight))<<1);
stAttr.pValue = (void*)&stH264Attr;

s32ret = HI_MPI_VDEC_SetChnAttr(VdChn, &stAttr);
if (HI_SUCCESS != s32ret)
{
    printf("set vdec chn atrr failed \n");
    return s32ret;
}
```

【相关主题】

无。

HI_MPI_VDEC_GetChnAttr

【描述】

获取视频解码通道属性。

【语法】

```
HI_S32 HI_MPI_VDEC_GetChnAttr(VDEC_CHN VdChn, VDEC_CHN_ATTR_S *pstAttr);
```

【参数】

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

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VDEC_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VDEC_SYS_NOTREADY	系统未初始化。
HI_ERR_VDEC_NULL_PTR	参数中有空指针。
HI_ERR_VDEC_UNEXIST	试图使用不存在的通道。
HI_ERR_VDEC_NOT_SUPPORT	不支持的参数或者功能。
HI_ERR_VDEC_BUSY	系统忙。

【需求】

- 头文件：mpi_vdec.h、hi_comm_vdec.h
- 库文件：libmpi.a

【注意】

获取属性前必须保证通道已创建，否则会返回通道未创建的错误。

【举例】

```
VDEC_CHN            VdChn = 0;
VDEC_CHN_ATTR_S     stAttr;
if (HI_MPI_VDEC_GetChnAttr (VdChn, &stAttr) != HI_RET_SUCCESS )
{
    printf("get vdec chn attr failed!);
    return -1;
}
```

【相关主题】

无。

HI_MPI_VDEC_SendStream

【描述】

向视频解码通道发送码流数据。

【语法】

```
HI_S32 HI_MPI_VDEC_SendStream(VDEC_CHN VdChn, const VDEC_STREAM_S
*pstStream, HI_U32 u32BlockFlag);
```

【参数】

参数名称	描述	输入/输出
VdChn	视频解码通道号。 取值范围：[0, VDEC_MAX_CHN_NUM)。	输入
pstStream	解码码流数据指针。	输入
u32BlockFlag	阻塞非阻塞标志。 取值范围： • HI_IO_BLOCK：阻塞。 • HI_IO_NOBLOCK：非阻塞。 动态属性。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VDEC_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VDEC_SYS_NOTREADY	系统未初始化。
HI_ERR_VDEC_NULL_PTR	参数中有空指针。
HI_ERR_VDEC_ILLEGAL_PARAM	参数超出合法范围。
HI_ERR_VDEC_UNEXIST	试图使用或者销毁不存在的设备、通道或者资源。

接口返回值	含义
HI_ERR_VDEC_NOT_PERM	操作不允许，在通道未准备好接收码流的情况下向此通道发送码流。
HI_ERR_VDEC_BUF_FULL	缓冲区满，非阻塞接口如果发送失败则返回此错误。
HI_ERR_VDEC_NOT_SUPPORT	不支持的参数或者功能。

【需求】

- 头文件：mpi_vdec.h、hi_comm_vdec.h
- 库文件：libmpi.a

【注意】

- 当前只支持 H.264 解码，可以按照小于码流总 buffer 大小的任意长度输入码流。创建其他协议的解码通道会返回 HI_ERR_VDEC_NOT_SUPPORT。
- 此接口通过改变 Flag 值支持阻塞方式和非阻塞方式工作。
- 发送数据前必须保证通道已经启动接收码流，否则直接返回 HI_ERR_VDEC_NOT_PERM。如果在发送数据过程中停止接收码流，就会立刻返回 HI_ERR_VDEC_NOT_PERM。
- 发送数据前必须保证通道已经被创建，否则直接返回 HI_ERR_VDEC_UNEXIST。如果在发送数据过程中销毁通道，就会立刻返回 HI_ERR_VDEC_UNEXIST。
- 发送码流时需要按照创建解码通道时设置的发送方式（按帧发送或流式发送）进行发送。

【举例】

请参见 HI_MPI_VDEC_CreateChn 的举例。

【相关主题】

无。

HI_MPI_VDEC_GetData

【描述】

获取视频解码通道的解码数据。

【语法】

```
HI_S32 HI_MPI_VDEC_GetData(VDEC_CHN VdChn, VDEC_DATA_S *pstData, HI_U32
u32BlockFlag);
```

【参数】

参数名称	描述	输入/输出
VdChn	视频解码通道号。 取值范围：[0, VDEC_MAX_CHN_NUM)。	输入
u32BlockFlag	阻塞非阻塞标志。 取值范围： • HI_IO_BLOCK：阻塞。 • HI_IO_NOBLOCK：非阻塞。 动态属性。	输入
pstData	获取的解码数据，包括解码后的图像信息和用户数据。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VDEC_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VDEC_SYS_NOTREADY	系统未初始化。
HI_ERR_VDEC_NULL_PTR	参数中有空指针。
HI_ERR_VDEC_UNEXIST	试图使用或者销毁不存在的设备、通道或者资源。
HI_ERR_VDEC_BUF_EMPTY	非阻塞获取时，缓冲区中无数据。
HI_ERR_VDEC_BUSY	系统忙。

【需求】

- 头文件：mpi_vdec.h、hi_comm_vdec.h
- 库文件：libmpi.a

【注意】

- 解码数据包括图像信息和用户数据。
- 此接口支持阻塞方式和非阻塞方式工作。
- 如果获取成功返回，要根据图像信息和用户数据结构体中的判断位来判断，判断获取的数据是图像信息，还是用户数据中的一项或者多项。
- 通过 HI_MPI_VDEC_GetData 获取的解码后图像数据使用后，需要通过 HI_MPI_VDEC_ReleaseData 来释放。
- 获取解码后数据时必须保证通道已经被创建，否则直接返回失败，如果在获取解码后数据的过程中销毁通道，则会立刻返回失败。

【举例】

请参见 HI_MPI_VDEC_CreateChn 的举例。

【相关主题】

无。

HI_MPI_VDEC_ReleaseData

【描述】

释放视频解码通道的图像缓存。

【语法】

```
HI_S32 HI_MPI_VDEC_ReleaseData(VDEC_CHN VdChn, VDEC_DATA_S *pstData);
```

【参数】

参数名称	描述	输入/输出
VdChn	视频解码通道号。 取值范围：[0, VDEC_MAX_CHN_NUM)。	输入
pstData	解码后的图像信息指针，由 HI_MPI_VDEC_GetData 接口获取。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VDEC_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VDEC_SYS_NOTREADY	系统未初始化。
HI_ERR_VDEC_NULL_PTR	参数中有空指针。
HI_ERR_VDEC_UNEXIST	试图使用或者销毁不存在的设备、通道或者资源。
HI_ERR_VDEC_ILLEGAL_PARAM	参数超出合法范围。
HI_ERR_VDEC_BUSY	系统忙。

【需求】

- 头文件：mpi_vdec.h、hi_comm_vdec.h
- 库文件：libmpi.a

【注意】

- 此接口需与 HI_MPI_VDEC_GetData 配对使用，获取的数据应当在使用完之后立即释放，如果不及时释放，会导致解码过程阻塞等待资源。
- 释放的数据必须是 HI_MPI_VDEC_GetData 从该通道获取的数据，不得对数据信息结构体进行任何修改，也不允许释放从其他的通道获取的数据，否则会导致数据不能释放，使此数据 buffer 丢失，甚至导致程序异常。
- 释放数据时必须保证通道已经被创建，否则直接返回 HI_ERR_VDEC_UNEXIST。如果在释放数据过程中销毁通道，则会立即返回 HI_ERR_VDEC_UNEXIST。

【举例】

请参见 HI_MPI_VDEC_CreateChn 的举例。

【相关主题】

无。

HI_MPI_VDEC_GetCapability

【描述】

获取视频解码能力集。

【语法】

```
HI_S32 HI_MPI_VDEC_GetCapability(VDEC_CAPABILITY_S *pstCap);
```

【参数】

参数名称	描述	输入/输出
pstCap	解码能力集指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VDEC_NULL_PTR	参数中有空指针。
HI_ERR_VDEC_NOT_SUPPORT	不支持的参数或者功能。
HI_ERR_VDEC_SYS_NOTREADY	系统未初始化。
HI_ERR_VDEC_BUSY	系统忙。

【需求】

- 头文件：mpi_vdec.h、hi_comm_vdec.h
- 库文件：libmpi.a

【注意】

- 无需创建通道即可获取解码能力集。
- 各个协议编码能力集不同。

【举例】

```
VDEC_CAPABILITY_S   scapability;
H264_VDEC_CAPABILITY_S sh264dcapability;

scapability.enType = PT_H264;
scapability.pCapability = &sh264dcapability;
if (HI_MPI_VDEC_GetCapability(&scapability) != HI_SUCCESS )
{
    printf("get vdec capability failed!\n");
}
```

【相关主题】

无。

HI_MPI_VDEC_GetFd

【描述】

获取视频解码通道的设备文件句柄。

【语法】

```
HI_S32 HI_MPI_VDEC_GetFd(VDEC_CHN VdChn);
```

【参数】

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

【返回值】

返回值	描述
正数值	有效返回值。
非正数值	无效返回值。

【错误码】

无。

【需求】

- 头文件：mpi_vdec.h、hi_comm_vdec.h
- 库文件：libmpi.a

【注意】

无。

【举例】

```
int fd;
VDEC_CHN VdChn = 0;

fd = HI_MPI_VDEC_GetFd(VdChn);
if (fd <= 0)
{
    printf("get vdec chn fd err \n!");
```

```
    return -1;
}
```

【相关主题】

无。

HI_MPI_VDEC_ResetChn

【描述】

复位视频解码通道。

【语法】

```
HI_S32 HI_MPI_VDEC_ResetChn(VDEC_CHN VdChn);
```

【参数】

参数名称	描述	输入/输出
VdChn	视频解码通道号。 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，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_VDEC_INVALID_CHNID	通道 ID 超出合法范围。
HI_ERR_VDEC_UNEXIST	通道不存在。
HI_ERR_VDEC_NOT_PERM	操作不允许。在进行此操作前必须停止接收码流。

【需求】

- 头文件：mpi_vdec.h、hi_comm_vdec.h
- 库文件：libmpi.a

【注意】

- 复位前必须保证通道已创建，否则会返回错误码 HI_ERR_VDEC_UNEXIST。
- 复位前必须停止接收码流，否则返回错误码 HI_ERR_VDEC_NOT_PERM。
- 复位操作未完成前，不响应用户的任何操作，如发送码流、获取数据、销毁通道等，均会返回错误码 HI_ERR_VDEC_NOT_PERM。

【举例】

```
/* stop recv stream */
s32ret =  HI_MPI_VDEC_StopRecvStream (VdChn);
if (HI_SUCCESS != s32ret){
    printf("stop recv stream failed errno 0x%x \n", s32ret);
    return s32ret;
}

/* reset vdec chn*/
s32ret = HI_MPI_VDEC_ResetChn(VdChn);
if (HI_SUCCESS != s32ret)
{
    printf("reset vdec chn failed errno 0x%x \n", s32ret);
    return s32ret;
}


/* re-start vdec to receive stream */
s32ret = HI_MPI_VDEC_StartRecvStream(VdChn);
if (HI_SUCCESS != s32ret)
{
    printf("start recv stream failed, errno 0x%x \n", s32ret);
    return s32ret;
}
```

【相关主题】

无。

2.3 音频功能函数参考

2.3.1 音频输入

音频输入（AI）主要实现配置及启用音频输入设备、获取音频帧数据等功能。

该功能模块提供以下 MPI：

- HI_MPI_AI_SetPubAttr：设置 AI 设备属性。

- HI_MPI_AI_GetPubAttr：获取 AI 设备属性。
- HI_MPI_AI_Enable：启用 AI 设备。
- HI_MPI_AI_Disable：禁用 AI 设备。
- HI_MPI_AI_EnableChn：启用 AI 通道。
- HI_MPI_AI_DisableChn：禁用 AI 通道。
- HI_MPI_AI_GetFrame：获取 AI 通道音频帧数据。
- HI_MPI_AI_EnableAec：启用回声抵消功能。
- HI_MPI_AI_DisableAec：禁用回声抵消功能。
- HI_MPI_AI_GetFd：获取 AI 通道对应设备文件句柄。

HI_MPI_AI_SetPubAttr

【描述】

设置 AI 设备属性。

【语法】

```
HI_S32 HI_MPI_AI_SetPubAttr(AUDIO_DEV AudioDevId,const AIO_ATTR_S
*pstAttr);
```

【参数】

参数名称	描述	输入/输出
AudioDevId	音频设备号。 取值范围：[0, SIO_MAX_NUM)。	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，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AI_INVALID_DEVID	音频输入设备号无效。
HI_ERR_AI_ILLEGAL_PARAM	输入参数无效。

接口返回值	含义
HI_ERR_AI_NULL_PTR	空指针错误。
HI_ERR_AI_SYS_NOTREADY	系统未初始化成功。
HI_ERR_AI_NOT_PERM	操作非法。
HI_ERR_AI_NOT_SUPPORT	操作不支持。

【需求】

- 头文件：hi_comm_aio.h、mpi_ai.h、hi_comm_aio.h
- 库文件：libmpi.a

【注意】

- 音频输入设备的属性决定了输入数据的格式，输入设备属性包括 SIO 工作模式、采样率、采样精度、buffer 大小、每帧的采样点数和扩展标志。
  - 工作模式

    SIO 输入输出目前支持 I^2S 主模式、I^2S 从模式以及 PCM 的两种从模式；主从模式决定时钟由谁发起，主模式表示 SIO 发起时钟，从模式表示 AD 发起时钟；I^2S 和 PCM 是两种音频数据传输总线协议，Hi3507 SIO 支持标准 I^2S 和 PCM 协议，另外支持一种非标准 PCM 协议；SIO 的工作模式必须与外围对接的 codec 芯片设置一致；总线协议相关具体描述可以参考《Hi3507 H.264 编解码处理器　用户指南》中音频部分。
  - 采样率

    采样率指一秒中内的采样点数，采样率越高表明失真度越小，处理的数据量也就随之增加。主模式下 AI 支持 8k～48k 的采样率，一般来说语音使用 8k 采样率，音频使用 32k 或以上的采样率；在从模式下，采样率由 codec 芯片决定。
  - 采样精度

    采样精度指某个通道的采样点数据宽度，同时决定整个设备的通道分布。采样位宽可以设置为 8bit、16bit 和 32bit。
  - buffer 大小

    buffer 大小以帧为单位，每帧的采样点数和采样精度决定帧长，buffer 大小设置当前能容纳帧的个数。
  - 扩展标志

    扩展标志表明在 8bit 采样精度的条件下是否需要对 8bit 数据进行 8bit 到 16bit 带符号扩展，扩展后获取的数据就为 16bit，以满足编码器需求。在设置非 8bit 采样精度的情况下，此标志是无效的。
- 在设置属性之前需要保证 AI 处于非启用状态，如果处于启用状态则需要首先禁用 AI 设备。
- 在主模式下，由于 AI、AO 设备共用同一时钟，所以如果 AI 配置的采样率和主从模式有变化则会引起相同设备号的 AO 的采样率和主从模式的相应变化。
- 在从模式下，采样率的设置不起作用。

- AI 必须和 AD 配合起来才能正常工作，用户必须清楚 AD 采集的数据分布和通道的关系才能从正确的通道取得数据。

【举例】

下面的代码实现设置 AI 设备属性及启用 AI 设备。

```
HI_S32 s32ret;
AIO_ATTR_S stAttr;
AUDIO_DEV AiDevId = 1;

stAttr.enBitwidth = AUDIO_BIT_WIDTH_16;
stAttr.enSamplerate = AUDIO_SAMPLE_RATE_8000;
stAttr.enSoundmode = AUDIO_SOUND_MODE_MOMO;
stAttr.enWorkmode = AIO_MODE_I2S_SLAVE;
stAttr.u32EXFlag = 0;
stAttr.u32FrmNum = 5;
stAttr.u32PtNumPerFrm = 160;

/* set public attribute of AI device*/
s32ret = HI_MPI_AI_SetPubAttr(AiDevId, &stAttr);
if(HI_SUCCESS != s32ret)
{
    printf("set ai %d attr err:0x%x\n", AiDevId,s32ret);
    return s32ret;
}
/* enable AI device */
s32ret = HI_MPI_AI_Enable(AiDevId);
if(HI_SUCCESS != s32ret)
{
    printf("enable ai dev %d err:0x%x\n", AiDevId, s32ret);
    return s32ret;
}
```

【相关主题】

无。

HI_MPI_AI_GetPubAttr

【描述】

获取 AI 设备属性。

【语法】

```
HI_S32 HI_MPI_AI_GetPubAttr(AUDIO_DEV AudioDevId, AIO_ATTR_S *pstAttr);
```

【参数】

参数名称	描述	输入/输出
AudioDevId	音频设备号。 取值范围：[0, SIO_MAX_NUM)。	输入
pstAttr	AI 设备公共属性指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AI_INVALID_DEVID	音频输入设备号无效。
HI_ERR_AI_NOT_CONFIG	音频输入设备属性未配置。
HI_ERR_AI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_aio.h、mpi_ai.h、hi_comm_aio.h
- 库文件：libmpi.a

【注意】

- 获取的属性为前一次配置的属性。
- 如果从来没有配置过属性，则返回属性未配置的错误。

【举例】

```
HI_S32 s32ret;
AUDIO_DEV AiDevId = 1;
AIO_ATTR_S stAttr;

s32ret = HI_MPI_AI_GetPubAttr(AiDevId, &stAttr);
if(HI_SUCCESS != s32ret)
{
    printf("get ai %d attr err:0x%x\n", AiDevId,s32ret);
    return s32ret;
```

}

【相关主题】

无。

HI_MPI_AI_Enable

【描述】

启用 AI 设备。

【语法】

```
HI_S32 HI_MPI_AI_Enable(AUDIO_DEV AudioDevId);
```

【参数】

参数名称	描述	输入/输出
AudioDevId	音频设备号。 取值范围：[0, SIO_MAX_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AI_INVALID_DEVID	音频输入设备号无效。
HI_ERR_AI_SYS_NOTREADY	系统未初始化成功。
HI_ERR_AI_NOT_PERM	操作非法。

【需求】

- 头文件：hi_comm_aio.h、mpi_ai.h、hi_comm_aio.h
- 库文件：libmpi.a

【注意】

- 必须在启用前配置 AI 设备属性，否则返回属性未配置错误。

- 如果 AI 设备已经处于运行状态，则直接返回成功。

【举例】

请参见 HI_MPI_AI_SetPubAttr 的举例。

【相关主题】

无。

HI_MPI_AI_Disable

【描述】

禁用 AI 设备。

【语法】

```
HI_S32 HI_MPI_AI_Disable(AUDIO_DEV AudioDevId);
```

【参数】

参数名称	描述	输入/输出
AudioDevId	音频设备号。 取值范围：[0, SIO_MAX_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AI_INVALID_DEVID	音频输入设备号无效。
HI_ERR_AI_SYS_NOTREADY	系统未初始化成功。
HI_ERR_AI_NOT_PERM	操作非法。

【需求】

- 头文件：hi_comm_aio.h、mpi_ai.h、hi_comm_aio.h
- 库文件：libmpi.a

【注意】

- 如果 AI 设备已经处于非运行状态，则直接返回成功。
- 禁用 AI 设备前必须先禁用该设备下已启用的所有 AI 通道。

【举例】

```
HI_S32 s32ret;
AUDIO_DEV AiDevId = 1;

s32ret = HI_MPI_AI_Disable(AiDevId);
if(HI_SUCCESS != s32ret)
{
    printf("disable ai %d err:0x%x\n", AiDevId);
    return s32ret;
}
```

【相关主题】

无。

HI_MPI_AI_EnableChn

【描述】

启用 AI 通道。

【语法】

```
HI_S32 HI_MPI_AI_EnableChn(AUDIO_DEV AudioDevId, AI_CHN AiChn);
```

【参数】

参数名称	描述	输入/输出
AudioDevId	音频设备号。 取值范围：[0, SIO_MAX_NUM)。	输入
AiChn	音频输入通道号。 实际支持的通道范围由采样精度决定。例如：采用 8bit 精度则支持 8 个 AI 通道，16bit 则只支持 4 个 AI 通道。 取值范围：[0, AIO_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。

返回值	描述
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AI_INVALID_DEVID	音频输入设备号无效。
HI_ERR_AI_INVALID_CHNID	音频输入通道号无效。
HI_ERR_AI_SYS_NOTREADY	系统未初始化成功。
HI_ERR_AI_NOT_ENABLED	音频设备未启用。

【需求】

- 头文件：hi_comm_aio.h、mpi_ai.h、hi_comm_aio.h
- 库文件：libmpi.a

【注意】

启用 AI 通道前，必须先启用其所属的 AI 设备，否则返回设备未启动的错误码。

【举例】

无。

【相关主题】

无。

HI_MPI_AI_DisableChn

【描述】

禁用 AI 通道。

【语法】

```
HI_S32 HI_MPI_AI_DisableChn(AUDIO_DEV AudioDevId, AI_CHN AiChn);
```

【参数】

参数名称	描述	输入/输出
AudioDevId	音频设备号。 取值范围：[0, SIO_MAX_NUM)。	输入

参数名称	描述	输入/输出
AiChn	音频输入通道号。 实际支持的通道范围由采样精度决定。例如：采用 8bit 精度则支持 8 个 AI 通道，16bit 则只支持 4 个 AI 通道。 取值范围：[0, AIO_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AI_INVALID_DEVID	音频输入设备号无效。
HI_ERR_AI_INVALID_CHNID	音频输入通道号无效。
HI_ERR_AI_SYS_NOTREADY	系统未初始化成功。
HI_ERR_AI_NOT_PERM	操作非法。

【需求】

- 头文件：hi_comm_aio.h、mpi_ai.h、hi_comm_aio.h
- 库文件：libmpi.a

【注意】

无。

【举例】

无。

HI_MPI_AI_GetFrame

【描述】

从 AI 通道获取音频帧数据。

【语法】

```
HI_S32 HI_MPI_AI_GetFrame(AUDIO_DEV AudioDevId, AI_CHN AiChn,
AUDIO_FRAME_S *pstData, AEC_FRAME_S *pstAecData, HI_U32 u32Block);
```

【参数】

参数名称	描述	输入/输出
AudioDevId	音频设备号。 取值范围：[0, SIO_MAX_NUM)。	输入
AiChn	音频输入通道号。实际支持的通道范围由采样精度和主从模式决定。例如：采用 8bit 精度则支持 8 个 AI 通道，16bit 则只支持 4 个 AI 通道。 取值范围：[0, AIO_MAX_CHN_NUM)。	输入
u32Block	阻塞/非阻塞标志。 取值范围： • HI_IO_BLOCK：阻塞。 • HI_IO_NOBLOCK：非阻塞。 动态属性。	输入
pstData	音频帧数据指针。	输出
pstAecData	用于回声抵消的参考帧数据信息。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AI_INVALID_DEVID	音频输入设备号无效。
HI_ERR_AI_INVALID_CHNID	音频输入通道号无效。
HI_ERR_AI_NULL_PTR	空指针错误。
HI_ERR_AI_BUF_EMPTY	音频输入缓存空。

接口返回值	含义
HI_ERR_AI_NOT_ENABLED	音频输入通道未启用。
HI_ERR_AI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_aio.h、mpi_ai.h、hi_comm_aio.h
- 库文件：libmpi.a

【注意】

- 获取音频帧数据前必须先启用音频输入通道，否则返回音频通道未启用错误码。
- 音频帧数据包含完整的帧信息：数据指针、帧长、采样率、采样精度、时间戳和帧号。
  - 帧长 len 是指数据 buffer 中单个声道的数据字节数，立体声表示左右声道均为 len 的长度。len 的最大长度由 MAX_AUDIO_FRAME_LEN 指定。
  - 采样精度一般和设备属性设置值是一致的，但是如果设置 8bit 采样精度，同时又指定 8 到 16bit 的扩展，那么获取帧的采样精度是 16bit。
  - 时间戳是指采集此帧的绝对时间，以 μs 为单位。
  - 帧号仅做调试使用。
- 单声道数据直接存放，采样点数为 ptnum，长度为 len；立体声数据按左右声道分开存放，先存放采样点为 ptnum，长度为 len 的左声道数据，然后存放采样点为 ptnum，长度为 len 的右声道数据。
- 音频参考帧是指启用回声抵消后从 AO 获取的数据，用于编码前回声抵消；pstAecData-> bValid 表明获取的数据是否有效（**注意**：即使成功启用回声抵消功能，也未必能获取有效参考帧数据，因此如果使用回声抵消，必须对此标志进行判断）。
- 获取的音频参考帧的有效性取决于回声抵消是否启用以及 AO 设备的属性（AO 设备必须处于启用状态，采样率必须为 8kHz，帧长必须与 AI 设置的帧长一致，采样精度必须为 8bit 或 16bit）。
- 如果不需要回声抵消的参考帧数据，可以将参数 pstAecData 置为空。
- 要求 AI 设备必须处于启用状态，否则会立刻返回失败。
- 在获取数据过程中禁用通道，会立刻返回失败。
- 如果取数据不及时，可能导致帧数据丢失。

【举例】

```
HI_S32 s32ret;
AUDIO_DEV AiDevId = 1;
AI_CHN AiChn = 0;
AUDIO_FRAME_S stData;

s32ret = HI_MPI_AI_GetFrame(AiDevId, AiChn, &stData, NULL, HI_IO_BLOCK);
if(HI_SUCCESS != s32ret)
```

```
{
    printf("get ai frame err:0x%x\n", s32ret);
    return s32ret;
}
```

【相关主题】

无。

HI_MPI_AI_EnableAec

【描述】

启用指定 AI 及 AO 的回声抵消功能。

【语法】

```
HI_S32 HI_MPI_AI_EnableAec(AUDIO_DEV AiDevId,
AI_CHN AiChn, AUDIO_DEV AoDevId, AO_CHN AoChn);
```

【参数】

参数名称	描述	输入/输出
AiDevId	需要进行回声抵消的 AI 设备号。 取值范围：[0, SIO_MAX_NUM)。	输入
AiChn	需要进行回声抵消的 AI 通道号。 取值范围：[0, AIO_MAX_CHN_NUM)。	输入
AoDevId	用于回声抵消的 AO 设备号。 取值范围：[0, SIO_MAX_NUM)。	输入
AoChn	用于回声抵消的 AO 通道号。 取值范围：[0, AIO_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。

接口返回值	含义
HI_ERR_AI_INVALID_DEVID	音频输入设备号无效。
HI_ERR_AI_INVALID_CHNID	音频输入通道号无效。
HI_ERR_AI_NOT_PERM	操作非法。
HI_ERR_AI_NOT_SUPPORT	操作不支持。
HI_ERR_AI_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_aio.h、mpi_ai.h、hi_comm_aio.h
- 库文件：libmpi.a

【注意】

- 启用回声抵消前必须先启用相对应的 AI 设备。
- 成功启用回声抵消需要具备一定条件：单声道模式，采样率为 8kHz，采样精度为 16bit，帧长为 80 或 160 个采样点，且 AI 和 AO 帧长必须相同。以上条件 AI 和 AO 都必须满足（但实际上本接口只检查 AI 的属性）。
- 多次启用相同 AI、AO 的回声抵消，则返回成功。
- 目前性能只够启用一路回声抵消，但设计中没有对此做限制。

【举例】

```
HI_S32 s32ret;
AUDIO_DEV AiDevId = 1;
AI_CHN AiChn = 0;
AUDIO_DEV AoDevId = 0;
AO_CHN AoChn = 0;

s32ret = HI_MPI_AI_EnableAec(AiDevId, AiChn,AUDIO_DEV AoDevId,AO_CHN
AoChn);
if(HI_SUCCESS != s32ret)
{
    printf("enable aec err:0x%x\n", s32ret);
    return s32ret;
}

s32ret = HI_MPI_AI_DisableAec(AiDevId, AiChn)
if(HI_SUCCESS != s32ret)
{
    printf("disable aec err:0x%x\n", s32ret);
    return s32ret;
}
```

【相关主题】

无。

HI_MPI_AI_DisableAec

【描述】

禁用回声抵消功能。

【语法】

```
HI_S32 HI_MPI_AI_DisableAec(AUDIO_DEV AiDevId, AI_CHN AiChn);
```

【参数】

参数名称	描述	输入/输出
AiDevId	AI 设备号。 取值范围：[0, SIO_MAX_NUM)。	输入
AiChn	AI 通道号。 取值范围：[0, AIO_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AI_INVALID_DEVID	音频输入设备号无效。
HI_ERR_AI_INVALID_CHNID	音频输入通道号无效。
HI_ERR_AI_SYS_NOTREADY	系统未初始化成功。
HI_ERR_AI_NOT_PERM	操作非法。

【需求】

- 头文件：hi_comm_aio.h、mpi_ai.h
- 库文件：libmpi.a

【注意】

重复调用本接口则返回成功。

【举例】

参考 HI_MPI_AI_EnableAec 的举例。

【相关主题】

无。

HI_MPI_AI_GetFd

【描述】

获取音频输入通道号对应的设备文件句柄。

【语法】

```
HI_S32 HI_MPI_AI_GetFd(AUDIO_DEV AudioDevId ,AI_CHN AiChn)
```

【参数】

参数名称	描述	输入/输出
AiDevId	AI 设备号。 取值范围：[0, SIO_MAX_NUM)。	输入
AiChn	AI 通道号。 取值范围：[0, AIO_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
正数值	有效返回值。
非正数值	无效返回值。

【错误码】

无。

【需求】

- 头文件：hi_comm_aio.h、mpi_ai.h
- 库文件：libmpi.a

【注意】

无。

【举例】

```
HI_S32 s32AiFd;
HI_S32 s32ret;
AUDIO_DEV AiDevId = 1;
AI_CHN AiChn = 0;

s32AiFd = HI_MPI_AI_GetFd(AiDevId, AiChn);
if(s32AiFd <= 0)
{
    return HI_FAILURE;
}
```

【相关主题】

无。

2.3.2 音频输出

音频输出（AO）主要实现启用音频输出设备、发送音频帧到输出通道等功能。

该功能模块提供以下 MPI：

- HI_MPI_AO_SetPubAttr：设置 AO 设备属性。
- HI_MPI_AO_GetPubAttr：获取 AO 设备属性。
- HI_MPI_AO_Enable：启用 AO 设备。
- HI_MPI_AO_Disable：禁用 AO 设备。
- HI_MPI_AO_EnableChn：启用 AO 通道。
- HI_MPI_AO_DisableChn：禁用 AO 通道。
- HI_MPI_AO_PauseChn：暂停 AO 通道。
- HI_MPI_AO_ResumeChn：恢复 AO 通道。
- HI_MPI_AO_SendFrame：发送音频帧到 AO 通道。
- HI_MPI_AO_GetFd：获取 AO 通道对应的设备文件句柄。

HI_MPI_AO_SetPubAttr

【描述】

设置 AO 设备属性。

【语法】

```
HI_S32 HI_MPI_AO_SetPubAttr(AUDIO_DEV AudioDevId ,const AIO_ATTR_S
*pstAttr);
```

【参数】

参数名称	描述	输入/输出
AudioDevId	音频设备号。 取值范围：[0, SIO_MAX_NUM)。	输入
pstAttr	音频输出设备属性。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AO_INVALID_DEVID	音频输出设备号无效。
HI_ERR_AO_ILLEGAL_PARAM	输出参数无效。
HI_ERR_AO_NULL_PTR	空指针错误。
HI_ERR_AO_SYS_NOTREADY	系统未初始化成功。
HI_ERR_AO_NOT_PERM	操作非法。

【需求】

- 头文件：hi_comm_aio.h、mpi_ai.h
- 库文件：libmpi.a

【注意】

- 在设置属性之前需要保证 AO 处于非启用状态，如果处于启用状态则需要首先禁用 AO 设备。
- 在主模式下，由于同一 SIO 下的 AI、AO 设备共用同一时钟，所以如果 AO 配置的采样率和主从模式有变化则会引起相同设备号的 AI 的采样率和主从模式的相应变化。
- 扩展标志对 AO 设备无效。
- 在从模式下，采样率由 codec 决定，采样率的设置不起作用。
- 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。

- 目前版本支持主模式输出和从模式输出（实际上设置为主模式时，SIO 和 DA 都作为从设备，时钟信号由模块 CRG 提供）。
- 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】

```
HI_S32 s32ret;
AIO_ATTR_S stAttr;
AUDIO_DEV AoDevId = 0;

stAttr.enBitwidth = AUDIO_BIT_WIDTH_16;
stAttr.enSamplerate = AUDIO_SAMPLE_RATE_8000;
stAttr.enSoundmode = AUDIO_SOUND_MODE_MOMO;
stAttr.enWorkmode = AIO_MODE_I2S_SLAVE;
stAttr.u32EXFlag = 0;
stAttr.u32FrmNum = 5;
stAttr.u32PtNumPerFrm = 160;
/* set ao public attr*/
s32ret = HI_MPI_AO_SetPubAttr(AoDevId, &stAttr);
if(HI_SUCCESS != s32ret)
{
    printf("set ao %d attr err:0x%x\n", AoDevId,s32ret);
    return s32ret;
}
/* enable ao device*/
s32ret = HI_MPI_AO_Enable(AoDevId);
if(HI_SUCCESS != s32ret)
{
    printf("enable ao dev %d err:0x%x\n", AoDevId, s32ret);
    return s32ret;
}
```

【相关主题】

无。

HI_MPI_AO_GetPubAttr

【描述】

获取 AO 设备属性。

【语法】

```
HI_S32 HI_MPI_AO_GetPubAttr(AUDIO_DEV AudioDevId ,AIO_ATTR_S *pstAttr);
```

【参数】

参数名称	描述	输入/输出
AudioDevId	音频设备号。 取值范围：[0, SIO_MAX_NUM)。	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，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AO_INVALID_DEVID	音频输出设备号无效。
HI_ERR_AO_NOT_CONFIG	音频输出设备属性未配置。
HI_ERR_AO_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_aio.h、mpi_ai.h
- 库文件：libmpi.a

【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】

```
HI_S32 s32ret;
AUDIO_DEV AoDevId = 0;
AIO_ATTR_S stAttr;

/* first enable ao device*/

s32ret = HI_MPI_AO_GetPubAttr(AoDevId, &stAttr);
if(HI_SUCCESS != s32ret)
{
```

```
    printf("get ao %d attr err:0x%x\n", AoDevId,s32ret);
    return s32ret;
}
```

【相关主题】

无。

HI_MPI_AO_Enable

【描述】

启用 AO 设备。

【语法】

```
HI_S32 HI_MPI_AO_Enable(AUDIO_DEV AudioDevId);
```

【参数】

参数名称	描述	输入/输出
AudioDevId	音频设备号。 取值范围：[0, SIO_MAX_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AO_INVALID_DEVID	音频输出设备号无效。
HI_ERR_AO_SYS_NOTREADY	系统未初始化成功。
HI_ERR_AO_NOT_PERM	操作非法。

【需求】

- 头文件：hi_comm_aio.h、mpi_ai.h
- 库文件：libmpi.a

【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 设备已经处于运行状态，则直接返回成功。

【举例】

请参见 HI_MPI_AO_SetPubAttr 的举例。

【相关主题】

无。

HI_MPI_AO_Disable

【描述】

禁用 AO 设备。

【语法】

```
HI_S32 HI_MPI_AO_Disable(AUDIO_DEV AudioDevId);
```

【参数】

参数名称	描述	输入/输出
AudioDevId	音频设备号。 取值范围：[0, SIO_MAX_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AO_INVALID_DEVID	音频输出设备号无效。
HI_ERR_AO_SYS_NOTREADY	系统未初始化成功。
HI_ERR_AO_NOT_PERM	操作非法。

【需求】

- 头文件：hi_comm_aio.h、mpi_ai.h
- 库文件：libmpi.a

【注意】

- 如果 AO 设备已经处于非运行状态，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。

【相关主题】

无。

HI_MPI_AO_EnableChn

【描述】

启用 AO 通道。

【语法】

```
HI_S32 HI_MPI_AO_EnableChn(AUDIO_DEV AudioDevId, AI_CHN AoChn);
```

【参数】

参数名称	描述	输入/输出
AudioDevId	音频设备号。 取值范围：[0, SIO_MAX_NUM)。	输入
AoChn	音频输出通道号。 实际支持的通道范围由采样精度决定。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AO_INVALID_DEVID	音频输出设备号无效。

接口返回值	含义
HI_ERR_AO_INVALID_CHNID	音频输出通道号无效。
HI_ERR_AO_SYS_NOTREADY	系统未初始化成功。
HI_ERR_AO_NOT_ENABLED	音频输出设备未启用。

【需求】

- 头文件：hi_comm_aio.h、mpi_ai.h、hi_comm_aio.h
- 库文件：libmpi.a

【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】

请参见 HI_MPI_AI_SetPubAttr 的举例。

【相关主题】

无。

HI_MPI_AO_DisableChn

【描述】

禁用 AO 通道。

【语法】

```
HI_S32 HI_MPI_AO_DisableChn(AUDIO_DEV AudioDevId, AI_CHN AoChn);
```

【参数】

参数名称	描述	输入/输出
AudioDevId	音频设备号。 取值范围：[0, SIO_MAX_NUM)。	输入
AoChn	音频输出通道号。 实际支持的通道范围由采样精度决定。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AO_INVALID_DEVID	音频输出设备号无效。
HI_ERR_AO_INVALID_CHNID	音频输出通道号无效。
HI_ERR_AO_SYS_NOTREADY	系统未初始化成功。
HI_ERR_AO_NOT_PERM	操作非法。

【需求】

- 头文件：hi_comm_aio.h、mpi_ai.h、hi_comm_aio.h
- 库文件：libmpi.a

【注意】

无。

【举例】

无。

HI_MPI_AO_PauseChn

【描述】

暂停 AO 通道。

【语法】

```
HI_S32 HI_MPI_AO_PauseChn(AUDIO_DEV AudioDevId, AI_CHN AoChn);
```

【参数】

参数名称	描述	输入/输出
AudioDevId	音频设备号。 取值范围：[0, SIO_MAX_NUM)。	输入
AoChn	音频输出通道号。 实际支持的通道范围由采样精度决定。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AO_INVALID_DEVID	音频输出设备号无效。
HI_ERR_AO_INVALID_CHNID	音频输出通道号无效。
HI_ERR_AO_SYS_NOTREADY	系统未初始化成功。
HI_ERR_AO_NOT_ENABLED	音频输出设备未启用。

【需求】

- 头文件：hi_comm_aio.h、mpi_ai.h、hi_comm_aio.h
- 库文件：libmpi.a

【注意】

- AO 通道暂停后，如果继续向此通道发送音频帧数据，将会被阻塞。
- 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。

【相关主题】

无。

HI_MPI_AO_ResumeChn

【描述】

恢复 AO 通道。

【语法】

```
HI_S32 HI_MPI_AO_ResumeChn(AUDIO_DEV AudioDevId, AI_CHN AoChn);
```

【参数】

参数名称	描述	输入/输出
AudioDevId	音频设备号。 取值范围：[0, SIO_MAX_NUM)。	输入
AoChn	音频输出通道号。 实际支持的通道范围由采样精度决定。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AO_INVALID_DEVID	音频输出设备号无效。
HI_ERR_AO_INVALID_CHNID	音频输出通道号无效。
HI_ERR_AO_SYS_NOTREADY	系统未初始化成功。
HI_ERR_AO_NOT_PERM	操作非法。

【需求】

- 头文件：hi_comm_aio.h、mpi_ai.h、hi_comm_aio.h
- 库文件：libmpi.a

【注意】

- AO 通道暂停后可以通过调用此接口重新恢复，
- AO 通道为非暂停状态时不应该调用此接口。

【举例】

无。

HI_MPI_AO_SendFrame

【描述】

向音频输出通道发送数据。

【语法】

```
HI_S32 HI_MPI_AO_SendFrame(AUDIO_DEV AudioDevId ,AO_CHN AoChn, const
AUDIO_FRAME_S *pstData,HI_U32 u32Block);
```

【参数】

参数名称	描述	输入/输出
AudioDevId	音频设备号。 取值范围：[0, SIO_MAX_NUM)。	输入
AoChn	音频通道号。 取值范围：[0, AIO_MAX_CHN_NUM)。实际支持的通道范围由采样精度和主从模式决定。	输入
pstData	音频帧数据指针。	输出
u32Block	阻塞/非阻塞标志。 取值范围： • HI_IO_BLOCK：阻塞。 • HI_IO_NOBLOCK：非阻塞。 动态属性。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AO_INVALID_DEVID	音频输出设备号无效。
HI_ERR_AO_INVALID_CHNID	音频输出通道号无效。
HI_ERR_AO_NULL_PTR	空指针错误。
HI_ERR_AO_BUF_FULL	音频输出缓存满。
HI_ERR_AO_NOT_ENABLED	音频输出通道未启用。
HI_ERR_AO_SYS_NOTREADY	系统未初始化成功。

【需求】

- 头文件：hi_comm_aio.h、mpi_ai.h
- 库文件：libmpi.a

【注意】

- 发送音频帧数据之前，AO 通道必须为启用状态，否则立刻返回失败。
- 如果在发送数据过程中禁用通道，也会立刻返回失败。
- 如果 AO 通道为暂停状态，此接口将会被阻塞，直到重新恢复或通道禁用。
- 如果发送数据不及时，可能导致断音。
- 确保输入的音频数据的采样率、采样精度与 AO 设备或 DA（SIO 从模式时）的采样率、采样精度一致，否则输出的声音会有异常。

【举例】

```
AUDIO_DEV AoDevId = 0;
AO_CHN AoChn = 0;
AUDIO_FRAME_S stData;

/* get audio frame from ADEC chn or file*/

/* send audio frame to AO chn */
s32ret = HI_MPI_AO_SendFrame(AoDevId, AoChn, &stData, HI_IO_BLOCK);
if(HI_SUCCESS != s32ret)
{
    printf("send frame to ao err:0x%x\n", s32ret);
    return s32ret;
}
```

【相关主题】

无。

HI_MPI_AO_GetFd

【描述】

获取音频输出通道号对应的设备文件句柄。

【语法】

```
HI_S32 HI_MPI_AO_GetFd(AUDIO_DEV AudioDevId ,AO_CHN AoChn)
```

【参数】

参数名称	描述	输入/输出
AudioDevId	AO 设备号。 取值范围：[0, SIO_MAX_NUM)。	输入
AoChn	AO 通道号。 取值范围：[0, AIO_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
正数值	有效返回值。
非正数值	无效返回值。

【错误码】

无。

【需求】

- 头文件：hi_comm_aio.h、mpi_ai.h
- 库文件：libmpi.a

【注意】

无。

【举例】

```
HI_S32 s32AoFd;
HI_S32 s32ret;
AUDIO_DEV AoDevId = 0;
AO_CHN AoChn = 0;

/* first enable ao device */

s32AoFd = HI_MPI_AO_GetFd(AoDevId, AoChn);
if(s32AoFd <= 0)
{
    return HI_FAILURE;
}
```

【相关主题】

无。

2.3.3 音频编码

音频编码主要实现创建编码通道、发送音频帧编码及获取编码码流等功能。

该功能模块提供以下 MPI：

- HI_MPI_AENC_CreateChn：创建音频编码通道。
- HI_MPI_AENC_DestroyChn：销毁音频编码通道。
- HI_MPI_AENC_SetChnAttr：设置音频编码通道属性。
- HI_MPI_AENC_GetChnAttr：获取音频编码通道属性。
- HI_MPI_AENC_SendFrame：发送音频帧到音频编码通道。
- HI_MPI_AENC_GetStream：获取音频编码码流。
- HI_MPI_AENC_ReleaseStream：释放音频编码码流。

HI_MPI_AENC_CreateChn

【描述】

创建音频编码通道。

【语法】

```
HI_S32 HI_MPI_AENC_CreateChn(AENC_CHN AeChn, const AENC_CHN_ATTR_S
*pstAttr);
```

【参数】

参数名称	描述	输入/输出
AeChn	通道号。 取值范围：[0, AENC_MAX_CHN_NUM)。	输入
pstAttr	音频编码通道属性指针。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AENC_INVALID_CHNID	音频编码通道号无效。

接口返回值	含义
HI_ERR_AENC_NULL_PTR	空指针错误。
HI_ERR_AENC_NOT_SUPPORT	不支持的属性设置或操作。
HI_ERR_AENC_EXIST	音频编码通道已经创建。
HI_ERR_AENC_NOMEM	系统内存不足。

【需求】

- 头文件：hi_comm_aio.h、hi_comm_aenc.h、mpi_aenc.h
- 库文件：libmpi.a lib_VoiceEngine.a lib_amr_spc.a lib_amr_fipop.a lib_aec.a lib_aacdec.a lib_aacenc.a

【注意】

- 协议类型指定该通道的编码协议，目前支持 G711、G726、ADPCM 和 AMR 和 AAC，具体内容如表 2-3 所示。
- 表 2-3 中列举的编解码协议只支持 16bit 线性 PCM 音频数据处理，如果输入的是 8bit 采样精度的数据，AENC 内部会将其扩展为 16bit；另外，使用 Hi3507 AI 时，建议将扩展标志置为 1，使得 AI 数据由 8bit 自动扩展到 16bit。
- 海思语音帧结构如表 2-6 所示。
- 音频编码的部分属性需要与输入的音频数据属性相匹配，例如采样率、帧长（每帧采样点数目）等。
- buffer 大小以帧为单位，取值范围是[1, MAX_AUDIO_FRAME_NUM]。
- 在通道闲置时才能使用此接口，如果通道已经被创建，则返回通道已经创建的错误。

表2-3 Hi3507 音频编解码协议说明

协议	采样率	帧长	码率	描述
G711	8kHz	80/160/240/320/480	64	优点：语音质量最好；CPU 消耗小；支持广泛，协议免费。 缺点：压缩效率低。 G.711 提供 A 律与 μ 律压缩编码，适用于综合业务网和大多数数字电话链路。北美与日本通常采用 μ 律编码，欧洲和其他地区大都采用 A 律编码。
G726	8kHz	80/160/240/320/480	16、24、32、40	优点：算法简单；语音质量高，多次转换后语音质量有保证，能够在低比特率上达到网络等级的话音质量。 缺点：压缩效率较低。 G726_16KBPS 与 MEDIA_G726_16KBPS 两种编码器区别在于编码输出的打包格式。

				G726_16KBPS 适用于网络传输；MEDIA_G726_16KBPS 适用于 ASF 存储。请参考 RFC3551.pdf。
ADPCM	8kHz	80/160/240/320/480	32	优点：算法简单；语音质量高，多次转换后语音质量有保证，能够在低比特率上达到网络等级的话音质量。 缺点：压缩效率较低。 ADPCM_IMA 编码器需要多输入一个样点作为每帧编码的参考电平，IMA 编码每帧输入样点个数为 81/161/241/321/481。DVI 编码每帧输入样点个数为 80/160/240/320/480。
AMR-NB	8kHz	160	4.75、5.15、5.9 、 6.7 、7.4 、 7.95 、10.2、12.2	优点：语音质量好；压缩率高；支持广泛。 缺点：运算复杂，CPU 消耗大；需要收专利费。 打开 DTX，在静音情况下，可进一步降低采样率。
AAC Encoder	16kHz 22.05kHz 24kHz 32kHz 44.1kHz 48kHz	1024 、2048	详细信息如表 2-4 所示。	AAC 有两次突破性的技术升级： • aacPlus1，增加 SBR(带宽扩展)技术，使得编解码器可以在比原来少一半的比特率的条件下达到相同的音质。 • aacPlus2，增加 PS(参数立体声)技术，在低码率情况下得到极佳的音质效果，aacPlus2 可以在 48kbit/s 的速率下得到 CD 音质。 推荐比特率设置如表 2-5 所示。
AAC Decoder	兼容全部速率	1024 、2048	详细信息如表 2-4 所示。	后向兼容。传统 AAC 解码器，仅解码 aac Plus v1 码流低频信息，而 aacPlus 解码器则可以同时还原高频信息。不支持 PS 的 AAC 解码器，解码 aac Plus v2 码流时，仅能得到单声道信息，而 aacPlus2 解码器则可以得到立体声声音。
AEC	8KHz	80、160	-	2 路语音输入，分别为远端语音和本地 MIC 采集语音，经过处理后可消除本地语音里的回声。 最大尾长为 960 个样点（120ms），推荐打开 NLP 开关。

表2-4 AAC Encoder 和 AAC Decoder 的码率说明

协议	码率		
	AAC LC 编码	aacPlus v1	aacPlus v2
AAC Encoder	16kHz 16～96 22.05kHz 16～128 24kHz 16～128 32kHz 24～256 44.1kHz 32～320 48kHz 32～320	32kHz 18～23 44.1kHz 24～51 48kHz 24～51	32kHz 10～17 44.1kHz 12～35 48kHz 12～35
AAC Decoder	8kHz、11.025kHz、12kHz、16kHz、22.05kHz、24kHz、32kHz、44.1kHz、48kHz、64kHz、88.2kHz、96kHz	16kHz、22.05kHz、24kHz、32kHz、44.1kHz、48kHz	16kHz、22.05kHz、24kHz、32kHz、44.1kHz、48kHz

表2-5 AAC Encoder 协议的情况下推荐的比特率设置

采样率	LC	Plus v1	Plus v2
16kHz	48	NO	NO
22.05kHz	64	NO	NO
24kHz	64	NO	NO
32kHz	96	22	16
44.1kHz	128	48	24、32
48kHz	128	48	24、32

表2-6 海思语音帧结构

参数位置(单位：HI_S16)	参数比特位说明	参数含义
0	[15:8]	数据帧类型标志位。 01：语音帧； 其他：保留。
	[7:0]	保留。
1	[15:8]	帧循环计数器：0～255。
	[7:0]	数据净荷长度(单位：HI_S16)。

参数位置(单位：HI_S16)	参数比特位说明	参数含义
2	[15:0]	净荷数据。
3	[15:0]	净荷数据。
……	[15:0]	净荷数据。
2+n-1	[15:0]	净荷数据。
2+n	[15:0]	净荷数据。

注：n 的取值范围请参见《客户端音频编解码库 API 参考》的描述。

【举例】

```
HI_S32 s32ret;
AENC_CHN_ATTR_S stAencAttr;
AENC_ATTR_ADPCM_S stAdpcmAenc;
AENC_CHN AencChn = 0;
AUDIO_FRAME_S stAudioFrm;
AUDIO_STREAM_S stAudioStream;

stAencAttr.enType = PT_ADPCMA;/* ADPCM */
stAencAttr.u32BufSize = 8;
stAencAttr.pValue = &stAdpcmAenc;
stAdpcmAenc.enADPCMType = ADPCM_TYPE_DVI4;

/* create aenc chn*/
s32ret = HI_MPI_AENC_CreateChn(AencChn, &stAencAttr);
if (HI_SUCCESS != s32ret)
{
    printf("create aenc chn %d err:0x%x\n", AencChn,s32ret);
    return s32ret;
}
/* get audio frame form AI or file */

/* send audio frame to AENC chn */
s32ret = HI_MPI_AENC_SendFrame(AencChn, &stAudioFrm, NULL);
if (HI_SUCCESS != s32ret)
{
    printf("send frame to aenc chn %d err:0x%x\n", AencChn,s32ret);
    return s32ret;
}

/* get stream from aenc chn */
s32ret = HI_MPI_AENC_GetStream(AencChn, &stAudioStream);
```

```
if (HI_SUCCESS != s32ret )
{
    printf("get stream from aenc chn %d fail \n", AencChn);
    return s32ret;
}

/* deal with audio stream */

/* release audio stream */
s32ret = HI_MPI_AENC_ReleaseStream(AencChn, &stAudioStream);
if (HI_SUCCESS != s32ret )
{
    return s32ret;
}

/* destroy aenc chn */
s32ret = HI_MPI_AENC_DestroyChn(AencChn);
if (HI_SUCCESS != s32ret )
{
    return s32ret;
}
```

【相关主题】

无。

HI_MPI_AENC_DestroyChn

【描述】

销毁音频编码通道。

【语法】

```
HI_S32 HI_MPI_AENC_DestroyChn(AENC_CHN AeChn);
```

【参数】

参数名称	描述	输入/输出
AeChn	通道号。 取值范围：[0, AENC_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。

返回值	描述
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AENC_INVALID_CHNID	音频编码通道号无效。
HI_ERR_AENC_UNEXIST	编码通道不存在。

【需求】

- 头文件：hi_comm_aio.h、hi_comm_aenc.h、mpi_aenc.h
- 库文件：libmpi.a lib_VoiceEngine.a lib_amr_spc.a lib_amr_fipop.a lib_aec.a

【注意】

- 先创建完编码通道，再调用此接口，否则返回通道未创建。
- 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。

【举例】

请参见 HI_MPI_AENC_CreateChn 的举例。

【相关主题】

无。

HI_MPI_AENC_SetChnAttr

【描述】

设置音频编码通道属性。

【语法】

```
HI_S32 HI_MPI_AENC_SetChnAttr(AENC_CHN AeChn, const AENC_CHN_ATTR_S *
pstAttr);
```

【参数】

参数名称	描述	输入/输出
AeChn	通道号。 取值范围：[0, AENC_MAX_CHN_NUM)。	输入
pstAttr	音频编码通道属性指针。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AENC_INVALID_CHNID	音频编码通道号无效。
HI_ERR_AENC_UNEXIST	音频编码通道不存在。
HI_ERR_AENC_NULL_PTR	空指针错误。
HI_ERR_AENC_NOT_PERM	操作不允许。
HI_ERR_AENC_NOT_SUPPORT	不支持的属性设置或操作。

【需求】

- 头文件：hi_comm_aio.h、hi_comm_aenc.h、mpi_aenc.h
- 库文件：libmpi.a lib_VoiceEngine.a lib_amr_spc.a lib_amr_fipop.a lib_aec.a

【注意】

- 先创建完通道，再设置属性。如果通道未被创建，则返回失败。
- 不支持静态属性（例如编码协议类型和 buffer 大小）的修改。
- 由于目前的协议没有动态属性，所以此接口无实际意义，仅仅是为了兼容可能存在的动态属性。

【举例】

无。

【相关主题】

无。

HI_MPI_AENC_GetChnAttr

【描述】

获取音频编码通道属性。

【语法】

```
HI_S32 HI_MPI_AENC_GetChnAttr(AENC_CHN AeChn, AENC_CHN_ATTR_S * pstAttr);
```

【参数】

参数名称	描述	输入/输出
AeChn	通道号。 取值范围：[0, AENC_MAX_CHN_NUM)。	输入
pstAttr	音频编码通道属性指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AENC_INVALID_CHNID	音频编码通道号无效。
HI_ERR_AENC_UNEXIST	音频编码通道不存在。
HI_ERR_AENC_NULL_PTR	空指针错误。
HI_ERR_AENC_NOT_SUPPORT	不支持的属性设置或操作。

【需求】

- 头文件：hi_comm_aio.h、hi_comm_aenc.h、mpi_aenc.h
- 库文件：libmpi.a lib_VoiceEngine.a lib_amr_spc.a lib_amr_fipop.a lib_aec.a

【注意】

获取属性前必须保证通道已经被创建，如果通道未被创建则直接返回失败。

【举例】

```
HI_S32 s32ret;
AENC_CHN_ATTR_S stAencAttr;
AENC_ATTR_ADPCM_S stAdpcmAenc;
AENC_CHN AencChn = 0;
```

```
/* first create aenc chn */

stAencAttr.enType = PT_ADPCMA;
stAencAttr.pValue = &stAdpcmAenc;

s32ret = HI_MPI_AENC_GetChnAttr(AencChn, &stAencAttr);
if (HI_SUCCESS != s32ret)
{
    return s32ret;
}
```

【相关主题】

无。

HI_MPI_AENC_SendFrame

【描述】

向音频编码通道发送音频帧。

【语法】

```
HI_S32 HI_MPI_AENC_SendFrame(AENC_CHN AeChn, const AUDIO_FRAME_S *
pstData, const AUDIO_FRAME_S * pstRefData);
```

【参数】

参数名称	描述	输入/输出
AeChn	通道号。 取值范围：[0, AENC_MAX_CHN_NUM)。	输入
pstData	音频帧数据。	输入
pstRefData	回声抵消参考帧。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AENC_INVALID_CHNID	音频编码通道号无效。
HI_ERR_AENC_UNEXIST	音频编码通道号不存在。
HI_ERR_AENC_NULL_PTR	空指针错误。
HI_ERR_AENC_ILLEGAL_PARAM	输入参数无效。
HI_ERR_AENC_ENCODER_ERR	音频编码器内部错误。

【需求】

- 头文件：hi_comm_aio.h、hi_comm_aenc.h、mpi_aenc.h
- 库文件：libmpi.a lib_VoiceEngine.a lib_amr_spc.a lib_amr_fipop.a lib_aec.a

【注意】

- 音频帧数据包含完整的帧信息：数据指针、帧长、采样率、采样精度、时间戳和帧号。
- 由于编码算法的约束，推荐使用 8kHz 采样数据进行编码，否则音质效果和码率会受到影响。
- 采样宽度支持 8bit 和 16bit，8bit 的数据在内部会被扩展为 16bit 进行处理。
- 语音编码支持帧长为 80、160、240、320、480、1024、2048 个采样点的数据编码，AMR 仅支持帧长为 160 个采样点的数据编码，AACLC 仅支持 1024 的帧长，eAAC、eAAC+仅支持 2048 的帧长。
- 帧长 len 是指数据 buffer 中单个声道的数据长度（以 Byte 为单位），立体声则表示左右声道均为 len 的长度。len 的最大长度由 MAX_AUDIO_FRAME_LEN 指定。
- 输入的数据可以是从 AI 直接获取的数据，也可以是从文件读取的数据，只要遵循数据存放规则（单声道数据直接存放，采样点数为 ptnum，长度为 len；立体声数据按左右声道分开存放，先存放采样点为 ptnum，长度为 len 的左声道数据，然后存放采样点为 ptnum，长度为 len 的右声道数据），就可以进行编码。
- 不需要做回声抵消则可以将参数 pstRefData 置为空。回声抵消需要原始帧和参考帧满足以下条件：采样率为 8kHz、采样精度为 8bit 或 16bit、帧长为 80 或 160、时间戳差值小于 120ms，否则不做回声抵消。
- 发送数据时必须保证通道已经被创建，否则直接返回失败，如果在发送数据过程中销毁通道则会立刻返回失败。
- 此接口为阻塞接口，如果音频数据 Buffer 满则此接口调用会被阻塞，直至取走音频数据或销毁 AENC 通道。

【举例】

请参见 HI_MPI_AENC_CreateChn 的举例。

【相关主题】

无。

HI_MPI_AENC_GetStream

【描述】

获取编码后码流。

【语法】

```
HI_S32 HI_MPI_AENC_GetStream(AENC_CHN AeChn, AUDIO_STREAM_S *pstStream);
```

【参数】

参数名称	描述	输入/输出
AeChn	通道号。 取值范围：[0, AENC_MAX_CHN_NUM)。	输入
pstStream	获取的音频码流。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AENC_INVALID_CHNID	音频编码通道号无效。
HI_ERR_AENC_UNEXIST	音频编码通道不存在。
HI_ERR_AENC_NULL_PTR	空指针错误。

【需求】

- 头文件：hi_comm_aio.h、hi_comm_aenc.h、mpi_aenc.h
- 库文件：libmpi.a lib_VoiceEngine.a lib_amr_spc.a lib_amr_fipop.a lib_aec.a

【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。

- 此接口为阻塞接口，如果音频数据 Buffer 空则此接口调用会被阻塞，直至 Buffer 中有新的数据或销毁 AENC 通道。

【举例】

请参见 HI_MPI_AENC_CreateChn 的举例。

【相关主题】

无。

HI_MPI_AENC_ReleaseStream

【描述】

释放从音频编码通道获取的码流。

【语法】

```
HI_S32 HI_MPI_AENC_ReleaseStream(AENC_CHN AeChn, const AUDIO_STREAM_S
*pstStream);
```

【参数】

参数名称	描述	输入/输出
AeChn	通道号。 取值范围：[0, AENC_MAX_CHN_NUM)。	输入
pstStream	获取的码流指针。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AENC_INVALID_CHNID	音频编码通道号无效。
HI_ERR_AENC_UNEXIST	音频编码通道不存在。
HI_ERR_AENC_NULL_PTR	空指针错误。

【需求】

- 头文件：hi_comm_aio.h、hi_comm_aenc.h、mpi_aenc.h
- 库文件：libmpi.a lib_VoiceEngine.a lib_amr_spc.a lib_amr_fipop.a lib_aec.a

【注意】

- 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等待码流释放。
- 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】

请参见 HI_MPI_AENC_CreateChn 的举例。

【相关主题】

无。

2.3.4 音频解码

音频解码主要实现创建解码通道、发送音频码流解码及获取解码后音频帧等功能。

该功能模块提供以下 MPI：

- HI_MPI_ADEC_CreateChn：创建音频解码通道。
- HI_MPI_ADEC_DestroyChn：销毁音频解码通道。
- HI_MPI_ADEC_SetChnAttr：设置音频解码通道属性。
- HI_MPI_ADEC_GetChnAttr：获取音频解码通道属性。
- HI_MPI_ADEC_SendStream：发送音频码流到音频解码通道。
- HI_MPI_ADEC_GetData：获取解码后音频帧数据。
- HI_MPI_ADEC_ReleaseData：释放解码后音频帧数据。

HI_MPI_ADEC_CreateChn

【描述】

创建音频解码通道。

【语法】

```
HI_S32 HI_MPI_ADEC_CreateChn(ADEC_CHN AdChn, ADEC_CHN_ATTR_S *pstAttr);
```

【参数】

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

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_ADEC_INVALID_CHNID	音频解码通道号无效。
HI_ERR_ADEC_NULL_PTR	空指针错误。
HI_ERR_ADEC_NOT_SUPPORT	不支持的属性设置或操作。
HI_ERR_ADEC_EXIST	音频解码通道已经创建。
HI_ERR_ADEC_NOMEM	系统内存不足。

【需求】

- 头文件：hi_comm_aio.h、hi_comm_adec.h、mpi_adec.h
- 库文件：libmpi.a lib_VoiceEngine.a lib_amr_spc.a lib_amr_fipop.a lib_aec.a

【注意】

- 协议类型指定了该通道的解码协议，目前支持 G711、G726、ADPCM 和 AMR 和 AAC。各种音频编解码协议的详细说明请参见“2.3.3 音频编码”。）
- 音频解码的部分属性需要与输出设备属性相匹配，例如采样率、帧长（每帧采样点数目）等。
- buffer 大小以帧为单位，取值范围是[0, MAX_AUDIO_FRAME_NUM]。
- 在通道未创建前（或销毁后）才能使用此接口，如果通道已经被创建，则返回通道已经创建。

【举例】

```
HI_S32 s32ret;
```

```
ADEC_CHN_ATTR_S stAdecAttr;
ADEC_ATTR_ADPCM_S stAdpcm;
ADEC_CHN AdChn = 0;
AUDIO_STREAM_S stAudioStream;
AUDIO_FRAME_INFO_S stAudioFrameInfo;

stAdecAttr.enType = PT_ADPCMA;
stAdecAttr.u32BufSize = 8;
stAdecAttr.enMode = ADEC_MODE_STREAM;
stAdecAttr.pValue = &stAdpcm;
stAdpcm.enADPCMType = ADPCM_TYPE_DVI4;

/* create adec chn*/
s32ret = HI_MPI_ADEC_CreateChn(AdChn, &stAdecAttr);
if (s32ret)
{
    printf("create adnc chn %d err:0x%x\n", AdChn,s32ret);
    return s32ret;
}

/* get audio stream from network or file*/

/* send audio stream to adec chn */
s32ret = HI_MPI_ADEC_SendStream(AdChn, &stAudioStream);
if (s32ret)
{
    printf("send stream to adec fail\n");
    return s32ret;
}

/* get audio frame from adec */
s32ret = HI_MPI_ADEC_GetData(AdChn, &stAudioFrameInfo);
if (HI_SUCCESS != s32ret)
{
    printf("adec get data err\n");
    return s32ret;
}

/* send audio frame to AO or others */

/* release audio frame */
s32ret = HI_MPI_ADEC_ReleaseData(AdChn, &stAudioFrameInfo);
if (HI_SUCCESS != s32ret)
{
```

```
    printf("adec release data err\n");
    return s32ret;
}

/* destroy adec chn */
s32ret = HI_MPI_ADEC_DestroyChn(AdChn);
if (HI_SUCCESS != s32ret )
{
    return s32ret;
}
```

【相关主题】

无。

HI_MPI_ADEC_DestroyChn

【描述】

销毁音频解码通道。

【语法】

```
HI_S32 HI_MPI_ADEC_DestroyChn(ADEC_CHN AdChn);
```

【参数】

参数名称	描述	输入/输出
AdChn	通道号。 取值范围：[0, ADEC_MAX_CHN_NUM)。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_ADEC_INVALID_CHNID	音频解码通道号无效。
HI_ERR_ADEC_UNEXIST	音频解码通道不存在。

【需求】

- 头文件：hi_comm_aio.h、hi_comm_adec.h、mpi_adec.h
- 库文件：libmpi.a lib_VoiceEngine.a lib_amr_spc.a lib_amr_fipop.a lib_aec.a

【注意】

- 要求解码通道已经被创建，如果通道未被创建则返回通道未创建。
- 如果正在获取/释放码流或者发送帧，销毁该通道则这些操作都会立即返回失败。

【举例】

请参见 HI_MPI_ADEC_CreateChn 的举例。

【相关主题】

无。

HI_MPI_ADEC_SetChnAttr

【描述】

设置音频解码通道属性。

【语法】

```
HI_S32 HI_MPI_ADEC_SetChnAttr(ADEC_CHN AdChn, ADEC_CHN_ATTR_S * pstAttr);
```

【参数】

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

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。

接口返回值	含义
HI_ERR_ADEC_INVALID_CHNID	音频解码通道号无效。
HI_ERR_ADEC_UNEXIST	音频解码通道不存在。
HI_ERR_ADEC_NULL_PTR	空指针错误。
HI_ERR_ADEC_NOT_PERM	操作非法。
HI_ERR_ADEC_NOT_SUPPORT	不支持的属性设置或操作。

【需求】

- 头文件：hi_comm_aio.h、hi_comm_adec.h、mpi_adec.h
- 库文件：libmpi.a lib_VoiceEngine.a lib_amr_spc.a lib_amr_fipop.a lib_aec.a

【注意】

- 先创建完通道，再设置属性。如果通道未被创建，则返回失败。
- 由于目前的协议没有动态属性，所以此接口无实际意义，仅仅是为了兼容可能存在的动态属性。

【举例】

无。

【相关主题】

无。

HI_MPI_ADEC_GetChnAttr

【描述】

获取音频解码通道属性。

【语法】

```
HI_S32 HI_MPI_ADEC_GetChnAttr(ADEC_CHN AdChn, ADEC_CHN_ATTR_S * pstAttr);
```

【参数】

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

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_ADEC_INVALID_CHNID	音频编码通道号无效。
HI_ERR_ADEC_UNEXIST	音频编码通道不存在。
HI_ERR_ADEC_NULL_PTR	空指针错误。
HI_ERR_ADEC_NOT_SUPPORT	不支持的属性设置或操作。

【需求】

- 头文件：hi_comm_aio.h、hi_comm_adec.h、mpi_adec.h
- 库文件：libmpi.a lib_VoiceEngine.a lib_amr_spc.a lib_amr_fipop.a lib_aec.a

【注意】

获取属性前必须保证通道已经被创建，如果通道未被创建则直接返回失败。

【举例】

```
HI_S32 s32ret;
ADEC_CHN_ATTR_S stAdecAttr;
ADEC_ATTR_ADPCM_S stAdpcmAdec;
ADEC_CHN AdecChn = 0;

stAdecAttr.enType = PT_ADPCMA;
stAdecAttr.pValue = &stAdpcmAdec;

s32ret = HI_MPI_AENC_GetChnAttr(AdecChn, &stAdecAttr);
if (HI_SUCCESS != s32ret)
{
    return s32ret;
}
```

【相关主题】

无。

HI_MPI_ADEC_SendStream

【描述】

向音频解码通道发送码流。

【语法】

```
HI_S32 HI_MPI_ADEC_SendStream(ADEC_CHN AdChn, const AUDIO_STREAM_S
*pstStream) ;
```

【参数】

参数名称	描述	输入/输出
AdChn	通道号。 取值范围：[0, ADEC_MAX_CHN_NUM)。	输入
pstStream	音频码流。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_ADEC_INVALID_CHNID	音频解码通道号无效。
HI_ERR_ADEC_UNEXIST	音频解码通道号不存在。
HI_ERR_ADEC_NULL_PTR	空指针错误。
HI_ERR_ADEC_ILLEGAL_PARAM	输入参数无效。
HI_ERR_ADEC_DECODER_ERR	音频解码器内部错误。

【需求】

- 头文件：hi_comm_aio.h、hi_comm_adec.h、mpi_adec.h
- 库文件：libmpi.a lib_VoiceEngine.a lib_amr_spc.a lib_amr_fipop.a lib_aec.a

【注意】

- 创建解码通道时可以指定解码方式为 pack 方式或 stream 方式。
  - pack 方式用于确定码流包为一帧的情况下，比如从 AENC 直接获取的码流，从文件读取确切知道一帧边界（语音编码码流长度固定，很容易确定边界）的码流，效率较高。
  - stream 方式用于不确定码流包为一帧的情况下，效率较低。
- 发送数据时必须保证通道已经被创建，否则直接返回失败，如果在送数据过程中销毁通道则会立刻返回失败。
- 此接口为阻塞接口，如果音频数据 Buffer 满则此接口调用会被阻塞，直至取走音频数据或销毁 ADEC 通道。

【举例】

请参见 HI_MPI_ADEC_CreateChn 的举例。

【相关主题】

无。

HI_MPI_ADEC_GetData

【描述】

获取解码后音频帧。

【语法】

```
HI_S32 HI_MPI_ADEC_GetData(ADEC_CHN AdChn, AUDIO_FRAME_INFO_S *pstFrm);
```

【参数】

参数名称	描述	输入/输出
AdChn	通道号。 取值范围：[0, ADEC_MAX_CHN_NUM)。	输入
pstFrm	获取的音频帧数据。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_ADEC_INVALID_CHNID	音频解码通道号无效。
HI_ERR_ADEC_UNEXIST	音频解码通道不存在。
HI_ERR_ADEC_NULL_PTR	空指针错误。

【需求】

- 头文件：hi_comm_aio.h、hi_comm_adec.h、mpi_adec.h
- 库文件：libmpi.a lib_VoiceEngine.a lib_amr_spc.a lib_amr_fipop.a lib_aec.a

【注意】

- 获取音频帧数据时必须保证通道已经被创建，否则直接返回失败，如果在获取音频帧数据过程中销毁通道则会立刻返回失败。
- 此接口为阻塞接口，如果音频数据 Buffer 空则此接口调用会被阻塞，直至 Buffer 中有新的数据或销毁 ADEC 通道。

【举例】

请参见 HI_MPI_AENC_CreateChn 的举例。

【相关主题】

无。

HI_MPI_ADEC_ReleaseData

【描述】

释放获取的解码帧数据。

【语法】

```
HI_S32 HI_MPI_ADEC_ReleaseData(ADEC_CHN AdChn, AUDIO_FRAME_INFO_S
*pstFrm);
```

【参数】

参数名称	描述	输入/输出
AdChn	通道号。 取值范围：[0, ADEC_MAX_CHN_NUM)。	输入
pstFrm	获取的音频帧信息。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_ADEC_INVALID_CHNID	音频解码通道号无效。
HI_ERR_ADEC_UNEXIST	音频解码通道不存在。
HI_ERR_ADEC_NULL_PTR	空指针错误。

【需求】

- 头文件：hi_comm_aio.h、hi_comm_adec.h、mpi_adec.h
- 库文件：libmpi.a lib_VoiceEngine.a lib_amr_spc.a lib_amr_fipop.a lib_aec.a

【注意】

- 音频帧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等待音频帧释放。
- 释放的音频帧必须是从该通道获取的码流，不得对帧信息结构体进行任何修改，否则会导致音频帧不能释放，使此音频帧丢失，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，如果在释放帧数据过程中销毁通道则会立刻返回失败。

【举例】

请参见 HI_MPI_AENC_CreateChn 的举例。

【相关主题】

无。

2.4 音视频复合功能函数参考

音视频复合功能实现创建、启动音视频复合编码通道以及获取编码码流等功能。

该功能模块提供以下 MPI：

- HI_MPI_AVENC_CreatCH：创建音视频复合编码通道。
- HI_MPI_AVENC_DestroyCH：销毁音视频复合编码通道。
- HI_MPI_AVENC_StartCH：启动音视频复合编码通道。

- HI_MPI_AVENC_StopCH：停止指定的音视频复合编码通道。
- HI_MPI_AVENC_GetStream：获取指定的音视频复合编码通道的编码码流。
- HI_MPI_AVENC_ReleaseStream：释放获取的音视频复合编码通道的编码码流缓存。

HI_MPI_AVENC_CreatCH

【描述】

创建音视频复合编码通道。

【语法】

```
HI_S32 HI_MPI_AVENC_CreatCH (AUDIO_DEV AudioDevId ,AI_CHN AiChn, VENC_CHN VChnId,AENC_CHN AChnId,AVENC_CHN *pAVChnId);
```

【参数】

参数名称	描述	输入/输出
AudioDevId	音频输入设备号。 取值范围：[0, SIO_MAX_NUM)。	输入
AiChn	音频输入通道号。 取值范围：[0, AIO_MAX_CHN_NUM)。	输入
VChnId	要复合的视频编码通道号。 取值范围：[0, VENC_MAX_CHN_NUM)。	输入
AChnId	要复合的音频编码通道号。 取值范围：[0, AENC_MAX_CHN_NUM)。	输入
pAVChnId	创建的复合编码通道号的指针。	输出

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AVENC_INVALID_PTR	空指针错误。

接口返回值	含义
HI_ERR_AVENC_INVALID_CHN	音视频通道号无效。
HI_ERR_AVENC_VCHN_USING	视频编码通道对应的复合通道已经存在。
HI_ERR_AVENC_ACHN_USING	音频编码通道对应的复合通道已经存在。
HI_FAILURE	内部错误。

【需求】

- 头文件：mpi_avenc.h、hi_comm_avenc.h
- 库文件：libmpi.a

【注意】

- 不允许重复创建相同视频编码通道号的复合通道，但允许创建相同音频编码通道号的复合通道，以实现视频的双码流对应同一音频编码通道。
- 可以创建仅包含音频或视频编码通道的复合通道，例如创建仅包含音频编码通道的复合通道，此时将视频编码通道传入–1 即可。

【举例】

```
HI_S32 s32Ret;
AUDIO_DEV AudioDevId;
AI_CHN AiChn;
AENC_CHN AencChn;
VENC_CHN VencChn;
AVENC_CHN AVEncChn;

/* 示例1： 创建的复合编码通道同时进行音频和视频编码 */
/* 创建复合编码通道之前，先创建音频编码通道AencChn和视频编码通道VencChn*/
s32Ret = HI_MPI_AVENC_CreatCH(AudioDevId, AiChn,VencChn, AencChn,
&AVEncChn);
if (HI_SUCCESS != s32Ret)
{
    printf("A&V channel create error!\n");
    return HI_FAILURE;
}

/* 示例2： 创建的复合编码通道只进行视频编码。 视频编码通道VencChn需要事先创建 */
s32Ret = HI_MPI_AVENC_CreatCH(-1,-1,VencChn, -1, &AVEncChn);
if (HI_SUCCESS != s32Ret)
{
    printf("A&V channel create error!\n");
    return HI_FAILURE;
```

```
}
/* 示例3： 创建的复合编码通道只进行音频编码。 音频编码通道AencChn需要事先创建 */
s32Ret = HI_MPI_AVENC_CreatCH(AudioDevId, AiChn,-1, AencChn, &AVEncChn);
HI_SUCCESS != s32Ret)
{
    printf("A&V channel create error!\n");
    return HI_FAILURE;
}
```

【相关主题】

无。

HI_MPI_AVENC_DestroyCH

【描述】

销毁音视频复合编码通道。

【语法】

```
HI_S32 HI_MPI_AVENC_DestroyCH(AVENC_CHN AVChnId, HI_BOOL bFlag);
```

【参数】

参数名称	描述	输入/输出
AVChnId	销毁的音视频复合编码通道号。 取值范围：[0, AVENC_CHN_MAXNUM)。	输入
bFlag	强制销毁标志。 取值范围： • HI_TRUE：不管该通道是否正在运行，强制销毁。 • HI_FALSE：若该通道正在运行，则不销毁，返回失败。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AVENC_INVALID_CHN	复合编码通道无效。
HI_ERR_AVENC_CHN_NOT_CREATE	复合编码通道未创建。
HI_ERR_AVENC_CHN_NOT_STOP	复合编码通道有正在运行的编码任务。
HI_ERR_AVENC_BUFFER_NOTFREE	复合编码通道的缓冲区满。

【需求】

- 头文件：mpi_avenc.h、hi_comm_avenc.h
- 库文件：libmpi.a

【注意】

- 销毁复合通道对复合通道中绑定的音频或视频通道本身无影响，如需要销毁音频或视频通道，需要在销毁复合通道后，继续销毁音频或视频通道。
- 在销毁复合通道前，应该释放已经获取的码流，并停止复合通道。
- 如果采用非强制销毁的方式，当复合通道未停止，或用户持有码流而未释放，将无法完成销毁。
- 如果采用强制销毁的方式，不管复合通道是否停止，码流缓存是否释放都将销毁复合通道，但这可能会导致潜在的危险发生。通常建议用户按照正常流程停止复合通道、释放持有的码流后使用非强制销毁方式。尽量不要采用强制销毁方式。

【举例】

```
HI_S32 s32Ret;
AVENC_CHN AVEncChn;

/* 销毁音视频复合编码通道AVEncChn ，如果用户正在使用码流，销毁将会失败*/
s32Ret = HI_MPI_AVENC_DestroyCH(AVEncChn, HI_FALSE);
if(HI_ERR_AVENC_BUFFER_NOTFREE == s32Ret)
{
    printf("Some AVENC buffer may be hold by user! Be sure to release
all"\
        "buffer! Otherwise some exception may arose!\n");

    /* 如果希望忽略此错误，且确信不会再使用复合码流数据，可以强制销毁编码通道. */
    HI_MPI_AVENC_DestroyCH(AVEncChn, HI_TRUE);
    return HI_FAILURE;
}
```

【相关主题】

无。

HI_MPI_AVENC_StartCH

【描述】

启动音视频复合编码通道。

【语法】

```
HI_S32 HI_MPI_AVENC_StartCH(AVENC_CHN AVChnId,
AVENC_OPERATE_OBJECT_E enObject,
HI_S32 s32Priority);
```

【参数】

参数名称	描述	输入/输出
AVChnId	要启动复合编码的通道号。	输入
enObject	启动的操作对象。	输入
s32Priority	编码通道的优先级，目前仅保留接口形式，参数未使用。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESSS	成功。
HI_ERR_AVENC_INVALID_CHN	复合编码通道无效。
HI_ERR_AVENC_INVALID_OBJECT	操作对象无效。
HI_ERR_AVENC_CHN_NOT_CREATE	复合编码通道未创建。
HI_ERR_AVENC_CHN_RESTART	复合编码通道已经启动。
HI_ERR_AVENC_VIDEO_NOT_EXIST	视频通道不存在。
HI_ERR_AVENC_AUDIO_NOT_EXIST	音频通道不存在。
HI_ERR_AVENC_VIDEO_USEDBYOTHER	视频编码通道已经被单独启动过。

接口返回值	含义
HI_ERR_AVENC_AUDIO_USEDBYOTHER	音频编码通道已经被单独启动过。
HI_FAILURE	内部错误。

【需求】

- 头文件：mpi_avenc.h、hi_comm_avenc.h
- 库文件：libmpi.a

【注意】

- 成功创建复合编码通道后，才能够调用本接口启动复合编码通道。启动成功与否取决于创建通道时的操作对象和启动通道时的操作对象是否一致，例如创建时只指定操作音频通道，那么启动时指定操作视频通道必然将会返回失败。
- 启动复合编码通道，其中绑定的视频编码通道会被启动并从视频编码通道获取码流（用户需自行将编码通道和视频输入通道绑定）；绑定的音频编码通道会从音频输入通道获取音频帧进行编码（用户需保证音频输入设备已经启用），并从音频编码通道获取码流。
- 对于被复合通道绑定的音视频通道，用户不应该再单独操作（例如送数据编码或获取码流），而应该使用复合通道的相关接口进行操作，否则将造成编码数据和操作的异常。

【举例】

```
HI_S32 s32Ret;
AVENC_CHN AVEncChn;

/* 示例1： 同时启动音频及视频编码，获取码流时将能够取得同步后的音视频数据*/
s32Ret = HI_MPI_AVENC_StartCH(AVEncChn, AVENC_OPERATION_BOTH, 0);
if(HI_SUCCESS != s32Ret)
{
    printf("A&V channel can't be started! Please check whether the VENC
"\
        "and AENC channel is usable!\n");
    return HI_FAILURE;
}

/* 示例2： 单独启动视频编码，获取码流时将得到的单独的视频数据*/
s32Ret = HI_MPI_AVENC_StartCH(AVEncChn, AVENC_OPERATION_VIDEO, 0);
if(HI_SUCCESS != s32Ret)
{
  printf("A&V channel can't be started! Please check whether the VENC "\
        "and AENC channel is usable!\n");
  return HI_FAILURE;
```

```
}

/* 示例3： 单独启动音频编码，获取码流时将得到的单独的音频数据*/
s32Ret = HI_MPI_AVENC_StartCH(AVEncChn, AVENC_OPERATION_AUDIO, 0);
if(HI_SUCCESS != s32Ret)
{
  printf("A&V channel can't be started! Please check whether the VENC "\
        "and AENC channel is usable!\n");
  return HI_FAILURE;
}
```

【相关主题】

无。

HI_MPI_AVENC_StopCH

【描述】

停止指定的音视频复合编码通道。

【语法】

```
HI_S32 HI_MPI_AVENC_StopCH(AVENC_CHN AVChnId, AVENC_OPERATE_OBJECT_E
enObject);
```

【参数】

参数名称	描述	输入/输出
AVChnId	要停止复合编码的通道号。 取值范围：[0, AVENC_CHN_MAXNUM)。	输入
enObject	停止的操作对象。 取值范围： • AVENC_OPERATION_BOTH：停止视频通道和音频通道。 • AVENC_OPERATION_AUDIO：仅停止音频通道。 • AVENC_OPERATION_VIDEO：仅停止视频通道。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AVENC_INVALID_CHN	复合编码通道无效。
HI_ERR_AVENC_INVALID_OBJECT	操作对象无效。
HI_ERR_AVENC_CHN_NOT_START	通道未启动。
HI_ERR_AVENC_VIDEO_NOT_EXIST	视频通道不存在。
HI_ERR_AVENC_AUDIO_NOT_EXIST	音频通道不存在。
HI_FAILURE	内部错误。

【需求】

- 头文件：mpi_avenc.h、hi_comm_avenc.h
- 库文件：libmpi.a

【注意】

必须先启动复合编码通道后才能停止复合编码通道。

【举例】

```
HI_S32 s32Ret;
AVENC_CHN AVEncChn;

/* 示例1： 同时停止音频及视频编码，可以调用HI_AVENC_StartCH 再次启动 */
s32Ret = HI_MPI_AVENC_StopCH(AVEncChn, AVENC_OPERATION_BOTH);
if(HI_SUCCESS != s32Ret)
{
  printf("A&V channel can't be stoped! \n");
  return HI_FAILURE;
}

/*示例2： 单独停止视频编码，获取码流时将只能得到单独的音频数据*/
s32Ret = HI_MPI_AVENC_StopCH(AVEncChn, AVENC_OPERATION_VIDEO);
if(HI_SUCCESS != s32Ret)
{
  printf("A&V channel can't be stoped!\n");
  return HI_FAILURE;
}

/* 示例3： 单独停止音频编码，获取码流时将只能得到的单独的视频数据*/
```

```
s32Ret = HI_MPI_AVENC_StopCH(AVEncChn, AVENC_OPERATION_AUDIO);
if(HI_SUCCESS != s32Ret)
{
  printf("A&V channel can't be stoped!\n");
  return HI_FAILURE;
}
```

【相关主题】

无。

HI_MPI_AVENC_GetStream

【描述】

获取指定的音视频复合编码通道的编码码流。

【语法】

```
HI_S32 HI_MPI_AVENC_GetStream(AVENC_CHN AVChnId, AVENC_STREAM_S
*pStream,HI_U32 u32Block);
```

【参数】

参数名称	描述	输入/输出
AVChnId	复合编码通道。 取值范围：[0, AVENC_CHN_MAXNUM)。	输入
pStream	输出复合编码码流指针。	输出
u32Block	码流获取阻塞标志。 取值范围： • HI_IO_BLOCK：阻塞。 • HI_IO_NOBLOCK：非阻塞。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AVENC_INVALID_CHN	复合编码通道无效。
HI_ERR_AVENC_INVALID_PTR	空指针错误。
HI_ERR_AVENC_CHN_NOT_START	通道未启动。
HI_ERR_AVENC_NO_STREAM	复合流缓冲队列中没有数据。
HI_FAILURE	内部错误。

【需求】

- 头文件：mpi_avenc.h、hi_comm_avenc.h
- 库文件：libmpi.a

【注意】

获取编码码流之前必须先启动复合编码通道。

【举例】

```
HI_S32          s32Ret;
AVENC_CHN       AVEncChn;
AVENC_STREAM_S  struStream;
AUDIO_STREAM_S  *pAStream   = NULL;
VENC_STREAM_S   *pVStream   = NULL;
AVENC_LIST_S    *pNode      = NULL;
FILE            *pVideoFile = NULL;
FILE            *pAudioFile = NULL;
HI_U32 i, j, writelen;

/* 获取音视频复合数据，并将数据存入文件 */
s32Ret = HI_MPI_AVENC_GetStream(AVEncChn, &struStream, HI_IO_BLOCK);

/* 首先将同步分组中的音频数据存入文件 */
pNode = struStream.pAudioListHead;
while(NULL != pNode)
{
    pAStream = (AUDIO_STREAM_S *)(pNode->pData);

    /* 存储音频编码单元的时间戳*/
    fwrite(&(pAStream->u64TimeStamp), sizeof(HI_U64), 1, pFile);

    /* 存储实际的编码数据 */
```

```
    fwrite(pAStream->pStream, pAStream->u32Len, 1, pFile);

    /* 音频链表中可能存在多个音频编码单元，逐一遍历 */
    pNode = pNode->pNext;
}

/* 再将同步分组中的视频数据存入文件 */
pNode = struStream.pVideoListHead;
while(NULL != pNode)
{
    pVStream = (VENC_STREAM_S *)(pNode->pData);

    /* 存储这个视频帧的时间戳 */
    fwrite(&(pVStream->struDataInfo.u64PTS), sizeof(HI_U64), 1, pFile);

    /* 一个视频帧由多个分片组成，顺序存储所有分片 */
    for ( i = 0 ; i < pVStream->u32PackCount; i++ )
    {
        fwrite(pVStream->pstPack[i].pu8Addr[0], pVStream-
>pstPack[i].u32Len[0], 1, pFile);
        fflush(pFile);
        if (pVStream->pstPack[i].u32Len[1] > 0)
        {
            fwrite(pVStream->pstPack[i].pu8Addr[1], pVStream-
>pstPack[i].u32Len[1], 1, pFile);
            fflush(pFile);
        }
    }
    /* 通常一个同步分组只有一个视频帧，为便于扩展仍以链表方式管理 */
    pNode = pNode->pNext;
}

/* 9. 复合数据使用完毕，将缓存送回音视频复合模块 */
s32Ret = HI_MPI_AVENC_ReleaseStream(AVEncChn, &struStream);
if(HI_SUCCESS != s32Ret)
{
    return HI_FAILURE;
}
```

【相关主题】

无。

HI_MPI_AVENC_ReleaseStream

【描述】

释放指定的音视频复合编码通道的编码码流缓存。

【语法】

```
HI_S32 HI_MPI_AVENC_ReleaseStream(AVENC_CHN AVChnId, AVENC_STREAM_S
*pStream);
```

【参数】

参数名称	描述	输入/输出
AVChnId	音视频复合编码通道号。 取值范围：[0, AVENC_CHN_MAXNUM)。	输入
pStream	已获取的音视频复合编码码流。	输入

【返回值】

返回值	描述
0	成功。
非 0	失败，其值为错误码。

【错误码】

接口返回值	含义
HI_SUCCESS	成功。
HI_ERR_AVENC_INVALID_CHN	复合编码通道无效。
HI_ERR_AVENC_INVALID_PTR	输入指针无效。
HI_ERR_AVENC_CHN_NOT_CREATE	复合编码通道未创建。

【需求】

- 头文件：mpi_avenc.h、hi_comm_avenc.h
- 库文件：libmpi.a

【注意】

用户获取复合编码码流后，必须及时释放已经获取的码流缓存。否则可能会导致码流 buffer 满影响编码器编码，并且必须按先获取先释放的顺序释放已经获取的码流缓存。

【举例】

请参见 HI_MPI_AVENC_ReleaseStream 的举例。

【相关主题】

无。

3 数据类型

3.1 概述

Hi3507 媒体处理软件开发包的数据类型分为基本数据类型、系统控制数据类型、视频功能类数据类型、音频功能类数据类型、音视频复合类数据类型。

3.2 基本数据类型

基本数据类型定义如下：

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

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

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

typedef char                      HI_CHAR;
typedef char*                     HI_PCHAR;

typedef float                     HI_FLOAT;
typedef double                    HI_DOUBLE;
```

```
typedef void                        HI_VOID;

typedef unsigned long               HI_SIZE_T;
typedef unsigned long               HI_LENGTH_T;

typedef enum {
    HI_FALSE    = 0,
    HI_TRUE     = 1,
} HI_BOOL;

#ifndef NULL
#define NULL                        0L
#endif
#define HI_NULL                     0L
#define HI_NULL_PTR                 0L

#define HI_SUCCESS                  0
#define HI_FAILURE                  (-1)

typedef HI_S32 AI_CHN;
typedef HI_S32 AO_CHN;
typedef HI_S32 AENC_CHN;
typedef HI_S32 ADEC_CHN;
typedef HI_S32 AUDIO_DEV;

typedef HI_S32 VI_DEV;
typedef HI_S32 VI_CHN;
typedef HI_S32 VO_CHN;
typedef HI_S32 VENC_CHN;
typedef HI_S32 VDEC_CHN;
typedef HI_S32 VENC_GRP;

/* 无效的通道号、无效的设备号 */
#define HI_INVALID_CHN              (-1)
#define HI_INVALID_DEV              (-1)

/* 最大的视频缓存池个数 */
#define VB_MAX_POOLS                128

/* 所有视频输入设备或通道的最大个数（包括VIU硬件通道和虚拟VI通道）*/
#define VI_MAX_CHN_NUM              8
#define VI_MAX_DEV_NUM              8
```

```
/* 硬件视频输入单元设备最大个数(Hi3507只支持一个视频输入设备)*/
#define VIU_MAX_DEV_NUM             1

/* 每个硬件视频输入设备支持的最大通道数目*/
#define VIU_MAX_CHN_NUM_PER_DEV     2

/* 硬件视频输入通道总的最大个数*/
#define VIU_MAX_CHN_NUM             8

/* VO通道的最大个数 */
#define VO_MAX_CHN_NUM              32
/* 最大的编码组个数 */
#define VENC_MAX_GRP_NUM            32

/* 最大的编码通道个数 */
#define VENC_MAX_CHN_NUM            32

/* 最大的解码通道个数 */
#define VDEC_MAX_CHN_NUM            32

/* SIO的个数，即音频设备的个数 */
#define SIO_MAX_NUM                 2

/* 最大的AIO通道数 */
#define AIO_MAX_CHN_NUM             8

/* 最大的音频编码通道数 */
#define AENC_MAX_CHN_NUM            32

/* 最大的音频解码通道数 */
#define ADEC_MAX_CHN_NUM            32

/* 调用MPI的阻塞标志 */
#define HI_IO_BLOCK                 0

/* 调用MPI的非阻塞标志 */
#define HI_IO_NOBLOCK               1

/* 最大的音视频复合流通道数 */
#define AVENC_CHN_MAXNUM            16

/*  最大音频帧缓存数 */
```

```
#define MAX_AUDIO_FRAME_NUM        50
```

除了上述基本数据类型外，其他基本数据类型定义如下：

- POINT_S：定义坐标信息结构体。
- DIMENSION_S：定义尺寸信息结构体。
- RECT_S：定义矩形区域信息结构体。
- PAYLOAD_TYPE_E：定义音视频净荷类型枚举。

POINT_S

【说明】

定义坐标信息结构体。

【定义】

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

【成员】

成员名称	描述
s32X	横坐标。
s32Y	纵坐标。

【注意事项】

无。

【相关数据类型及接口】

REGION_CTRL_PARAM_U

DIMENSION_S

【说明】

定义尺寸信息结构体。

【定义】

```
typedef struct hiDIMENSION_S
{
    HI_S32  s32Width;
    HI_S32  s32Height;
}DIMENSION_S;
```

【成员】

成员名称	描述
s32Width	宽度。
s32Height	高度。

【注意事项】

无。

【相关数据类型及接口】

REGION_CTRL_PARAM_U

RECT_S

【说明】

定义矩形区域信息结构体。

【定义】

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

【成员】

成员名称	描述
s32X	横坐标。
s32Y	纵坐标。
u32Width	宽度。
u32Height	高度。

【注意事项】

无。

【相关数据类型及接口】

- REGION_ATTR_S
- OVERLAY_ATTR_S

- VI_CHN_ATTR_S
- VI_PUB_ATTR_S
- VO_CHN_ATTR_S

PAYLOAD_TYPE_E

【说明】

定义音视频净荷类型枚举。

【定义】

```
typedef enum
{
    PT_PCMU = 0,
    PT_1016 = 1,
    PT_G721 = 2,
    PT_GSM = 3,
    PT_G723 = 4,
    PT_DVI4_8K = 5,
    PT_DVI4_16K = 6,
    PT_LPC = 7,
    PT_PCMA = 8,
    PT_G722 = 9,
    PT_S16BE_STEREO,
    PT_S16BE_MONO = 11,
    PT_QCELP = 12,
    PT_CN = 13,
    PT_MPEGAUDIO = 14,
    PT_G728 = 15,
    PT_DVI4_3 = 16,
    PT_DVI4_4 = 17,
    PT_G729 = 18,
    PT_G711A = 19,
    PT_G711U = 20,
    PT_G726 = 21,
    PT_G729A = 22,
    PT_LPCM = 23,
    PT_CelB = 25,
    PT_JPEG = 26,
    PT_CUSM = 27,
    PT_NV = 28,
    PT_PICW = 29,
    PT_CPV = 30,
    PT_H261 = 31,
    PT_MPEGVIDEO = 32,
```

```
    PT_MPEG2TS = 33,
    PT_H263 = 34,
    PT_SPEG = 35,
    PT_MPEG2VIDEO = 36,
    PT_AAC = 37,
    PT_WMA9STD = 38,
    PT_HEAAC = 39,
    PT_PCM_VOICE = 40,
    PT_PCM_AUDIO = 41,
    PT_AACLC = 42,
    PT_MP3 = 43,
    PT_ADPCMA = 49,
    PT_AEC = 50,
    PT_X_LD = 95,
    PT_H264 = 96,
    PT_D_GSM_HR = 200,
    PT_D_GSM_EFR = 201,
    PT_D_L8 = 202,
    PT_D_RED = 203,
    PT_D_VDVI = 204,
    PT_D_BT656 = 220,
    PT_D_H263_1998 = 221,
    PT_D_MP1S = 222,
    PT_D_MP2P = 223,
    PT_D_BMPEG = 224,
    PT_MP4VIDEO = 230,
    PT_MP4AUDIO = 237,
    PT_VC1 = 238,
    PT_JVC_ASF = 255,
    PT_D_AVI = 256,
    PT_MAX = 257,

    PT_AMR = 1001, /* add by mpp */
    PT_MJPEG = 1002,
}PAYLOAD_TYPE_E;
```

【成员】

略。

【注意事项】

无。

【相关数据类型及接口】

- VENC_CHN_ATTR_S

- VDEC_CHN_ATTR_S
- AENC_CHN_ATTR_S
- ADEC_CHN_ATTR_S

3.3 系统控制数据类型

系统控制相关数据类型定义如下：

- MPP_SYS_CONF_S：定义 MPP 系统控制属性结构体。
- VB_CONF_S：定义视频缓存池属性结构体。
- MPP_VERSION_S：定义 MPP 版本描述结构体。

MPP_SYS_CONF_S

【说明】

定义 MPP 系统控制属性结构体。

【定义】

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

#define PMC_VI_0_BT601      0x01    /* bit 0~1: VI_0 conect with BT601 */
#define PMC_VI_2_BT601      0x04    /* bit 2~3: VI_1 conect with BT601 */
#define PMC_EN_VI_13        0x10    /* bit 4  : VI_1&VI_3 (or ETH) */

#define PMC_EN_SIO_0        0x0100  /* bit 9  : SIO_0 (or GPIO) */
#define PMC_SIO_0_CLK_R     0x0200  /* bit 10 : pSIO0XCK (or pSIO0RCK) */
    HI_U32  u32PinMuxCtrl;
}MPP_SYS_CONF_S;
```

【成员】

成员名称	描述
u32AlignWidth	整个系统中使用图像的 stride 字节对齐数。 取值范围：[1, 1024]，直接配置成 16 或者 64 即可。 静态属性。

成员名称	描述
u32PinMuxCtrl	管脚复用配置，暂时不需要配置。 取值范围：参考其中的宏定义。 静态属性。

【注意事项】

无。

【相关数据类型及接口】

HI_MPI_SYS_SetConf

VB_CONF_S

【说明】

定义视频缓存池属性结构体。

【定义】

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

【成员】

成员名称	描述
u32MaxPoolCnt	整个系统中可容纳的缓存池个数。 取值范围：(0, VB_MAX_POOLS]。建议配置成产品中视频编码和解码通道个数和的 2 倍。 静态属性。
astCommPool	公共缓存池属性结构体，成员包括公共缓存池中每个缓存块的大小（以 byte 为单位）和缓存块的个数。具体配置依赖于产品形态，具体的配置请参见《Hi3507 媒体处理软件　开发指南》中的描述。 静态属性。

【注意事项】

无。

【相关数据类型及接口】

HI_MPI_SYS_SetConf

MPP_VERSION_S

【说明】

定义 MPI 版本描述结构体。

【定义】

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

【成员】

成员名称	描述
aVersion	版本描述字符串。 比如“HI_VERSION=Hi3511_MPP_V1.0.4.0”。

【注意事项】

无。

【相关数据类型及接口】

HI_MPI_SYS_GetVersion

3.4 视频类数据类型

3.4.1 视频公共类型

视频公共数据类型定义如下：

- VIDEO_NORM_E：定义视频输入制式类型。
- PIXEL_FORMAT_E：定义像素格式类型。
- VIDEO_FRAME_S：定义视频原始图像帧结构体。
- VIDEO_FRAME_INFO_S：定义视频图像帧信息结构体。
- BITMAP_S：定义位图图像信息结构。
- VIDEO_CONTROL_MODE_E：定义视频输入、输出的主从模式。

VIDEO_NORM_E

【说明】

定义视频输入制式类型。

【定义】

```
typedef enum hiVIDEO_NORM_E
{
    VIDEO_ENCODING_MODE_PAL=0,
    VIDEO_ENCODING_MODE_NTSC,
    VIDEO_ENCODING_MODE_AUTO,
    VIDEO_ENCODING_MODE_BUTT,
} VIDEO_NORM_E;
```

【成员】

成员名称	描述
VIDEO_ENCODING_MODE_PAL	PAL 制式。
VIDEO_ENCODING_MODE_NTSC	NTSC 制式。
VIDEO_ENCODING_MODE_AUTO	自动识别制式。

【注意事项】

自动识别制式目前暂不支持。

【相关数据类型及接口】

- VI_PUB_ATTR_S
- TV_CFG_S

PIXEL_FORMAT_E

【说明】

定义像素格式类型。

【定义】

```
typedef enum hiPIXEL_FORMAT_E
{
    PIXEL_FORMAT_RGB_1BPP = 0,
    PIXEL_FORMAT_RGB_2BPP,
    PIXEL_FORMAT_RGB_4BPP,
    PIXEL_FORMAT_RGB_8BPP,
    PIXEL_FORMAT_RGB_444,
    PIXEL_FORMAT_RGB_4444,
```

```
    PIXEL_FORMAT_RGB_555,
    PIXEL_FORMAT_RGB_565,
    PIXEL_FORMAT_RGB_1555,

    PIXEL_FORMAT_RGB_888,
    PIXEL_FORMAT_RGB_8888,
    PIXEL_FORMAT_RGB_PLANAR_888,
    PIXEL_FORMAT_RGB_BAYER,

    PIXEL_FORMAT_YUV_A422,
    PIXEL_FORMAT_YUV_A444,

    PIXEL_FORMAT_YUV_PLANAR_422,
    PIXEL_FORMAT_YUV_PLANAR_420,
    PIXEL_FORMAT_YUV_PLANAR_444,
    PIXEL_FORMAT_YUV_SEMIPLANAR_422,
    PIXEL_FORMAT_YUV_SEMIPLANAR_420,
    PIXEL_FORMAT_YUV_SEMIPLANAR_444,

    PIXEL_FORMAT_UYVY_PACKAGE_422,
    PIXEL_FORMAT_YCbCr_PLANAR,

    PIXEL_FORMAT_BUTT
} PIXEL_FORMAT_E;
```

【成员】

无。

【注意事项】

无。

【相关数据类型及接口】

- VI_CHN_ATTR_S
- OVERLAY_ATTR_S

VIDEO_FRAME_S

【说明】

定义视频原始图像帧结构。

【定义】

```
typedef struct hiVIDEO_FRAME_S
{
    PIXEL_FORMAT_E  enPixelFormat;
    HI_U32          u32Width;
```

```
    HI_U32          u32Height;

#define VIDEO_FIELD_TOP         0x01    /* even field */
#define VIDEO_FIELD_BOTTOM      0x02    /* odd field */
#define VIDEO_FIELD_INTERLACED 0x03    /* two interlaced fields */
#define VIDEO_FIELD_FRAME       0x04    /* frame */

    HI_U32  u32Field;

    HI_U32  u32PhyAddr[3];
    HI_VOID *pVirAddr[3];
    HI_U32  u32Stride[3];

    HI_U16  u16OffsetTop;       /* top offset of show area */
    HI_U16  u16OffsetBottom;    /* bottom offset of show area */
    HI_U16  u16OffsetLeft;      /* left offset of show area */
    HI_U16  u16OffsetRight;     /* right offset of show area */

    HI_U64  u64pts;
    HI_U32  u32TimeRef;

    HI_U32  u32PrivateData;
}VIDEO_FRAME_S;
```

【成员】

成员名称	描述
enPixelFormat	视频图像像素格式。
u32Width	图像宽度。
u32Height	图像高度。
u32Field	帧场模式。
u32PhyAddr	物理地址。
pVirAddr	虚拟地址。
u32Stride	图像跨距。
u16OffsetTop	图像顶部剪裁宽度。
u16OffsetBottom	图像底部剪裁宽度。
u16OffsetLeft	图像左侧剪裁宽度。
u16OffsetRight	图像右侧剪裁宽度。

成员名称	描述
u64pts	图像时间戳。
u32TimeRef	图像帧序列号。
u32PrivateData	私有数据。

【注意事项】

无。

【相关数据类型及接口】

VI_PUB_ATTR_S

VIDEO_FRAME_INFO_S

【说明】

定义视频图像帧信息结构体。

【定义】

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

【成员】

成员名称	描述
stVFrame	视频图像帧。
u32PoolId	视频缓存池 ID。

【注意事项】

无。

【相关数据类型及接口】

VIDEO_FRAME_S

BITMAP_S

【说明】

定义位图图像信息结构。

【定义】

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

【成员】

成员名称	描述
enPixelFormat	位图像素格式。
u32Width	位图宽度。
u32Height	位图高度。
pData	位图数据。

【注意事项】

无。

【相关数据类型及接口】

REGION_CTRL_PARAM_U

VIDEO_CONTROL_MODE_E

【说明】

定义视频输入、输出主从模式。

【定义】

```
typedef enum hiVIDEO_CONTROL_MODE_E
{
    VIDEO_CONTROL_MODE_SLAVER=0,
    VIDEO_CONTROL_MODE_MASTER,
    VIDEO_CONTROL_MODE_BUTT
}VIDEO_CONTROL_MODE_E;
```

【成员】

成员名称	描述
VIDEO_CONTROL_MODE_SLAVER	从模式。
VIDEO_CONTROL_MODE_MASTER	主模式。
VIDEO_CONTROL_MODE_BUTT	无意义。

【注意事项】

Hi3507 芯片的主从模式一定要与外围 AD/DA 相适配。

【相关数据类型及接口】

略。

3.4.2 视频输入

视频输入相关数据类型定义如下：

- VI_ADTYPE_E：定义 AD 芯片类型。
- VI_CAPSEL_E：定义视频输入捕获图像的帧场选择。
- VI_INPUT_MODE_E：定义视频输入接口模式。
- VI_WORK_MODE_E：定义视频输入工作模式。
- VI_PIXEL_FORMAT_E：定义视频输入图像的像素格式。
- VI_CHN_ATTR_S：定义视频输入通道属性结构体。
- VI_PUB_ATTR_S：定义视频输入设备的公共属性结构体。
- VI_CH_LUM_S：定义视频输入通道的图像亮度信息结构体。

VI_ADTYPE_E

【说明】

定义 AD 芯片类型。

【定义】

```
typedef enum hiVI_ADTYPE_E
{
    AD_2815=0,
    AD_OTHERS
} VI_ADTYPE_E;
```

【成员】

成员名称	描述
AD_2815	TW2815 AD 芯片。
AD_OTHERS	其他 AD 芯片。

【注意事项】

无。

【相关数据类型及接口】

VI_PUB_ATTR_S

VI_CAPSEL_E

【说明】

定义视频输入捕获图像的帧场选择。

【定义】

```
typedef enum hiVI_CAPSEL_E
{
    VI_CAPSEL_TOP=0,
    VI_CAPSEL_BOTTOM,
    VI_CAPSEL_BOTH,
    VI_CAPSEL_BUTT
} VI_CAPSEL_E;
```

【成员】

成员名称	描述
VI_CAPSEL_TOP	选择顶场。
VI_CAPSEL_BOTTOM	选择底场（推荐）。
VI_CAPSEL_BOTH	选择顶底两场。

【注意事项】

无。

【相关数据类型及接口】

VI_CHN_ATTR_S

VI_INPUT_MODE_E

【说明】

定义视频输入接口模式。

【定义】

```
typedef enum hiVI_INPUT_MODE_E
{
    VI_MODE_BT656=0,
    VI_MODE_BT601,
    VI_MODE_DIGITAL_CAMERA,
    VI_MODE_720P,
    VI_MODE_BUTT
} VI_INPUT_MODE_E;
```

【成员】

成员名称	描述
VI_MODE_BT656	ITUR BT.656 模式。
VI_MODE_BT601	ITUR BT.601 模式。
VI_MODE_DIGITAL_CAMERA	数字输入模式。
VI_MODE_720P	数字高清模式。

【注意事项】

无。

【相关数据类型及接口】

VI_PUB_ATTR_S

VI_WORK_MODE_E

【说明】

定义视频输入工作模式。

【定义】

```
typedef enum hiVI_WORK_MODE_E
{
    VI_WORK_MODE_1D1=0,
    VI_WORK_MODE_2D1,
    VI_WORK_MODE_4HALFD1,
    VI_WORK_MODE_BUTT
} VI_WORK_MODE_E;
```

【成员】

成员名称	描述
VI_WORK_MODE_1D1	1D1 工作模式。
VI_WORK_MODE_2D1	2D1 工作模式。
VI_WORK_MODE_4HALFD1	4Half D1 工作模式。

【注意事项】

无。

【相关数据类型及接口】

VI_PUB_ATTR_S

VI_PIXEL_FORMAT_E

【说明】

定义视频输入图像的像素格式。

【定义】

```
typedef enum hiVI_PIXEL_FORMAT_E
{
    VI_PIXFORMAT_422,
    VI_PIXFORMAT_420,
    VI_PIXFORMAT_BUTT,
} VI_PIXEL_FORMAT_E;
```

【成员】

成员名称	描述
VI_PIXFORMAT_422	YUV4:2:2 格式。
VI_PIXFORMAT_420	YUV4:2:0 格式。

【注意事项】

无。

【相关数据类型及接口】

VI_CHN_ATTR_S

VI_CHN_ATTR_S

【说明】

定义视频输入通道属性结构体。

【定义】

```
typedef struct hiVI_CHN_ATTR_S
{
    RECT_S          stCapRect;
    VI_CAPSEL_E     enCapSel;
    HI_BOOL         bDownScale;
    HI_BOOL         bChromaResample;
    HI_BOOL         bHighPri;
    PIXEL_FORMAT_E  enViPixFormat;
}VI_CHN_ATTR_S;
```

【成员】

成员名称	描述
stCapRect	通道捕获区域属性。 动态属性。
enCapSel	帧场选择。 动态属性。
bDownScale	1/2 水平压缩选择。 动态属性。
bChromaResample	色度重采样选择。 动态属性。
bHighPri	高优先级选择。 动态属性。
enViPixFormat	像素格式。 动态属性。

【注意事项】

无。

【相关数据类型及接口】

- RECT_S
- VI_CAPSEL_E
- VI_PIXEL_FORMAT_E
- HI_MPI_VI_SetChnAttr

VI_PUB_ATTR_S

【说明】

定义视频输入设备的公共属性结构体。

【定义】

```
typedef struct hiVI_PUB_ATTR_S
{
    VI_INPUT_MODE_E enInputMode;
    VI_WORK_MODE_E  enWorkMode;
    VI_ADTYPE_E     u32AdType;
    VIDEO_NORM_E    enViNorm;
    RECT_S          stRect;
}VI_PUB_ATTR_S;
```

【成员】

成员名称	描述
enInputMode	视频输入接口模式。 静态属性。
enWorkMode	视频输入工作模式。 静态属性。
u32AdType	AD 芯片类型。 静态属性。
enViNorm	接口制式。 静态属性。
stRect	整体区域设置（TW2815 下无效）。 静态属性。

【注意事项】

无。

【相关数据类型及接口】

- VI_ADTYPE_E
- VI_INPUT_MODE_E
- VI_WORK_MODE_E
- HI_MPI_VI_SetPubAttr

VI_CH_LUM_S

【说明】

定义视频输入通道的图像亮度信息结构体。

【定义】

```
typedef struct hiVI_CH_LUM_S
{
    HI_U32  u32FramId;
    HI_U32  u32Lum;
}VI_CH_LUM_S;
```

【成员】

成员名称	描述
u32FramId	原始图像帧序号。

成员名称	描述
u32Lum	亮度值。

【注意事项】

无。

【相关数据类型及接口】

HI_MPI_VI_GetChnLuma

3.4.3 视频输出

视频输出相关数据类型定义如下：

- TV_CFG_S：定义 TV 配置属性结构体。
- VO_PUB_ATTR_S：定义视频输出公共属性结构体。
- VO_CHN_ATTR_S：定义视频输出通道属性结构体。
- VO_DISPLAY_FIELD_E：定义视频输出时的顶底场模式。
- VO_SYNC_GROUP_ATTR_SRECT_S：定义视频输出多通道同步组属性。
- RECT_S：定义局部放大窗口矩形结构体。
- VO_SYNC_MODE_E：定义视频输出多通道同步模式。
- VO_QUERY_STATUS_S：定义视频输出通道状态结构体。
- VO_SYNC_BASE_S：定义视频输出同步基准结构体。

TV_CFG_S

【说明】

定义 TV 配置属性结构体。

【定义】

```
typedef struct hiTV_CFG_S
{
    VIDEO_NORM_E    stComposeMode;
} TV_CFG_S;
```

【成员】

成员名称	描述
stComposeMode	输出信号合成制式模式。

【注意事项】

无。

【相关数据类型及接口】

VO_PUB_ATTR_S

VO_PUB_ATTR_S

【说明】

定义视频输出公共属性结构体。

【定义】

```
typedef struct hiVO_PUB_ATTR_S
{
    TV_CFG_S    stTvConfig;
    HI_U32      u32BgColor;
}VO_PUB_ATTR_S;
```

【成员】

成员名称	描述
stTvConfig	TV 模式配置结构。 静态属性。
u32BgColor	背景色 RGB（8:8:8）低 24bit 有效。 静态属性。

【注意事项】

无。

【相关数据类型及接口】

- TV_CFG_S
- HI_MPI_VO_SetPubAttr

VO_CHN_ATTR_S

【说明】

定义视频输出通道属性。

【定义】

```
typedef struct hiVO_CHN_ATTR_S
{
    HI_U32  u32Priority;
    RECT_S  stRect;
    HI_BOOL bZoomEnable;
}VO_PUB_ATTR_S;
```

【成员】

成员名称	描述
u32Priority	视频通道叠加优先级，优先级高的在上层。 取值范围：[0, 15]。 动态属性。
stRect	通道矩形显示区域。以屏幕的左上角为原点。该矩形的左上角座标必须是 8 对齐，且该矩形区域必须在屏幕范围之内。 取值范围： • s32X：[0, 720]。 • s32Y：[0, 576]。 • u32Width：(0, 720]。 • u32Height：(0, 576]。 动态属性。
bZoomEnable	缩放开关标识。 取值范围： • HI_TRUE：将输入图像缩放成 stRect 定义的尺寸在屏幕上显示。 • HI_FALSE：输入图像上剪裁 stRect 定义的矩形区域进行显示。 动态属性。

【注意事项】

无。

【相关数据类型及接口】

- RECT_S
- HI_MPI_VO_SetChnAttr

VO_DISPLAY_FIELD_E

【说明】

定义视频输出时的顶底场模式。

【定义】

```
typedef enum hiVO_DISPLAY_FIELD_E
{
  VO_FIELD_TOP,     /* top field*/
  VO_FIELD_BOTTOM,  /* bottom field*/
  VO_FIELD_BOTH,    /* top and bottom field*/
  VO_FIELD_BUTT
} VO_DISPLAY_FIELD_E;
```

【成员】

成员名称	描述
VO_FIELD_TOP	视频输出缩放时只处理顶场。
VO_FIELD_BOTTOM	视频输出缩放时只处理底场。
VO_FIELD_BOTH	视频输出缩放时两场都处理。

【注意事项】

无。

【相关数据类型及接口】

HI_MPI_VO_SetChnField

VO_SYNC_GROUP_ATTR_S

【说明】

视频输出多通道同步组属性。

【定义】

```
typedef struct hiVO_SYNC_GROUP_ATTR_S
{
    HI_U32 u32SynFrameRate;     /* frame rate of this group */
    VO_SYNC_MODE_E enSyncMode;  /* synchronized mode */
    HI_S32 u32UsrBaseChn;       /* reference channel of sync */
} VO_SYNC_GROUP_ATTR_S;
```

【成员】

成员名称	描述
u32SynFrameRate	视频输出同步组帧率。
enSyncMode	视频输出同步组同步模式。
u32UsrBaseChn	视频输出同步组同步时间戳基准通道。

【注意事项】

无。

【相关数据类型及接口】

HI_MPI_VO_CreateSyncGroup

RECT_S

【说明】

定义局部放大窗口矩形结构体。

【定义】

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

【成员】

成员名称	描述
s32X	局部放大窗起始 X 坐标，要求非负，并且 8 像素对齐。
s32Y	局部放大窗起始 Y 坐标，要求非负，并且 8 像素对齐。
u32Width	局部放大窗宽度，要求不大于输入图像宽度。
u32Height	局部放大窗高度，要求不大于输入图形高度。

【注意事项】

该结构体在 hi_common.h 中定义，但是 VO 中对它的使用作了一定的限制，详见成员说明。

【相关数据类型及接口】

无。

VO_SYNC_MODE_E

【说明】

视频输出多通道同步模式。

【定义】

```
typedef enum hiVO_SYNC_MODE_E
{
    VO_SYNC_MODE_MIN_CHN,   /* use the channel which has minimal pts */
    VO_SYNC_MODE_MAX_CHN,   /* use the channel which has maximal pts */
    VO_SYNC_MODE_USR_CHN,   /* use the channel which user appointed */
    VO_SYNC_MODE_BUTT
} VO_SYNC_MODE_E;
```

【成员】

成员名称	描述
VO_SYNC_MODE_MIN_CHN	最小时间戳模式。
VO_SYNC_MODE_MAX_CHN	最大时间戳模式。
VO_SYNC_MODE_USR_CHN	用户指定基准时间戳通道模式。

【注意事项】

无。

【相关数据类型及接口】

HI_MPI_VO_CreateSyncGroup

VO_QUERY_STATUS_S

【说明】

视频输出通道状态结构体。

【定义】

```
typedef struct hiVO_QUERY_STATUS_S
{
    HI_U32 u32ChnBufUsed;      /* channel buffer that been occupied */
} VO_QUERY_STATUS_S;
```

【成员】

成员名称	描述
u32ChnBufUsed	视频输出通道当前占用的视频 buffer 数目。

【注意事项】

无。

【相关数据类型及接口】

HI_MPI_VO_Query

VO_SYNC_BASE_S

【说明】

定义视频输出同步基准结构体。

【定义】

```
typedef struct hiVO_SYNC_BASE_S
{
    VO_CHN VoChn;             /* base channel to synchronize */
    HI_U64 u64BasePts;         /* base pts to synchronize */
} VO_SYNC_BASE_S;
```

【成员】

成员名称	描述
VoChn	基准通道。
u64BasePts	基准时间戳。

【注意事项】

无。

【相关数据类型及接口】

HI_MPI_VO_SetSyncGroupBase

3.4.4 视频前处理

视频前处理相关数据类型、数据结构定义如下：

- MAX_COVER_NUM：定义同一个 VI 通道的最大遮挡区域的个数。
- MAX_OVERLAY_NUM：定义同一个 GROUP 通道组的最大叠加区域的个数。
- MAX_SOVERLAY_NUM：定义同一个 VI 通道的最大软叠加区域的个数。
- MAX_SOVERLAY_AREA：定义同一个 VI 通道的所有软叠加区域的面积最大值。
- MAX_SOVERLAY_ALPHA：定义软叠加区域 alpha 值的最大值。
- VI_COVER_REGION：定义所有的遮挡区域的个数最大值。
- VENC_OVERLAY_REGION：定义所有的叠加区域的个数最大值。
- VI_SOVERLAY_REGION：定义所有的软叠加区域的个数最大值。
- VPP_SCALE_DENOISE_CHOICE_E：定义缩放去噪选择。
- VPP_SCALE_FILTER_E：定义缩放系数。
- VPP_DENOISE_E：定义去噪系数。
- VPP_CE_E：定义色彩增强开关。
- VPP_LUMA_STR_E：定义对比度拉伸开关。
- VPP_SCALE_MODE_E：定义缩放方式。
- VIDEO_PREPROC_CONF_S：定义视频前处理配置的数据结构体。
- VPP_SCALE_CONF_S：定义视频缩放任务配置的数据结构体。
- REGION_HANDLE：定义区域句柄。
- REGION_TYPE_E：定义区域类型。

- COVER_ATTR_S：定义遮挡区域属性结构体。
- OVERLAY_ATTR_S：定义叠加区域属性结构体。
- SOFT_OVERLAY_ATTR_S：定义软叠加区域属性结构体。
- MOSAIC_ATTR_S：定义马赛克区域属性结构体。
- REGION_ATTR_U：定义区域属性联合体。
- REGION_ATTR_S：定义区域类型结构体。
- REGION_CRTL_CODE_E：定义区域操作命令。
- COVER_S：定义所有遮挡区域结构体。
- OVERLAY_S：定义所有叠加区域结构体。
- SOFT_OVERLAY_S：定义所有软叠加区域结构体。
- MOSAIC_S：定义所有马赛克区域结构体。
- REGION_CTRL_PARAM_U：定义控制区域参数。
- PIC_SCALE_TASK_S：定义缩放任务结构体。

MAX_COVER_NUM

【说明】

定义同一个 VI 通道的最大遮挡区域的个数。

【定义】

```
#define MAX_COVER_NUM      4;
```

【成员】

无。

【注意事项】

无。

【相关数据类型及接口】

无。

MAX_OVERLAY_NUM

【说明】

定义同一个 GROUP 通道组的最大叠加区域的个数。

【定义】

```
#define MAX_OVERLAY_NUM    4;
```

【成员】

无。

【注意事项】

无。

【相关数据类型及接口】

无。

MAX_SOVERLAY_NUM

【说明】

定义同一个 VI 通道的最大软叠加区域的个数。

【定义】

```
#define MAX_SOVERLAY_NUM       4;
```

【成员】

无。

【注意事项】

无。

【相关数据类型及接口】

无。

MAX_SOVERLAY_AREA

【说明】

定义同一个 VI 通道的所有软叠加区域的面积最大值。

【定义】

```
#define MAX_SOVERLAY_AREA  414720
```

【成员】

无。

【注意事项】

无。

【相关数据类型及接口】

无。

MAX_SOVERLAY_ALPHA

【说明】

定义软叠加区域 alpha 值的最大值。

【定义】

```
#define MAX_SOVERLAY_ALPHA 128
```

【成员】

无。

【注意事项】

无。

【相关数据类型及接口】

无。

VI_COVER_REGION

【说明】

定义所有的遮挡区域的个数最大值。

【定义】

```
#define VI_COVER_REGION (VIU_MAX_DEV_NUM * VIU_MAX_CHN_NUM_PER_DEV
*MAX_COVER_NUM);
```

【成员】

无。

【注意事项】

无。

【相关数据类型及接口】

MAX_COVER_NUM

VENC_OVERLAY_REGION

【说明】

定义所有的叠加区域的个数最大值。

【定义】

```
#define VENC_OVERLAY_REGION (VENC_MAX_CHN_NUM*MAX_OVERLAY_NUM);
```

【成员】

无。

【注意事项】

无。

【相关数据类型及接口】

MAX_OVERLAY_NUM

VI_SOVERLAY_REGION

【说明】

定义所有的软叠加区域的个数最大值。

【定义】

```
#define VI_SOVERLAY_REGION (VIU_MAX_DEV_NUM * VIU_MAX_CHN_NUM_PER_DEV
*MAX_SOVERLAY_NUM)
```

【成员】

无。

【注意事项】

无。

【相关数据类型及接口】

MAX_SOVERLAY_NUM

VPP_SCALE_DENOISE_CHOICE_E

【说明】

定义缩放去噪选择。

【定义】

```
typedef enum hiVPP_SCALE_DENOISE_CHOICE_E
{
    VPP_SCALE,
    VPP_DENOISE,
    VPP_BUTT,
}VPP_SCALE_DENOISE_CHOICE_E;
```

【成员】

成员名称	描述
VPP_SCALE	缩放。
VPP_DENOISE	空域去噪。

【注意事项】

无。

【相关数据类型及接口】

VPP_SCALE_CONF_S

VPP_SCALE_FILTER_E

【说明】

缩放系数。

【定义】

```
typedef enum hiVPP_SCALE_FILTER_E
{
    VPP_SCALE_FILTER_DEFAULT = 0,
    VPP_SCALE_FILTER_1M,
    VPP_SCALE_FILTER_2M,
    VPP_SCALE_FILTER_3M,
    VPP_SCALE_FILTER_4M,
    VPP_SCALE_FILTER_5M,
    VPP_SCALE_FILTER_6M,
    VPP_SCALE_FILTER_BUTT,
}VPP_SCALE_FILTER_E;
```

【成员】

成员名称	描述
VPP_SCALE_FILTER_DEFAULT	默认缩放系数。
VPP_SCALE_FILTER_n M （n = 1，……，6）	缩放系数（1M～6M）。数值越小，滤波后的图像越模糊。

【注意事项】

无。

【相关数据类型及接口】

- VIDEO_PREPROC_CONF_S
- VPP_SCALE_CONF_S

VPP_DENOISE_E

【说明】

定义去噪系数。

【定义】

```
typedef enum hiVPP_DENOISE_E
{
    VPP_DENOISE_ONLYEDAGE = 0,
    VPP_DENOISE_LOWNOISE,
    VPP_DENOISE_MIDNOISE,
```

```
    VPP_DENOISE_HIGHNOISE,
    VPP_DENOISE_VERYHIGHNOISE,
    VPP_DENOISE_BUTT,
}VPP_DENOISE_E;
```

【成员】

成员名称	描述
VPP_DENOISE_ONLYEDAGE	边缘去噪。
VPP_DENOISE_LOWNOISE	低去噪。
VPP_DENOISE_MIDNOISE	中去噪。
VPP_DENOISE_HIGHNOISE	高去噪。
VPP_DENOISE_VERYHIGHNOISE	极高去噪。

【注意事项】

无。

【相关数据类型及接口】

无。

VPP_CE_E

【说明】

定义色彩增强开关。

【定义】

```
typedef enum hiVPP_CE_E
{
    VPP_CE_DISABLE = 0,
    VPP_CE_ENABLE,
    VPP_CE_BUTT,
}VPP_CE_E;
```

【成员】

成员名称	描述
VPP_CE_DISABLE	色彩增强关。
VPP_CE_ENABLE	色彩增强开。

【注意事项】

无。

【相关数据类型及接口】

VPP_SCALE_CONF_S

VPP_LUMA_STR_E

【说明】

定义对比度拉伸开关。

【定义】

```
typedef enum hiVPP_LUMA_STR_E
{
    VPP_LUMA_STR_DISABLE = 0,
    VPP_LUMA_STR_ENABLE,
    VPP_LUMA_STR_BUTT,
}VPP_LUMA_STR_E;
```

【成员】

成员名称	描述
VPP_LUMA_STR_DISABLE	对比度拉伸关。
VPP_LUMA_STR_ENABLE	对比度拉伸开。

【注意事项】

无。

【相关数据类型及接口】

- VIDEO_PREPROC_CONF_S
- VPP_SCALE_CONF_S

VPP_SCALE_MODE_E

【说明】

定义缩放方式。

【定义】

```
typedef enum hiVPP_SCALE_MODE_E
{
    VPP_SCALE_MODE_USEBOTTOM = 0,   /*use bottom scale*/
    VPP_SCALE_MODE_USETOP,          /*use top scale*/
    VPP_SCALE_MODE_DIRECT,          /*scale directlly*/
    VPP_SCALE_MODE_BUTT,
```

```
}VPP_SCALE_MODE_E;
```

【成员】

成员名称	描述
VPP_SCALE_MODE_USETOP	取顶场缩放。
VPP_SCALE_MODE_USEBOTTOM	取底场缩放。
VPP_SCALE_MODE_DIRECT	直接缩放。

【注意事项】

- 丢场缩放可以有效的提高系统性能，当不满足丢场缩放时，必须进行直接缩放。类似这种问题可以推广到各种不同的场景，比如 VI 输入 D1 或编码 QVGA 之类大小不严格满足 1/2 和 1/4 的情形。
- 缩放方式只对大码流（JPEG 抓拍和 MPEG4）有效，小码流无效。
- 如果输入图像和编码图像的高度比大于 1/2 则自动使用 SCALE_MODE_DIRECT 方式，其他设置无效。

【相关数据类型及接口】

VIDEO_PREPROC_CONF_S

VIDEO_PREPROC_CONF_S

【说明】

定义视频前处理配置的数据结构体。

【定义】

```
typedef struct hiVIDEO_PREPROC_CONF_S
{
    HI_BOOL             bTemporalDenoise;
    HI_BOOL             bColorToGrey;
    HI_S32 s32SrcFrmRate;
    HI_S32 s32TarFrmRate;
    VPP_SCALE_MODE_E    enScaleMode;
    VPP_SCALE_FILTER_E  enFilter;
}VIDEO_PREPROC_CONF_S;
```

【成员】

成员名称	描述
bTemporalDenoise	时域去噪开关。 取值范围： • HI_TRUE：开启时域去噪。 • HI_FALSE：关闭时域去噪。
bColorToGrey	彩转灰开关。 取值范围： • HI_TRUE：打开彩转灰。 • HI_FALSE：关闭彩转灰。
s32SrcFrmRate	Group 的原始帧率，与 VI 采集的帧率相等。
s32TarFrmRate	Group 的目标帧率。
enScaleMode	缩放方式。
enFilter	缩放系数。

【注意事项】

无。

【相关数据类型及接口】

- VPP_SCALE_MODE_E
- VPP_SCALE_FILTER_E
- HI_MPI_VPP_SetConf

VPP_SCALE_CONF_S

【说明】

定义视频缩放任务配置的数据结构体。

【定义】

```
typedef struct hiVPP_SCALE_CONF_S
{
    HI_BOOL                     bTemporalDenoise;
    HI_BOOL                     bDeInterlace;
    HI_BOOL                     bColorToGrey;
    VPP_SCALE_DENOISE_CHOICE_E  enChoice;
    VPP_CE_E                    enCE;
    VPP_LUMA_STR_E              enLumaStr;
    VPP_SCALE_FILTER_E          enFilter;
    VPP_DENOISE_E               enSpatialDenoise;
}VPP_SCALE_CONF_S;
```

【成员】

成员名称	描述
bTemporalDenoise	时域去噪开关。 取值范围： • HI_TRUE：打开时域去噪。 • HI_FALSE：关闭时域去噪。
bDeInterlace	DeInterlace 开关。 取值范围： • HI_TRUE：打开 Deinterlace。 • HI_FALSE：关闭 Deinterlace。
bColorToGrey	彩转灰开关。 取值范围： • HI_TRUE：打开彩转灰。 • HI_FALSE：关闭彩转灰。
enChoice	缩放空域去噪选择。
enCE	色彩增强开关。
enLumaStr	对比度拉伸开关。
enFilter	缩放系数。
enSpatialDenoise	空域去噪系数。

【注意事项】

无。

【相关数据类型及接口】

- VPP_SCALE_DENOISE_CHOICE_E
- VPP_SCALE_FILTER_E
- VPP_DENOISE_E
- VPP_CE_E
- VPP_LUMA_STR_E
- HI_MPI_VPP_CreateScaleTask

说明

现版本不支持缩放任务配置中的时域去噪、Deinterlace、彩转灰、空域去噪、色彩增强、对比度拉伸；仅支持缩放。

REGION_HANDLE

【说明】

定义区域句柄。

【定义】

```
typedef HI_U32 REGION_HANDLE;
```

【成员】

成员名称	描述
REGION_HANDLE	区域句柄。

【注意事项】

无。

【相关数据类型及接口】

- HI_MPI_VPP_CreateRegion
- HI_MPI_VPP_DestroyRegion
- HI_MPI_VPP_ControlRegion

REGION_TYPE_E

【说明】

定义区域类型。

【定义】

```
typedef enum hiREGION_TYPE_E
{
    COVER_REGION = 0,
    OVERLAY_REGION,
    SOFTOVERLAY_REGION,
    REGION_BUTT
} REGION_TYPE_E;
```

【成员】

成员名称	描述
COVER_REGION	遮挡区域。
OVERLAY_REGION	叠加区域。
SOFTOVERLAY_REGION	软叠加区域。

【注意事项】

无。

【相关数据类型及接口】

REGION_ATTR_S

COVER_ATTR_S

【说明】

定义遮挡区域属性结构体。

【定义】

```
typedef struct hiCOVER_ATTR_S
{
    VI_DEV  ViDevId;
    VI_CHN  ViChn;
    HI_BOOL bIsPublic;
    HI_U32  u32Layer;
    RECT_S  stRect;
    HI_U32  u32Color;
} COVER_ATTR_S;
```

【成员】

成员名称	描述
ViDevId	VI 设备号。 取值范围：[0, 3]。 静态属性。
ViChn	VI 通道号。 取值范围：[0, 3]。 静态属性。
bIsPublic	是否公共区域。 取值范围： • HI_TRUE：是公共区域。 • HI_FALSE：不是公共区域。 静态属性。
u32Layer	层次。 取值范围：[0, 100]。 动态属性。
stRect	区域的位置和大小。 取值范围：遮挡区域起始点坐标大于等于 0，高宽大于 0，最大为 4095 % 4095，高宽最大值都为 4095。 动态属性。

成员名称	描述
u32Color	背景色。 取值范围：[0, 0xFFFFFF]。 动态属性。

【注意事项】

无。

【相关数据类型及接口】

REGION_ATTR_U

OVERLAY_ATTR_S

【说明】

定义叠加区域属性结构体。

【定义】

```
typedef struct hiOVERLAY_ATTR_S
{
    VENC_GRP        VeGroup;
    HI_BOOL         bPublic;
    RECT_S          stRect;
    PIXEL_FORMAT_E  enPixelFmt;
    HI_U32          u32FgAlpha;
    HI_U32          u32BgAlpha;
    HI_U32          u32BgColor;
} OVERLAY_ATTR_S;
```

【成员】

成员名称	描述
VeGroup	编码通道组号。 取值范围：[0, 31]。 静态属性。
bPublic	公共区域标识。 取值范围： • HI_TRUE：公共区域。 • HI_FALSE：非公共区域。 静态属性。

成员名称	描述
stRect	区域的位置和高宽。 取值范围：叠加区域的起始点的座标必须大于 0 且为偶数，长宽也都必须为偶数，以像素为单位，最大为 2047 % 2047，高宽最大值都为 2047。而且对于叠加区域，起始点的 X 坐标必须为 4 的倍数。 起始位置可以动态改变。 高宽不可以改变。
enPixelFmt	像素的格式。 取值范围： • PIXEL_FORMAT_RGB_1555。 • PIXEL_FORMAT_RGB_4444。 静态属性。
u32FgAlpha	前景 Alpha 值。 取值范围：[0, 128]。 动态属性。
u32BgAlpha	背景 Alpha 值。 取值范围：[0, 128]。 动态属性。
u32BgColor	背景色。 取值范围：[0, 0x7FFF]。 动态属性。

【注意事项】

无。

【相关数据类型及接口】

REGION_ATTR_U

SOFT_OVERLAY_ATTR_S

【说明】

定义软叠加区域属性结构体。

【定义】

```
typedef struct hiSOFT_OVERLAY_ATTR_S
{
    VI_DEV ViDevId;
    VI_CHN ViChn;
```

```
    HI_BOOL bIsPublic;
    RECT_S stRect;
    PIXEL_FORMAT_E enPixelFmt;
    HI_U32 u32Layer;
    HI_U32 u32BgColor;
    HI_U32 u32FgAlpha;
    HI_U32 u32BgAlpha;
}SOFT_OVERLAY_ATTR_S;
```

【成员】

成员名称	描述
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM]。 静态属性。
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM]。 静态属性。
bPublic	公共区域标识。 取值范围： • HI_TRUE：公共区域。 • HI_FALSE：非公共区域。 静态属性。
stRect	区域的位置和高宽。 取值范围：软叠加区域的起始点的座标必须大于 0 且为偶数，长宽也都必须为偶数，以像素为单位。 起始位置可以动态改变。 高宽不可以改变。
enPixelFmt	像素的格式。 静态属性。
u32Layer	层次。 取值范围：[0, 100]。 动态属性。
u32FgAlpha	前景 Alpha 值。 取值范围：[0, 128]。 动态属性。

成员名称	描述
u32BgAlpha	背景 Alpha 值。 取值范围：[0, 128]。 动态属性。
u32BgColor	背景色。 取值范围：[0, 0x7FFF]。 动态属性。

【注意事项】

无。

【相关数据类型及接口】

REGION_ATTR_U

MOSAIC_ATTR_S

【说明】

定义马赛克区域属性结构体。

【定义】

```
typedef struct hiMOSAIC_ATTR_S
{
    VI_DEV ViDevId; /* VI device ID */
    VI_CHN ViChn;   /* VI channel ID */
    HI_BOOL bIsPublic; /*is a public region? */
    RECT_S stRect; /*position and size of region */
    /*bitmap pixel format,now only support semi_plan 420 or
semi_plan422*/
    PIXEL_FORMAT_E enPixelFmt;
    /*
    *The top level when more than one OSD region is showing.
    *The smaller value will be show more backgroud.
    *bound [0~100]
    */
    HI_U32 u32Layer;
    /*
    ** The transparent color will be treated as invisible
color.COLORSPACECOEF
    ** RGB format should be used for this parament.now only support RGB888!
    */
    HI_U32 u32BgColor;
```

```
}MOSAIC_ATTR_S;
```

【成员】

成员名称	描述
ViDevId	VI 设备号。 取值范围：[0, VIU_MAX_DEV_NUM]。 静态属性。
ViChn	VI 通道号。 取值范围：[0, VIU_MAX_CHN_NUM]。 静态属性。
bPublic	公共区域标识。 取值范围： • HI_TRUE：公共区域。 • HI_FALSE：非公共区域。 静态属性。
stRect	区域的位置和高宽。 取值范围：软叠加区域的起始点的座标必须大于 0 且为偶数，长宽也都必须为偶数，以像素为单位。 起始位置可以动态改变。 高宽不可以改变。
enPixelFmt	像素的格式。（目前只支持 semi_plan 420 or semi_plan422 格式） 静态属性。
u32Layer	层次。 取值范围：[0, 100]。 动态属性。
u32BgColor	背景色。 取值范围：[0, 0x7FFF]。 动态属性。

【注意事项】

无。

【相关数据类型及接口】

REGION_ATTR_U

REGION_ATTR_U

【说明】

定义区域属性联合体。

【定义】

```
typedef union hiREGION_ATTR_U
{
    COVER_ATTR_S    stCover;
    OVERLAY_ATTR_S  stOverlay;
    SOFT_OVERLAY_ATTR_S stSOverlay;
    MOSAIC_ATTR_S stMosaic;
}REGION_ATTR_U;
```

【成员】

成员名称	描述
stCover	遮挡区域属性。
stOverlay	叠加区域属性。
stSOverlay	软叠加区域属性。
stMosaic	马赛克区域属性。

【注意事项】

无。

【相关数据类型及接口】

- COVER_ATTR_S
- OVERLAY_ATTR_S
- VI_SOVERLAY_REGION

REGION_ATTR_S

【说明】

定义区域类型结构体。

【定义】

```
typedef struct hiREGION_ATTR_S
{
    REGION_TYPE_E   enType;
    REGION_ATTR_U   unAttr;
}REGION_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】

- REGION_TYPE_E
- REGION_ATTR_U
- HI_MPI_VPP_CreateRegion

REGION_CRTL_CODE_E

【说明】

定义区域操作命令。

【定义】

```
typedef enum hiREGION_CRTL_CODE_E
{
    REGION_SHOW = 0,
    REGION_HIDE,
    REGION_SET_POSTION,
    REGION_SET_COLOR,
    REGION_SET_LAYER,
    REGION_SET_SIZE,
    REGION_SET_ALPHA0,
    REGION_SET_ALPHA1,
    REGION_SET_BITMAP,
    REGION_GET_SIGNLE_ATTR,
    REGION_GET_ALL_COVER_ATTR,
    REGION_GET_ALL_OVERLAY_ATTR,
    REGION_GET_ALL_SOFT_OVERLAY_ATTR,
}REGION_CRTL_CODE_E;
```

【成员】

成员名称	描述
REGION_SHOW	显示区域。

成员名称	描述
REGION_HIDE	隐藏区域。
REGION_SET_POSTION	改变区域位置。
REGION_SET_COLOR	改变区域背景色。
REGION_SET_LAYER	改变区域层次（只针对遮挡区域）。
REGION_SET_SIZE	改变区域高宽（只针对遮挡区域）。
REGION_SET_ALPHA0	改变区域背景 Alpha 值（只针对叠加区域且像素格式为 ARGB1555）。
REGION_SET_ALPHA1	改变区域前景 Alpha 值（只针对叠加区域且像素格式为 ARGB1555）。
REGION_SET_BITMAP	填充区域位图（只针对叠加区域）。
REGION_GET_SIGNLE_ATTR	获取单个区域属性。
REGION_GET_ALL_COVER_ATTR	获取所有遮挡区域属性。
REGION_GET_ALL_OVERLAY_ATTR	获取所有叠加区域属性。
REGION_GET_ALL_SOFT_OVERLAY_ATTR	获取所有软叠加区域的属性。

【注意事项】

无。

【相关数据类型及接口】

HI_MPI_VPP_ControlRegion

COVER_S

【说明】

定义所有遮挡区域结构体。

【定义】

```
typedef struct hiCOVER_S
{
    HI_U32          u32CoverNum;
    REGION_HANDLE   aCoverHandles[VI_COVER_REGION];
    COVER_ATTR_S    astAttr[VI_COVER_REGION];
```

```
}COVER_S;
```

【成员】

成员名称	描述
u32CoverNum	遮挡区域的个数。
aCoverHandles[VI_COVER_REGION]	遮挡区域的句柄。
astAttr[VI_COVER_REGION]	遮挡区域的属性。

【注意事项】

无。

【相关数据类型及接口】

- REGION_HANDLE
- COVER_ATTR_S

OVERLAY_S

【说明】

定义所有叠加区域结构体。

【定义】

```
typedef struct hiOVERLAY_S
{
    HI_U32          u32OverlayNum;
    REGION_HANDLE   aOverlayHandles[VENC_OVERLAY_REGION];
    OVERLAY_ATTR_S  astAttr[VENC_OVERLAY_REGION];
}OVERLAY_S;
```

【成员】

成员名称	描述
u32OverlayNum	叠加区域的个数。
aOverlayHandles[VENC_OVERLAY_REGION]	叠加区域的句柄。
astAttr[VENC_OVERLAY_REGION]	叠加区域的属性。

【注意事项】

无。

【相关数据类型及接口】

- REGION_HANDLE
- OVERLAY_ATTR_S

SOFT_OVERLAY_S

【说明】

定义所有软叠加区域结构体。

【定义】

```
typedef struct hiSOFT_OVERLAY_S
{
    HI_U32 u32SoftOverlayNum;
    REGION_HANDLE aSoftOverlayHandles[VI_SOVERLAY_REGION];
    SOFT_OVERLAY_ATTR_S astAttr[VI_SOVERLAY_REGION];
}SOFT_OVERLAY_S;
```

【成员】

成员名称	描述
u32OverlayNum	软叠加区域的个数。
aOverlayHandles[VENC_OVERLAY_REGION]	软叠加区域的句柄。
astAttr[VENC_OVERLAY_REGION]	软叠加区域的属性。

【注意事项】

无。

【相关数据类型及接口】

- REGION_HANDLE
- SOFT_OVERLAY_ATTR_S

MOSAIC_S

【说明】

定义所有马赛克区域结构体。

【定义】

```
typedef struct hiMOSAIC_S
{
    HI_U32 u32MosaicNum;
    REGION_HANDLE aMosaicHandles[VI_MOSAIC_REGION];
    MOSAIC_ATTR_S astAttr[VI_MOSAIC_REGION];
}MOSAIC_S;
```

【成员】

成员名称	描述
u32MosaicNum	马赛克区域的个数。
aMosaicHandles[VI_MOSAIC_REGION]	马赛克区域的句柄。
astAttr[VI_MOSAIC_REGION]	马赛克区域的属性。

【注意事项】

无。

【相关数据类型及接口】

- REGION_HANDLE
- SOFT_OVERLAY_ATTR_S

REGION_CTRL_PARAM_U

【说明】

定义控制区域参数联合体。

【定义】

```
typedef union hiREGION_CTRL_PARAMETER_U
{
    HI_U32          u32Layer;
    HI_U32          u32Alpha;
    HI_U32          u32Color;
    POINT_S         stPoint;
    DIMENSION_S     stDimension;
    BITMAP_S        stBitmap;
    REGION_ATTR_S   stRegionAttr;
    COVER_S         stCovers;
    OVERLAY_S       stOverlays;
    SOFT_OVERLAY_S  stSoftOverlays;
}REGION_CTRL_PARAM_U;
```

【成员】

成员名称	描述
u32Layer	区域层次（只针对遮挡区域）。 取值范围：[0, 100]。 动态属性。

成员名称	描述
u32Alpha	区域 Alpha 值（只针对叠加区域且像素格式为 ARGB1555）。 取值范围：[0, 128]。 动态属性。
u32Color	背景色。 取值范围：请参见 COVER_ATTR_S 和 OVERLAY_ATTR_S 的相应取值范围。 动态属性。
stPoint	位置。 取值范围：请参见 COVER_ATTR_S 和 OVERLAY_ATTR_S 的相应取值范围。 动态属性。
stDimension	高宽（只针对遮挡区域）。 取值范围：请参见 COVER_ATTR_S 和 OVERLAY_ATTR_S 的相应取值范围。 动态属性。
stBitmap	位图结构体（只针对叠加区域）。
stRegionAttr	区域属性。
stCovers	所有遮挡区域。
stOverlays	所有叠加区域。
stSoftOverlays	所有软叠加区域。

【注意事项】

无。

【相关数据类型及接口】

HI_MPI_VPP_ControlRegion

PIC_SCALE_TASK_S

【说明】

定义缩放任务结构体。

【定义】

```
typedef struct hiPIC_SCALE_TASK_S
{
    HI_U32              u32TaskId;  /* used to identify a task */
    VIDEO_FRAME_INFO_S  stSrcPic;   /* source picture */
```

```
    VIDEO_FRAME_INFO_S  stDesPic;   /* destination picture */
}PIC_SCALE_TASK_S;
```

【成员】

成员名称	描述
u32TaskId	任务 ID。
stSrcPic	源图像。
stDesPic	目标图像。

【注意事项】

无。

【相关数据类型及接口】

- HI_MPI_VPP_CreateScaleTask
- HI_MPI_VPP_WaitScaleTask

3.4.5 视频编码

相关数据类型、数据结构定义如下：

- H264E_NALU_TYPE_E：定义 H.264 码流 NALU 类型。
- JPEGE_PACK_TYPE_E：定义 JPEG 码流的 PACK 类型。
- MPEG4E_PACK_TYPE_E：定义 MPEG4 码流的 PACK 类型。
- VENC_DATA_TYPE_U：定义码流结果联合体。
- VENC_PACK_S：定义帧码流包结构体。
- VENC_STREAM_S：定义帧码流类型结构体。
- VENC_ATTR_H264_S：定义 H.264 编码属性结构体。
- VENC_ATTR_MJPEG_S：定义 MJPEG 编码属性结构体。
- VENC_ATTR_JPEG_S：定义 JPEG 抓拍编码属性结构体。
- VENC_ATTR_MPEG4_S：定义 MPEG4 编码通道属性结构体。
- VENC_ATTR_MEPARA_S：定义视频编码搜索窗设置结构体。
- H264_VENC_CAPABILITY_S：定义 H.264 私有能力集描述结构体。
- JPEG_VENC_CAPABILITY_S：定义 JPEGE 私有能力集描述结构体。
- VENC_CHN_ATTR_S：定义编码通道属性结构体。
- VENC_CHN_STAT_S：定义编码通道的状态结构体。
- VENC_CAPABILITY_S：定义编码通道编码能力集结构体。
- VENC_WM_ATTR_S：定义编码的数字水印的结构体。
- VENC_ATTR_H264_RC_S：定义 H264 编码的量化系数的结构体。
- VENC_ATTR_H264_NALU_S：定义 H.264 编码的 nalu 大小设置结构体。

H264E_NALU_TYPE_E

【说明】

定义 H.264 码流 NALU 类型。

【定义】

```
typedef enum hiH264E_NALU_TYPE_E
{
    H264E_NALU_PSLICE   = 1,
    H264E_NALU_ISLICE   = 5,
    H264E_NALU_SEI      = 6,
    H264E_NALU_SPS      = 7,
    H264E_NALU_PPS      = 8,
    H264E_NALU_BUTT
} H264E_NALU_TYPE_E;
```

【成员】

成员名称	描述
H264E_NALU_PSLICE	PSLICE 类型。
H264E_NALU_ISLICE	ISLICE 类型。
H264E_NALU_SEI	SEI 类型。
H264E_NALU_SPS	SPS 类型。
H264E_NALU_PPS	PPS 类型。

【注意事项】

无。

【相关数据类型及接口】

无。

JPEGE_PACK_TYPE_E

【说明】

定义 JPEG 码流的 PACK 类型。

【定义】

```
typedef enum hiJPEGE_PACK_TYPE_E
{
    JPEGE_PACK_ECS = 5,
    JPEGE_PACK_APP = 6,
```

```
    JPEGE_PACK_VDO = 7,
    JPEGE_PACK_PIC = 8,
    JPEGE_PACK_BUTT
} JPEGE_PACK_TYPE_E;
```

【成员】

成员名称	描述
JPEGE_PACK_ECS	ECS 类型。
JPEGE_PACK_APP	APP 类型。
JPEGE_PACK_VDO	VDO 类型。
JPEGE_PACK_PIC	PIC 类型。

【注意事项】

无。

【相关数据类型及接口】

无。

MPEG4E_PACK_TYPE_E

【说明】

定义 MPEG4 码流的 PACK 类型。

【定义】

```
typedef enum hiMPEG4E_PACK_TYPE_E
{
    MPEG4E_PACK_VO  = 1,
    MPEG4E_PACK_VOS = 2,
    MPEG4E_PACK_VOL = 3,
    MPEG4E_PACK_VOP = 4,
    MPEG4E_PACK_SLICE = 5
} MPEG4E_PACK_TYPE_E;
```

【成员】

成员名称	描述
MPEG4E_PACK_VO	VO 类型。
MPEG4E_PACK_VOS	VOS 类型。
MPEG4E_PACK_VOL	VOL 类型。

成员名称	描述
MPEG4E_PACK_VOP	VOP 类型。

【注意事项】

此版本暂不支持 MPEG4 编码。

【相关数据类型及接口】

无。

VENC_DATA_TYPE_U

【说明】

定义码流结果类型。

【定义】

```
typedef union hiVENC_DATA_TYPE_U
{
    H264E_NALU_TYPE_E   enH264EType;
    JPEGE_PACK_TYPE_E   enJPEGEType;
    MPEG4E_PACK_TYPE_E  enMPEG4EType;
}VENC_DATA_TYPE_U;
```

【成员】

成员名称	描述
enH264EType	H.264 码流包类型。
enJPEGEType	JPEG 码流包类型。
enMPEG4EType	MPEG4 码流包类型。

【注意事项】

此版本暂不支持 MPEG4 编码。

【相关数据类型及接口】

- H264E_NALU_TYPE_E
- JPEGE_PACK_TYPE_E
- MPEG4E_PACK_TYPE_E

VENC_PACK_S

【说明】

定义帧码流包结构体。

【定义】

```
typedef struct hiVENC_PACK_S
{
    HI_U32              u32PhyAddr[2];
    HI_U8               *pu8Addr[2];
    HI_U32              u32Len[2];
    VENC_DATA_TYPE_U    DataType;

    HI_U64              u64PTS;
    HI_BOOL             bFieldEnd;
    HI_BOOL             bFrameEnd;
}VENC_PACK_S;
```

【成员】

成员名称	描述
pu8Addr[2]	码流包首地址。
u32PhyAddr[2]	码流包物理地址。
u32Len[2]	码流包长度。
DataType	码流类型。
u64PTS	时间戳。
bFieldEnd	场结束标识。 取值范围： HI_TRUE：该码流包是该场的最后一个包。 HI_FALSE：该码流包不是该场的最后一个包。
bFrameEnd	帧结束标识。 取值范围： HI_TRUE：该码流包是该帧的最后一个包。 HI_FALSE：该码流包不是该场的最后一个包。

【注意事项】

无。

【相关数据类型及接口】

VENC_DATA_TYPE_U

VENC_STREAM_S

【说明】

定义帧码流类型结构体。

【定义】

```
typedef struct hiVENC_STREAM_S
{
    VENC_PACK_S *pstPack;
    HI_U32      u32PackCount;
    HI_U32      u32Seq;
}VENC_STREAM_S;
```

【成员】

成员名称	描述
pstPack	帧码流包结构。
u32PackCount	一帧码流的所有包的个数。
u32Seq	码流序列号。按帧获取帧序号；按包获取包序号。

【注意事项】

无。

【相关数据类型及接口】

- VENC_PACK_S
- HI_MPI_VENC_GetStream

VENC_ATTR_H264_S

【说明】

定义 H.264 编码属性结构体。

【定义】

```
typedef struct hiVENC_ATTR_H264_S
{
    HI_U32  u32Priority;
    HI_U32  u32PicWidth;
    HI_U32  u32PicHeight;
    HI_U32  u32ViFramerate;
    HI_U32  u32TargetFramerate;
    HI_BOOL bMainStream;
    HI_BOOL bVIField;
```

```
    HI_BOOL bField;
    HI_BOOL bCBR;
    HI_U32  u32Bitrate;
    HI_U32  u32PicLevel;
    HI_U32  u32Gop;
    HI_U32  u32MaxDelay;
    HI_U32  u32BufSize;
    HI_BOOL bByFrame;
}VENC_ATTR_H264_S;
```

【成员】

成员名称	描述
u32Priority	通道优先级。
u32PicWidth	编码图像宽度。 取值范围：[160, 2048]，以像素为单位。 静态属性。
u32PicHeight	编码图像宽度。 取值范围：[112, 1536]，以像素为单位。 静态属性。
u32ViFramerate	VI 输入的帧率（原始帧率）。 取值范围： • P 制：(0, 25]，以帧为单位。 • N 制：(0, 30]，以帧为单位。 静态属性。
u32TargetFramerate	目标帧率。 取值范围： • P 制：(0, 25]，以帧为单位。 • N 制：(0, 30]，以帧为单位。 动态属性。
bMainStream	主次码流标识。 取值范围：{HI_TRUE, HI_FALSE}。 • HI_TRUE：主码流。 • HI_FALSE：次码流。 静态属性。

成员名称	描述
bVIField	输入图像的帧场标志。 取值范围：{HI_TRUE, HI_FALSE}。 • HI_TRUE：场。 • HI_FALSE：帧。 静态属性。
bField	帧场编码模式。 取值范围：{HI_TRUE, HI_FALSE}。 • HI_TRUE：场编码。 • HI_FALSE：帧编码。 静态属性。
bCBR	VBR/CBR 模式。 取值范围：{HI_TRUE, HI_FALSE}。 • HI_TRUE：CBR。 • HI_FALSE：VBR。 动态属性。
u32Bitrate	在 CBR 模式下，指目标码率。在 VBR 的模式下，指的是最大码率。 取值范围：[16K, 20M]，以 bit/s 为单位。 动态属性。
u32PicLevel	图像等级，其意义取决于 VBR/CBR 模式。 VBR 模式下，表示图像的质量等级。 取值范围：[0, 5]，值越小，图像质量越好。 CBR 模式下，表示码率波动范围。 取值范围：[0, 5]。 0：由 SDK 软件自行控制码率，推荐使用。 1～5：对应码率波动范围分别为±10%～±50%。
u32Gop	I 帧间隔。 取值范围：[0, 1000]，以帧为单位。 动态属性。
u32MaxDelay	最大延迟。 取值范围：最大延迟，以帧为单位。 动态属性。

成员名称	描述
u32BufSize	码流 buffer 大小。 取值范围：[Min, Max]，以 byte 为单位。 最小值为存放原始图像所需内存的 1.5 倍，以 D1 为例，最小值为 704%576%1.5%1.5 Bytes；最大值无限制，但是太大可能导致缓存不够。推荐使用最小值。 静态属性。
bByFrame	帧/包模式获取码流。 取值范围：{HI_TRUE, HI_FALSE}。 • HI_TRUE：按帧获取。 • HI_FALSE：按包获取。 静态属性。

【注意事项】

无。

【相关数据类型及接口】

无。

VENC_ATTR_MJPEG_S

【说明】

定义 MJPEG 编码属性结构体。

【定义】

```
typedef struct hiVENC_ATTR_MJPEG_S
{
    HI_U32  u32Priority;
    HI_U32  u32BufSize;
    HI_U32  u32PicWidth;
    HI_U32  u32PicHeight;
    HI_BOOL bByFrame;
    HI_U32  u32ViFramerate;
    HI_BOOL bMainStream;
    HI_BOOL bVIField;
    HI_U32  u32TargetBitrate;
    HI_U32  u32TargetFramerate;
    HI_U32  u32MCUPerECS;
}VENC_ATTR_MJPEG_S;
```

【成员】

成员名称	描述
u32Priority	通道优先级。
u32BufSize	配置 buffer 大小。 取值范围：不小于图像宽高乘积的 1.5 倍。 静态属性。
u32PicWidth	编码图像宽度。 取值范围：[160, 2048]，以像素为单位。 静态属性。
u32PicHeight	编码图像高度。 取值范围：[112, 1536]，以像素为单位。 静态属性。
bByFrame	获取码流模式。 取值范围：{HI_TRUE, HI_FALSE}。 • HI_TRUE：按帧获取。 • HI_FALSE：按包获取。 静态属性。
u32ViFramerate	原始帧率。 取值范围： • PAL：[0, 25]，以帧为单位。 • NTSC：[0, 30]，以帧为单位。 静态属性。
bMainStream	主次码流标志。 取值范围：{HI_TRUE, HI_FALSE}。 • HI_TRUE：主码流。 • HI_FALSE：次码流。 静态属性。
bVIField	输入图像的帧场标志。 取值范围：{HI_TRUE, HI_FALSE}。 • HI_TRUE：场。 • HI_FALSE：帧。 静态属性。
u32TargetBitrate	目标码率。 取值范围：[1M, 20M]，以 kbit/s 为单位。 动态属性。

成员名称	描述
u32TargetFramerate	目标帧率。 取值范围： P 制（0, 25]，以帧为单位。 N 制（0, 30]，以帧为单位。 动态属性。
u32MCUPerECS	每个 ECS 中 MCU 个数。 取值范围：[0, MCU 总数]。建议不小于 16 个。 MCU 总数即图像中宏块（16 像素 ％16 像素）个数。 动态属性。 本版本只支持 u32MCUPerECS=heigh*width/256。

【注意事项】

无。

【相关数据类型及接口】

无。

VENC_ATTR_JPEG_S

【说明】

定义 JPEG 抓拍属性结构体。

【定义】

```
typedef struct hiVENC_ATTR_JPEG_S
{
    HI_U32  u32Priority;
    HI_U32  u32BufSize;
    HI_U32  u32PicWidth;
    HI_U32  u32PicHeight;
    HI_BOOL bVIField;
    HI_BOOL bByFrame;
    HI_U32  u32MCUPerECS;
    HI_U32  u32ImageQuality;
}VENC_ATTR_JPEG_S;
```

【成员】

成员名称	描述
u32Priority	通道优先级。

成员名称	描述
u32BufSize	配置 buffer 大小。 取值范围：不小于图像宽高乘积的 1.5 倍。 静态属性。
u32PicWidth	编码图像宽度。 取值范围：[80, 2048]。 静态属性。
u32PicHeight	编码图像高度。 取值范围：[64, 1536]。 静态属性。
bVIField	输入图像的帧场标志。 取值范围：{HI_TRUE, HI_FALSE}。 • HI_TRUE：场。 • HI_FALSE：帧。 静态属性。
bByFrame	获取码流模式,帧或包。 取值范围：{HI_TRUE, HI_FALSE}。 • HI_TRUE：按帧获取。 • HI_FALSE：按包获取。 静态属性。
u32MCUPerECS	每个 ECS 中 MCU 个数。 取值范围：[0, MCU 总数]。 建议不小于 16 个。 MCU 总数即图像中宏块（16 像素 % 16 像素）个数。 动态属性。 本版本只支持 u32MCUPerECS=heigh*width/256。
u32ImageQuality	抓拍图像质量。 取值范围：[0, 5]。 数字越小，图像质量越好。 动态属性。

【注意事项】

无。

【相关数据类型及接口】

无。

VENC_ATTR_MPEG4_S

【说明】

MPEG4 编码属性结构体。

【定义】

```
typedef struct  hiVENC_ATTR_MPEG4_S
{
    HI_U32  u32Priority;
    HI_U32  u32PicWidth;
    HI_U32  u32PicHeight;
    HI_U32  u32TargetBitrate;
    HI_U32  u32ViFramerate;
    HI_U32  u32TargetFramerate;
    HI_U32  u32Gop;
    HI_U32  u32MaxDelay;
    M4Qtype enQuantType;
    HI_U32  u32BufSize;
    HI_BOOL bVIField;
    HI_BOOL bByFrame;
}VENC_ATTR_MPEG4_S;
```

【成员】

成员名称	描述
u32Priority	通道优先级。 取值范围：(0, 7)。0 最高，7 最低。 静态属性。 目前无效。
u32PicWidth	图像宽度。 取值范围：QCIF（PAL：176 % 144，NTSC：176 % 112）。 静态属性。
u32PicHeight	图像高度。 取值范围：QCIF（PAL：176 % 144，NTSC：176 % 112）。 静态属性。

成员名称	描述
u32TargetBitrate	目标码率。 取值范围：[1, 64]，以 kbit/s 为单位。 动态属性。
u32ViFramerate	原始帧率。 以帧每秒（f/s）为单位。 静态属性。
u32TargetFramerate	目标帧率。 取值范围：[1, 15]。以帧每秒（f/s）为单位。 动态属性。
u32Gop	I 帧间隔。 动态属性。
u32MaxDelay	最大延迟。 取值范围：[5, 20]。 静态属性。
enQuantType	MPEG4 编码量化方式。 动态属性。
u32BufSize	配置码流 buffer 大小。 取值范围：应大于图像大小。 静态属性。
bVIField	输入图像的帧场标志。 静态属性。
bByFrame	帧/包模式获取码流方式。 静态属性。 目前只支持按帧获取。

【注意事项】

无。

【相关数据类型及接口】

无。

VENC_ATTR_MEPARA_S

【说明】

定义视频编码搜索窗设置结构体。

【定义】

```
typedef struct hiVENC_ATTR_MEPARA_S
{
    HI_S32 s32HWSize;
    HI_S32 s32VWSize;

    HI_S32 s32IterNum[8];
    HI_S32 s32Denoise[2];
    HI_S32 s32RefPicNum;

} VENC_ATTR_MEPARA_S;
```

【成员】

成员名称	描述
s32HWSize	水平搜索窗。 取值范围：[0, 2]。
s32VWSize	垂直搜索窗。 取值范围：[0, 1]。
s32IterNum[8]	保留。
s32Denoise[2]	保留。
s32RefPicNum	保留。

【注意事项】

无。

【相关数据类型及接口】

HI_MPI_VENC_CfgMestPara

H264_VENC_CAPABILITY_S

【说明】

H.264 私有能力集描述结构体。

【定义】

```
typedef struct hiH264_VENC_CAPABILITY_S
{
    HI_U8   u8Profile;
    HI_U8   u8Level;
    HI_U8   u8BaseAttr;
    HI_U8   u8ViFormat;
```

```
    HI_U8   u8MaxWInMb;
    HI_U8   u8MaxHInMb;
    HI_U16  u16MaxCifNum;
    HI_U16  u16MaxBitrate;
    HI_U16  upperbandwidth;
    HI_U16  lowerbandwidth;
    HI_U8   palfps;
    HI_U8   ntscfps;
}H264_VENC_CAPABILITY_S;
```

【成员】

成员名称	描述
u8Profile	0：baseline。 1：mainprofile。 2：externed profile。
u8Level	例如 22 表示 level2.2。
u8BaseAttr	bit0～bit5 分别表示 MBAFF、PAFF、B SLICE、FMO、ASO 和 PARTITION。
u8ViFormat	支持输入制式。 bit0：PAL(25)。 bit1：NTSC(30)。
u8MaxWInMb	输入图像宽度的最大尺寸。
u8MaxHInMb	输入图像高度的最大尺寸。
u16MaxCifNum	最大编码能力，多通道总和。 以 CIF 为单位。
u16MaxBitrate	最大单通道输出码率能力。 以 kbit/s 为单位。
upperbandwidth	带宽上限。 以 kbit/s 为单位。
lowerbandwidth	带宽下限。 以 kbit/s 为单位。
palfps	PAL 制式帧率。 以帧每秒（f/s）为单位。
ntscfps	NTSC 制式帧率。 以帧每秒（f/s）为单位。

【注意事项】

无。

【相关数据类型及接口】

无。

JPEG_VENC_CAPABILITY_S

【说明】

JPEG 私有能力集描述结构体。

【定义】

```
typedef struct hiJPEG_VENC_CAPABILITY_S
{
    HI_U8   u8Profile;
    HI_U8   u8ViFormat;
    HI_U8   u8MaxWInMb;
    HI_U8   u8MaxHInMb;
    HI_U16  u16MaxCifNum;
    HI_U16  u16MaxBitrate;
    HI_U16  upperbandwidth;
    HI_U16  lowerbandwidth;
    HI_U8   palfps;
    HI_U8   ntscfps;
}JPEG_VENC_CAPABILITY_S;
```

【成员】

成员名称	描述
u8Profile	0：baseline。 1：extened profile。 2：loseless profile。 3：hierarchical profile。
u8ViFormat	支持输入制式。 bit0：PAL(25)。 bit1：NTSC(30)。
u8MaxWInMb	输入图像宽度的最大尺寸。
u8MaxHInMb	输入图像高度的最大尺寸。
u16MaxCifNum	最大编码能力，多通道总和。 以 CIF 为单位。

成员名称	描述
u16MaxBitrate	最大单通道输出码率能力。 以 kbit/s 为单位。
upperbandwidth	带宽上限。 以 kbit/s 为单位。
lowerbandwidth	带宽下限。 以 kbit/s 为单位。
palfps	PAL 制式帧率。 以帧每秒（f/s）为单位。
ntscfps	NTSC 制式帧率。 以帧每秒（f/s）为单位。

【注意事项】

无。

【相关数据类型及接口】

无。

VENC_CHN_ATTR_S

【说明】

定义编码通道属性结构体。

【定义】

```
typedef struct hiVENC_CHN_ATTR_S
{
    PAYLOAD_TYPE_E  enType;
    HI_VOID         *pValue;
}VENC_CHN_ATTR_S;
```

【成员】

成员名称	描述
enType	编码协议类型。
pValue	编码属性指针。

【注意事项】

无。

【相关数据类型及接口】

HI_MPI_VENC_CreateChn

VENC_CHN_STAT_S

【说明】

定义编码通道的状态结构体。

【定义】

```
typedef struct hiVENC_CHN_STAT_S
{
    HI_BOOL bRegistered;
    HI_U32  u32LeftPics;
    HI_U32  u32LeftStreamBytes;
    HI_U32  u32CurPacks;
}VENC_CHN_STAT_S;
```

【成员】

成员名称	描述
bRegistered	注册到通道组标志。 取值范围：{HI_TRUE, HI_FALSE}。 • HI_TRUE：注册。 • HI_FALSE：未注册。
u32LeftPics	待编码的图像数。
u32LeftStreamBytes	码流 buffer 剩余的 byte 数。
u32CurPacks	当前帧的码流包个数。

【注意事项】

无。

【相关数据类型及接口】

HI_MPI_VENC_Query

VENC_CAPABILITY_S

【说明】

定义编码通道编码能力集结构体。

【定义】

```
typedef struct hiVENC_CAPABILITY_S
```

```
{
    PAYLOAD_TYPE_E  enType;
    HI_VOID         *pCapability;
}VENC_CAPABILITY_S;
```

【成员】

成员名称	描述
enType	编码协议类型。
pCapability	能力集。

【注意事项】

无。

【相关数据类型及接口】

HI_MPI_VENC_GetCapability

VENC_WM_ATTR_S

【说明】

定义编码的数字水印的结构体。

【定义】

```
#define     DWM_KEY_LEN     8       /* 密钥字符的最大个数 */
#define     DWM_CHAR_LEN    16      /*水印字符个数*/
typedef struct hiVENC_WM_ATTR_S
{
    HI_U8   au8Key[DWM_KEY_LEN];    /* 数字水印密钥字符串 */
    HI_U8   au8User[DWM_CHAR_LEN]; /* 数字水印用户字符 */
}VENC_WM_ATTR_S;
```

【成员】

成员名称	描述
au8Key	数字水印的密钥字符串，最多 8 个字符，不满 8 个字符填充 0。
au8User	数字水印用户字符，个数最多不超过 DWM_CHAR_LEN。

【注意事项】

无。

【相关数据类型及接口】

HI_MPI_VENC_SetWmAttr

VENC_ATTR_H264_RC_S

【说明】

定义 H.264 编码的量化系数的结构体。

【定义】

```
typedef struct hiVENC_ATTR_H264_RC_S
{
    HI_S32 s32MinQP; /*H264编码的最小QP值*/

} VENC_ATTR_H264_RC_S;
```

【成员】

成员名称	描述
s32MinQP	H.264 编码的最小量化系数。 取值范围：[4, 50]。

【注意事项】

无。

【相关数据类型及接口】

HI_MPI_VENC_SetH264eRcPara

VENC_ATTR_H264_NALU_S

【说明】

定义 H.264 编码的 nalu 大小设置结构体。

【定义】

```
typedef struct hiVENC_ATTR_H264_NALU_S
{
    HI_BOOL bNaluSplitEnable;
    HI_U32  u32NaluSize;
} VENC_ATTR_H264_NALU_S;
```

【成员】

成员名称	描述
bNaluSplitEnable	指示是否打开 Nalu 划分。 HI_TURE：打开使能。 HI_FALSE：关闭使能。
u32NaluSize	Nalu 划分使能的情况下指定 Nalu 的大小，以字节为单位，在关闭使能的情况下，此参数无效。 必须满足 128 u32NaluSize 图象大小(包括色度)。

【注意事项】

无。

【相关数据类型及接口】

- HI_MPI_VENC_SetH264eNaluPara
- HI_MPI_VENC_GetH264eNaluPara

3.4.6 移动侦测

相关数据类型、数据结构定义如下：

- MD_MB_MODE_S：定义宏块模式结构体。
- MD_DLIGHT_S：定义去关照效应属性结构体。
- MD_SADBITS_E：定义运动侦测的 SAD 的精度。
- MD_REF_MODE_E：定义参考图像模式。
- MD_REF_STATUS_E：定义参考图像更新状态。
- MD_REF_ATTR_S：定义参考图像属性结构体。
- MD_MB_DATA_S：定义宏块结果结构体。
- MD_DATA_S：定义运动侦测结果结构体。
- MD_CHN_ATTR_S：定义运动侦测属性结构体。

MD_MB_MODE_S

【说明】

定义宏块模式结构体。

【定义】

```
typedef struct hiMD_MB_MODE_S
{
    HI_BOOL bMBSADMode;
    HI_BOOL bMBMVMode;
    HI_BOOL bMBALARMMode;
    HI_BOOL bMBPelNumMode;
```

```
} MD_MB_MODE_S;
```

【成员】

成员名称	描述
bMBSADMode	宏块 SAD 模式开关。 取值范围：{HI_TRUE,HI_FALSE}。 • HI_TRUE：打开。 • HI_FALSE：关闭。
bMBMVMode	宏块 Mv 模式开关。 取值范围：{HI_TRUE, HI_FALSE}。 • HI_TRUE：打开。 • HI_FALSE：关闭。
bMBALARMMode	宏块报警模式开关。 取值范围：{HI_TRUE, HI_FALSE}。 • HI_TRUE：打开。 • HI_FALSE：关闭。
bMBPelNumMode	宏块报警像素模式开关。 取值范围：{HI_TRUE,HI_FALSE}。 • HI_TRUE：打开。 • HI_FALSE：关闭。

【注意事项】

无。

【相关数据类型及接口】

无。

MD_DLIGHT_S

【说明】

定义去光照效应属性结构体。

【定义】

```
typedef struct hiMD_DLIGHT_S
{
    HI_BOOL bEnable;
    HI_U8   u8DlBeta;
    HI_U8   u8DlAlpha;
    HI_U16  Reserved;
```

```
} MD_DLIGHT_S;
```

【成员】

成员名称	描述
bEnable	去光照效应开关。 取值范围：{HI_TRUE, HI_FALSE}。 • HI_TRUE：打开。 • HI_FALSE：关闭。
u8DlBeta	去关照应 Beta 值。 取值范围：[0, 7]。
u8DlAlpha	去关照应 Alpha 值。 取值范围：[0, 7]。
Reserved	保留。

【注意事项】

目前此结构体设置无效。

【相关数据类型及接口】

无。

MD_SADBITS_E

【说明】

定义运动侦测的 SAD 的精度。

【定义】

```
typedef enum hiMD_SADBITS_E
{
    MD_SAD_8BIT = 0,
    MD_SAD_16BIT,
    MD_SAD_BUTT
} MD_SADBITS_E;
```

【成员】

成员名称	描述
MD_SAD_8BIT	SAD 精度为 8bit。
MD_SAD_16BIT	SAD 精度为 16bit。

【注意事项】

不管精度值如何，在运动侦测结果中，每个宏块的 SAD 值都要占用 2byte 内存。

【相关数据类型及接口】

无。

MD_REF_MODE_E

【说明】

定义参考图像模式。

【定义】

```
typedef enum hiMD_REF_MODE_E
{
    MD_REF_AUTO = 0,
    MD_REF_USER,
    MD_REF_MODE_BUTT
} MD_REF_MODE_E;
```

【成员】

成员名称	描述
MD_REF_AUTO	自动设置参考图像模式。
MD_REF_USER	用户输入参考图像模式。

【注意事项】

无。

【相关数据类型及接口】

无。

MD_REF_STATUS_E

【说明】

定义参考图像更新状态。

【定义】

```
typedef enum hiMD_REF_STATUS_E
{
    MD_REF_STATIC = 0,
    MD_REF_DYNAMIC,
    MD_REF_STATUS_BUTT
} MD_REF_STATUS_E;
```

【成员】

成员名称	描述
MD_REF_STATIC	参考图像不更新。
MD_REF_DYNAMIC	参考图像更新。

【注意事项】

参考图像是否更新均是在参考图像模式为自动模式的情况下。参考图像为用户设置模式时，此参考图像状态无效。

【相关数据类型及接口】

无。

MD_REF_ATTR_S

【说明】

定义参考图像属性结构体。

【定义】

```
typedef struct hiMD_REF_ATTR_S
{
    MD_REF_MODE_E   enRefFrameMode;
    MD_REF_STATUS_E enRefFrameStat;
    VIDEO_FRAME_S   stUserRefFrame;
} MD_REF_ATTR_S;
```

【成员】

成员名称	描述
enRefFrameMode	参考图像模式。
enRefFrameStat	参考图像更新状态。
stUserRefFrame	用户输入模式，参考图像的信息结构体。

【注意事项】

参考图像是否更新均是在参考图像模式为自动模式的情况下。参考图像为用户设置模式时，此参考图像更新状态无效，在此模式下用户还需配置参考图像。

【相关数据类型及接口】

- MD_REF_MODE_E
- MD_REF_STATUS_E

- HI_MPI_MD_SetRefFrame
- HI_MPI_MD_GetRefFrame

MD_MB_DATA_S

【说明】

定义宏块结果结构体。

【定义】

```
typedef struct hiMD_MB_DATA_S
{
    HI_U32* pu32Addr;
    HI_U32  u32Stride;
} MD_MB_DATA_S;
```

【成员】

成员名称	描述
pu32Addr	宏块数据指针。
u32Stride	宏块数据以行为单位的内存宽度。

【注意事项】

无。

【相关数据类型及接口】

无。

MD_DATA_S

【说明】

定义运动侦测结果结构体。

【定义】

```
typedef struct hiMD_DATA_S
{
    HI_U32*         pu32Addr;
    HI_U16          u16MBWidth;
    HI_U16          u16MBHeight;
    HI_U64          u64Pts;

    MD_MB_MODE_S    stMBMode;
    MD_MB_DATA_S    stMBSAD;
    MD_MB_DATA_S    stMBMV;
```

```
    MD_MB_DATA_S    stMBAlarm;
    MD_MB_DATA_S    stMBPelAlarmNum;
} MD_DATA_S;
```

【成员】

成员名称	描述
pu32Addr	MD 结果地址信息，释放结果时需用到。 不允许修改。
u16MBWidth	图像的宽度。以宏块为单位。
u16MBHeight	图像的高度。以宏块为单位。
u64Pts	时间戳。
stMBMode	宏块模式。
stMBSAD	宏块 SAD 值。
stMBMV	宏块 MV 值。
stMBAlarm	宏块报警信息。
stMBPelAlarmNum	宏块报警像素的个数。

【注意事项】

无。

【相关数据类型及接口】

- MD_MB_MODE_S
- MD_MB_DATA_S
- HI_MPI_MD_GetData
- HI_MPI_MD_ReleaseData

MD_CHN_ATTR_S

【说明】

定义运动侦测属性结构体。

【定义】

```
typedef struct hiMD_CHN_ATTR_S
{
    MD_MB_MODE_S    stMBMode;
    MD_SADBITS_E    enSADBits;
    MD_DLIGHT_S     stDlight;
    HI_U8           u8MBPelALTh;
```

```
    HI_U8           u8MBPerALNumTh;
    HI_U16          u16MBALSADTh;
    HI_U32          u32MDInternal;
    HI_U32          u32MDBufNum;
} MD_CHN_ATTR_S;
```

【成员】

成员名称	描述
stMBMode	宏块模式。
enSADBits	SAD 输出精度。
stDlight	去光照效应属性。
u8MBPelALTh	像素报警阈值。 取值范围：(0, 255)。
u8MBPerALNumTh	像素报警个数阈值。 取值范围：(0, 255)。
u16MBALSADTh	宏块报警阈值。 取值范围： • 去光照效应打开时：(0, 255)。 • 去光照效应不打开时：(0, 65535)。
u32MDInternal	MD 侦测间隔。 取值范围：[0, 256]，以帧为单位。
u32MDBufNum	MD 缓存大小。 取值范围：[1, 16]。

【注意事项】

无。

【相关数据类型及接口】

- MD_MB_MODE_S
- MD_SADBITS_E
- MD_DLIGHT_S
- HI_MPI_MD_SetChnAttr
- HI_MPI_MD_GetChnAttr

3.4.7 视频解码

视频解码相关数据类型、数据结构定义如下：

- VDEC_CHN_ATTR_S：定义视频解码通道属性。

- VDEC_ATTR_JPEG_S：定义使用 JPEG 解码协议的视频解码通道属性结构体。
- VDEC_ATTR_H264_S：定义使用 H.264 解码协议的视频解码通道属性结构体。
- VDEC_STREAM_S：定义解码码流结构体。
- VDEC_USERDATA_S：定义用户私有数据结构体。
- VDEC_CHN_STAT_S：定义通道状态结构体。
- VDEC_FRAME_INFO_S：定义解码帧信息结构体。
- VDEC_DATA_S：定义解码数据结构体。
- VDEC_WM_ATTR_S：定义解码数字水印设置结构体。
- H264_VDEC_CAPABILITY_S：定义 H.264 解码私有能力集描述结构体。
- JPEG_VDEC_CAPABILITY_S：定义 JPEG 解码私有能力集描述结构体。
- VDEC_CAPABILITY_S：定义解码通道能力集。
- H264D_MODE_E：定义码流发送方式。

VDEC_CHN_ATTR_S

【说明】

定义视频解码通道属性结构体。

【定义】

```
typedef struct hiVDEC_CHN_ATTR_S
{
    PAYLOAD_TYPE_E  enType;
    HI_U32          u32BufSize;
    HI_VOID         *pValue;
}VDEC_CHN_ATTR_S;
```

【成员】

成员名称	描述
enType	解码协议类型枚举值。 静态属性。
u32BufSize	码流缓存的大小。 取值范围：大于等于解码通道大小（宽%高）的 1.5 倍，以 byte 为单位。 静态属性。
pValue	void*类型，指向解码协议的属性的指针。

【描述】

无。

【相关数据类型及接口】

无。

VDEC_ATTR_JPEG_S

【说明】

定义使用 JPEG 解码协议的视频解码通道属性。

【定义】

```
typedef struct hiVDEC_ATTR_JPEG_S
{
    HI_U32  u32Priority;
    HI_U32  u32PicWidth;
    HI_U32  u32PicHeight;
}VDEC_ATTR_JPEG_S;
```

【成员】

成员名称	描述
u32Priority	通道优先级。
u32PicWidth	解码图像宽度。 取值范围：[64, 4096]。 静态属性。
u32PicHeight	解码图像高度。 取值范围：[32, 4096]。 静态属性。

【注意事项】

无。

【相关数据类型及接口】

无。

VDEC_ATTR_H264_S

【说明】

定义使用 H.264 解码协议的视频解码通道属性。

【定义】

```
typedef struct hiVDEC_ATTR_H264_S
{
```

```
    HI_U32  u32Priority;
    HI_U32  u32PicWidth;
    HI_U32  u32PicHeight;
    HI_U32  u32RefFrameNum;
    H264D_MODE_E enMode;
}VDEC_ATTR_H264_S;
```

【成员】

成员名称	描述
u32Priority	通道优先级。
u32PicWidth	解码图像宽度。 取值范围：最大支持 D1 大小的解码。 静态属性。
u32PicHeight	解码图像高度。 取值范围：最大支持 D1 大小的解码。 静态属性。
u32RefFrameNum	参考帧的数目。 取值范围：[2, 16]，以帧为单位。 参考帧的数目决定解码时需要的参考帧个数，会较大的影响内存占用，根据实际情况设置合适的值。 • 海思自编码流：推荐设为 2。此时就是快速输出。 • 其他监控码流：推荐设为 5。 • 测试码流：推荐设为 16。 静态属性。
enMode	码流发送方式。 支持按帧或者按流。 静态属性。

【注意事项】

无。

【相关数据类型及接口】

无。

VDEC_STREAM_S

【说明】

定义视频解码的码流结构体。

【定义】

```
typedef struct hiVDEC_STREAM_S
{
    HI_U8*  pu8Addr;
    HI_U32  u32Len;
    HI_U64  u64PTS;
}VDEC_STREAM_S;
```

【成员】

成员名称	描述
pu8Addr	码流包的地址。
u32Len	码流包的长度。 以 byte 为单位。
u64PTS	码流包的时间戳。 以 μs 为单位。

【注意事项】

无。

【相关数据类型及接口】

无。

VDEC_USERDATA_S

【说明】

定义用户私有数据结构体。

【定义】

```
typedef struct hiVDEC_PRIDATA_S
{
    HI_U8*  pu8Addr;
    HI_U32  u32Len;
    HI_BOOL bValid;
}VDEC_USERDATA_S;
```

【成员】

成员名称	描述
pu8Addr	私有数据的地址。

成员名称	描述
u32BufSize	私有数据的长度。 以 byte 为单位。
bValid	当前私有数据的有效标识。 取值范围：{HI_TRUE, HI_FALSE}。 • HI_TRUE：有效。 • HI_FALSE：无效。

【注意事项】

无。

【相关数据类型及接口】

无。

VDEC_CHN_STAT_S

【说明】

定义通道状态结构体。

【定义】

```
typedef struct hiVDEC_CHN_STAT_S
{
    HI_U32  u32LeftStreamBytes; /*待解码byte数*/
    HI_U32 u32LeftStreamFrames;/*待解码的frame数*/
    HI_U32  u32LeftPics;        /*待获取的pic数*/
    HI_BOOL bStartRecvStream;   /*是否允许接收码流*/
}VDEC_CHN_STAT_S;
```

【成员】

成员名称	描述
u32LeftStreamBytes	码流 buffer 中待解码的 byte 数。
u32LeftStreamFrames	码流 buffer 中待解码的帧数。 –1 表示无效。 仅 H.264 解码且按帧发送时有效。
u32LeftPics	图像 buffer 中剩余的 pic 数目。
bStartRecvStream	解码器是否已经启动了接收码流。

【注意事项】

无。

【相关数据类型及接口】

无。

VDEC_FRAME_INFO_S

【说明】

定义视频解码帧信息结构体。

【定义】

```
typedef struct hiVDEC_FRAME_S
{
    VIDEO_FRAME_INFO_S  stVideoFrameInfo;
    HI_BOOL             bValid;
}VDEC_FRAME_INFO_S;
```

【成员】

成员名称	描述
stVideoFrameInfo	解码视频帧信息结构体。
bValid	当前解码视频帧的有效标识。 取值范围：{HI_TRUE, HI_FALSE}。 • HI_TRUE：有效。 • HI_FALSE：无效。

【注意事项】

无。

【相关数据类型及接口】

无。

VDEC_DATA_S

【说明】

定义视频解码数据结构体。

【定义】

```
typedef struct hiVDEC_DATA_S
{
    VDEC_FRAME_INFO_S   stFrameInfo;
```

```
    VDEC_WATERMARK_S    stWaterMark;
    VDEC_USERDATA_S     stUserData;
} VDEC_DATA_S;
```

【成员】

成员名称	描述
stFrameInfo	视频解码帧信息。
stWaterMark	数字水印（本版本不支持）。
stUserData	用户私有数据。

【注意事项】

无。

【相关数据类型及接口】

无。

VDEC_WM_ATTR_S

【说明】

定义解码数字水印设置结构体。

【定义】

```
typedef struct hiVDEC_WM_ATTR_S
{
    HI_U8       u8Key[DWM_KEY_LEN];
}VDEC_WM_ATTR_S;
```

【成员】

成员名称	描述
u8Key[DWM_KEY_LEN]	数字水印字符串。

【注意事项】

无。

【相关数据类型及接口】

HI_MPI_VDEC_CreateChn

H264_VDEC_CAPABILITY_S

【说明】

定义 H.264 解码私有能力集描述结构体。

【定义】

```
typedef struct hiH264_VDEC_CAPABILITY_S
{
    HI_U8 profile;
    HI_U8 level;
    HI_U8 FMO;
    HI_U8 ASO;
    HI_U8 MBAFF;
    HI_U8 PAFF;
    HI_U8 BSlice;
    HI_U8 subqcif;
    HI_U8 qcif;
    HI_U8 cif;
    HI_U8 fourcif;
    HI_U8 sixteencif;
    HI_U8 lostpacket;
    HI_U16 upperbandwidth;
    HI_U16 lowerbandwidth;
    HI_U8 palfps;
    HI_U8 ntscfps;
}H264_VDEC_CAPABILITY_S;;
```

【成员】

成员名称	描述
profile	0：baseline。 1：mainprofile
level	4:4，高 4 位表示整数位，低 4 位表示小数位。 例如 0x22 表示 level2.2。
FMO	Flexible Macroblock Ordering。 0：不支持。 1：支持。
ASO	Arbitrary Slice Ordering。 0：不支持。 1：支持。
MBAFF	macroblock-adaptive frame/field。 0：不支持。 1：支持。

成员名称	描述
PAFF	picture-adaptive frame/field。 0：不支持。 1：支持。
BSlice	B slice。 0：不支持。 1：支持。
subqcif	SubQCIF 图像。 0：不支持。 1：支持。
qcif	QCIF 图像。 0：不支持。 1：支持。
cif	CIF 图像。 0：不支持。 1：支持。
fourcif	4CIF 图像。 0：不支持。 1：支持。
sixteencif	16CIF 图像。 0：不支持。 1：支持。
lostpacket	丢包。 0：不支持。 1：支持。
upperbandwidth	带宽上限。 以 kbit/s 为单位。
lowerbandwidth	带宽下限。 以 kbit/s 为单位。
palfps	pal 制式帧率。 以帧/秒单位。
ntscfps	ntsc 制式帧率。 以帧/秒单位。

【注意事项】

无。

【相关数据类型及接口】

无。

JPEG_VDEC_CAPABILITY_S

【说明】

定义 JPEG 解码私有能力集描述结构体。

【定义】

```
typedef struct hiJPEG_VDEC_CAPABILITY_S
{
    HI_U32  enProcess;      /* 支持的JPEG编码过程 0:baseline 1:extened
    profile  2:loseless profile 3:hierarchical profile*/
    HI_U32  u32ComponentNum;/* 支持的分量个数 */
    HI_U32  u32QTNum;       /* 支持的量化表数目 */
    HI_BOOL  bYUV420;       /* 是否支持4:2:0采样格式 */
    HI_BOOL  bYUV422;       /* 是否支持4:2:2采样格式 */
    HI_BOOL  bYUV444;       /* 是否支持4:4:4采样格式 */
    HI_U8   lostpacket;     /*丢包：1支持；0不支持*/
    HI_U16  upperbandwidth; /*带宽上限 单位kbps*/
    HI_U16  lowerbandwidth; /*带宽下限 单位kbps*/
    HI_U8   palfps;         /*pal制式fps 单位帧/秒*/
    HI_U8   ntscfps;        /*ntsc制式fps 单位帧/秒*/
}JPEG_VDEC_CAPABILITY_S;
```

【成员】

成员名称	描述
enProcess	支持的 JPEG 编码过程。 0：baseline。 1：extened profile。 2：loseless profile。 3：hierarchical profile。
u32ComponentNum	支持的分量个数。
u32QTNum	支持的量化表数目。
bYUV420	是否支持 4:2:0 采样格式。

成员名称	描述
bYUV422	是否支持 4:2:2 采样格式。
bYUV444	是否支持 4:4:4 采样格式。
lostpacket	丢包。 0：不支持。 1：支持。
upperbandwidth	带宽上限。 以 kbit/s 为单位。
lowerbandwidth	带宽下限。 以 kbit/s 为单位。
palfps	pal 制式帧率。 以帧/秒单位。
ntscfps	ntsc 制式帧率。 以帧/秒单位。

【注意事项】

无。

【相关数据类型及接口】

无。

VDEC_CAPABILITY_S

【说明】

定义解码通道能力集结构体。

【定义】

```
typedef struct hiVDEC_CAPABILITY_S
{
    PAYLOAD_TYPE_E enType;
    HI_VOID *pCapability;
} VDEC_CAPABILITY_S;
```

【成员】

成员名称	描述
enType	解码协议类型。
pCapability	能力集。

【注意事项】

无。

【相关数据类型及接口】

无。

H264D_MODE_E

【说明】

定义码流发送方式。

【定义】

```
typedef enum hiH264D_MODE_E
{
    H264D_MODE_STREAM = 0,
    H264D_MODE_FRAME,
    H264D_MODE_BUTT
}H264D_MODE_E;
```

【成员】

成员名称	描述
H264D_MODE_STREAM	按流方式发送码流。
H264D_MODE_FRAME	发送码流。 以帧为单位。

【描述】

无。

【相关数据类型及接口】

无。

3.5 音频类数据类型

3.5.1 音频输入输出

音频输入输出相关数据类型、数据结构定义如下：

- AUDIO_SAMPLE_RATE_E：定义音频采样率。
- AUDIO_BIT_WIDTH_E：定义音频采样精度。

- AIO_MODE_E：定义音频输入输出工作模式。
- AUDIO_SOUND_MODE_E：定义音频声道模式。
- AIO_ATTR_S：定义音频输入输出设备属性结构体。
- AUDIO_FRAME_S：定义音频帧数据结构体。
- AEC_FRAME_S：定义回声抵消参考帧信息结构体。
- AUDIO_FRAME_INFO_S：定义音频帧信息结构体。
- AUDIO_STREAM_S：定义音频码流结构体。
- AMR_MODE_E：定义 AMR 编解码速率模式。
- AMR_FORMAT_E：定义 AMR 编解码格式。
- G726_BPS_E：定义 G.726 编解码协议速率。
- ADPCM_TYPE_E：定义 ADPCM 编解码协议类型。
- AAC_TYPE_E：定义 AAC 音频编解码协议类型。
- AAC_BPS_E：定义 AAC 音频编码码率。

AUDIO_SAMPLE_RATE_E

【说明】

定义音频采样率。

【定义】

```
typedef enum hiAUDIO_SAMPLE_RATE_E
{
    AUDIO_SAMPLE_RATE_8000  =8000,      /* 8kHz sampling rate */
    AUDIO_SAMPLE_RATE_11025 =11025,     /* 11.025kHz sampling rate */
    AUDIO_SAMPLE_RATE_16000 =16000,     /* 16kHz sampling rate */
    AUDIO_SAMPLE_RATE_22050 =22050,     /* 22.050kHz sampling rate */
    AUDIO_SAMPLE_RATE_24000 =24000,     /* 24kHz sampling rate */
    AUDIO_SAMPLE_RATE_32000 =32000,     /* 32kHz sampling rate */
    AUDIO_SAMPLE_RATE_44100 =44100,     /* 44.1kHz sampling rate */
    AUDIO_SAMPLE_RATE_48000 =48000,     /* 48kHz sampling rate */
}AUDIO_SAMPLE_RATE_E;
```

【成员】

成员名称	描述
AUDIO_SAMPLE_RATE_8000	8kHz 采样率。
AUDIO_SAMPLE_RATE_11025	11.025kHz 采样率。
AUDIO_SAMPLE_RATE_16000	16kHz 采样率。
AUDIO_SAMPLE_RATE_22050	22.050kHz 采样率。
AUDIO_SAMPLE_RATE_24000	24kHz 采样率。

成员名称	描述
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】

这里枚举值不是从 0 开始，而是与实际的采样率值相同。

【相关数据类型及接口】

AIO_ATTR_S

AUDIO_BIT_WIDTH_E

【说明】

定义音频采样精度。

【定义】

```
typedef enum hiAUDIO_BIT_WIDTH_E
{
    AUDIO_BIT_WIDTH_8   =0,     /* 8bit/sample */
    AUDIO_BIT_WIDTH_16  =1,     /* 16bit/sample */
    AUDIO_BIT_WIDTH_32  =2,     /* 32bit/sample */
    AUDIO_BIT_WIDTH_BUTT,
}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_16	采样精度为 16bit 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】

AIO_ATTR_S

AIO_MODE_E

【说明】

定义音频输入输出设备工作模式。

【定义】

```
typedef enum hiAIO_MODE_E
{
    AIO_MODE_I2S_MASTER = 0,        /* I2S master mode */
    AIO_MODE_I2S_SLAVE  = 1,        /* I2S slave mode */
    AIO_MODE_PCM_SLAVE_STD,         /* SIO PCM slave standard mode */
    AIO_MODE_PCM_SLAVE_NSTD,        /* SIO PCM slave non-standard mode */
    AIO_MODE_BUTT
}AIO_MODE_E;
```

【成员】

成员名称	描述
AIO_MODE_I2S_MASTER	I^2S 主模式。
AIO_MODE_I2S_SLAVE	I^2S 从模式。
AIO_MODE_PCM_SLAVE_STD	PCM 从模式（标准协议）
AIO_MODE_PCM_SLAVE_NSTD	PCM 从模式（非标准协议）

【注意事项】

目前只支持 I^2S 从模式。

【相关数据类型及接口】

AIO_ATTR_S

AUDIO_SOUND_MODE_E

【说明】

定义音频声道模式。

【定义】

```
typedef enum hiAIO_SOUND_MODE_E
{
    AUDIO_SOUND_MODE_MOMO   =0,     /*momo*/
    AUDIO_SOUND_MODE_STEREO =1,     /*stereo*/
    AUDIO_SOUND_MODE_BUTT
}AUDIO_SOUND_MODE_E;
```

【成员】

成员名称	描述
AUDIO_SOUND_MODE_MOMO	单声道。
AUDIO_SOUND_MODE_STEREO	双声道。

【注意事项】

无。

【相关数据类型及接口】

AIO_ATTR_S

AIO_ATTR_S

【说明】

定义音频输入输出设备属性结构体。

【定义】

```
typedef struct hiAIO_ATTR_S
{
    AUDIO_SAMPLE_RATE_E enSamplerate;       /*sample rate*/
    AUDIO_BIT_WIDTH_E   enBitwidth;         /*bitwidth*/
    AIO_MODE_E          enWorkmode;         /*master or slave mode*/
    AUDIO_SOUND_MODE_E  enSoundmode;        /*momo or steror*/
    HI_U32              u32EXFlag;          /*expand 8bit to 16bit */
    HI_U32              u32FrmNum;          /*frame num in buffer*/
    HI_U32              u32PtNumPerFrm;     /*number of samples*/
}AIO_ATTR_S;
```

【成员】

成员名称	描述
enSamplerate	音频采样率（从模式下，此参数不起作用）。 静态属性。
enBitwidth	音频采样精度（从模式下，此参数必须和音频 AD/DA 的采样精度匹配）。 静态属性。
enWorkmode	音频输入输出工作模式（当前版本只支持从模式）。 静态属性。

成员名称	描述
enSoundmode	音频声道模式。 静态属性。
u32EXFlag	8bit 到 16bit 扩展标志（8bit 精度时有效）。 取值范围：{0, 1}。 • 0：不扩展。 • 1：扩展。 静态属性。
u32FrmNum	缓存帧数目。 取值范围：[1, 10]。 静态属性。
u32PtNumPerFrm	每帧的采样点个数。 取值范围：G711、G726、ADPCM_DVI4 编码时取值为 80、160、240、320、480；ADPCM_IMA 编码时取值为 81、161、241、321、481；AMR 编码时只支持 160。 静态属性。

【注意事项】

无。

【相关数据类型及接口】

HI_MPI_AI_SetPubAttr

AUDIO_FRAME_S

【说明】

定义音频帧结构体。

【定义】

```
typedef struct hiAUDIO_FRAME_S
{
    AUDIO_BIT_WIDTH_E   enBitwidth;  /*audio frame bitwidth*/
    AUDIO_SOUND_MODE_E  enSoundmode; /*audio frame momo or stereo mode*/
    HI_U8               aData[MAX_AUDIO_FRAME_LEN*2];
    HI_U64              u64TimeStamp;/*audio frame timestamp*/
    HI_U32              u32Seq;      /*audio frame seq*/
    HI_U32              u32Len;      /*data lenth per channel in frame*/
}AUDIO_FRAME_S;
```

【成员】

成员名称	描述
enBitwidth	音频采样精度。
enSoundmode	音频声道模式。
aData[MAX_AUDIO_FRAME_LEN*2]	实际音频帧数据。
u64TimeStamp	音频帧时间戳。以 μs 为单位。
u32Seq	音频帧序号。
u32Len	音频帧长度。以 byte 为单位。

【注意事项】

- u32Len（音频帧长度）指单个声道的数据长度。
- 单声道数据直接存放，采样点数为 ptnum，长度为 len；立体声数据按左右声道分开存放，先存放采样点为 ptnum、长度为 len 的左声道数据，然后存放采样点为 ptnum，长度为 len 的右声道数据。

【相关数据类型及接口】

- HI_MPI_AENC_SendFrame
- HI_MPI_AO_SendFrame

AEC_FRAME_S

【说明】

定义音频回声抵消参考帧信息结构体。

【定义】

```
typedef struct hiAEC_FRAME_S
{
    AUDIO_FRAME_S   stRefFrame;     /* aec reference audio frame */
    HI_BOOL         bValid;         /* whether frame is valid */
}AEC_FRAME_S;
```

【成员】

成员名称	描述
stRefFrame	回声抵消参考帧结构体。
bValid	参考帧有效的标志。 取值范围： HI_TRUE：参考帧有效。 HI_FALSE：参考帧无效，无效时不能使用此参考帧进行回声抵消。

【注意事项】

无。

【相关数据类型及接口】

HI_MPI_AENC_SendFrame

AUDIO_FRAME_INFO_S

【说明】

定义音频帧信息结构体。

【定义】

```
typedef struct hiAUDIO_FRAME_INFO_S
{
    AUDIO_FRAME_S   *pstFrame;  /*frame ptr*/
    HI_U32          u32Id;      /*frame id*/
}AUDIO_FRAME_INFO_S;
```

【成员】

成员名称	描述
pstFrame	音频帧数据结构体指针。
u32Id	音频帧序号。 内部保留字，不能更改。

【注意事项】

无。

【相关数据类型及接口】

- HI_MPI_ADEC_GetData
- HI_MPI_ADEC_ReleaseData

AUDIO_STREAM_S

【说明】

定义音频码流结构体。

【定义】

```
typedef struct hiAUDIO_STREAM_S
{
    HI_U8   *pStream;       /*stream buffer */
```

```
    HI_U32  u32Len;         /*stream lenth*/
    HI_U64  u64TimeStamp;   /*frame time stamp*/
    HI_U32  u32Seq;         /*frame seq,if stream is not a valid frame,
                            u32Seq is 0*/
}AUDIO_STREAM_S;
```

【成员】

成员名称	描述
pStream	音频码流数据指针。
u32Len	音频码流长度。以 byte 为单位。
u64TimeStamp	音频码流时间戳。
u32Seq	音频码流序号。

【注意事项】

无。

【相关数据类型及接口】

HI_MPI_AENC_GetStream

AMR_MODE_E

【说明】

定义 AMR 编解码速率模式。

【定义】

```
typedef enum hiAMR_MODE_E
{
    AMR_MODE_MR475 = 0,
    AMR_MODE_MR515,
    AMR_MODE_MR59,
    AMR_MODE_MR67,
    AMR_MODE_MR74,
    AMR_MODE_MR795,
    AMR_MODE_MR102,
    AMR_MODE_MR122,
    AMR_MODE_MRDTX,
    AMR_MODE_N_MODES,
    AMR_MODE_BUTT
}AMR_MODE_E;
```

【成员】

成员名称	描述
AMR_MODE_MR475	可选速率，4.75 kbit/s。
AMR_MODE_MR515	可选速率，5.15 kbit/s。
AMR_MODE_MR59	可选速率，5.9 kbit/s。
AMR_MODE_MR67	可选速率，6.7 kbit/s。
AMR_MODE_MR74	可选速率，7.4 kbit/s。
AMR_MODE_MR795	可选速率，7.95 kbit/s。
AMR_MODE_MR102	可选速率，10.2 kbit/s。
AMR_MODE_MR122	可选速率，12.2 kbit/s。
AMR_MODE_MRDTX	静音模式（内部模式，不可选）。
AMR_MODE_N_MODES	基本速率数目（保留）。

【注意事项】

无。

【相关数据类型及接口】

略。

AMR_FORMAT_E

【说明】

定义 AMR 编解码格式。

【定义】

```
typedef enum hiAMR_FORMAT_E
{
    AMR_FORMAT_MMS,
    AMR_FORMAT_IF1,
    AMR_FORMAT_IF2,
    AMR_FORMAT_BUTT
}AMR_FORMAT_E;
```

【成员】

成员名称	描述
AMR_FORMAT_MMS	MMS 格式（推荐采用）。
AMR_FORMAT_IF1	IF1 格式。

成员名称	描述
AMR_FORMAT_IF2	IF2 格式。

【注意事项】

无。

【相关数据类型及接口】

略。

G726_BPS_E

【说明】

定义 G.726 编解码协议速率。

【定义】

```
typedef enum hiG726_BPS_E
{
    G726_16K = 0,
    G726_24K,
    G726_32K,
    G726_40K,
    MEDIA_G726_16K,     /* G726 16kbit/s for ASF */
    MEDIA_G726_24K,     /* G726 24kbit/s for ASF */
    MEDIA_G726_32K,     /* G726 32kbit/s for ASF */
    MEDIA_G726_40K,     /* G726 40kbit/s for ASF */
    G726_BUTT,
}G726_BPS_E;
```

【成员】

成员名称	描述
G726_16K	16kbit/s G.726，请参见“RFC3551 文档 4.5.4 G72616”。
G726_24K	24kbit/s G.726，请参见“RFC3551 文档 4.5.4 G72624”。
G726_32K	32kbit/s G.726，请参见“RFC3551 文档 4.5.4 G72632”。
G726_40K	40kbit/s G.726，请参见“RFC3551 文档 4.5.4 G72640”。
MEDIA_G726_16K	G726 16kbit/s for ASF。
MEDIA_G726_24K	G726 24kbit/s for ASF。
MEDIA_G726_32K	G726 32kbit/s for ASF。
MEDIA_G726_40K	G726 40kbit/s for ASF。

【注意事项】

无。

【相关数据类型及接口】

略。

ADPCM_TYPE_E

【说明】

定义 ADPCM 编解码协议类型。

【定义】

```
typedef enum hiADPCM_TYPE_E
{
    ADPCM_TYPE_DVI4 = 0,
    ADPCM_TYPE_IMA,
    ADPCM_TYPE_BUTT,
}ADPCM_TYPE_E;
```

【成员】

成员名称	描述
ADPCM_TYPE_DVI4	32kbit/s ADPCM(DVI4)。
ADPCM_TYPE_IMA	32kbit/s ADPCM(IMA)。

【注意事项】

无。

【相关数据类型及接口】

略。

AAC_TYPE_E

【说明】

定义 AAC 音频编解码协议类型。

【定义】

```
typedef enum hiAAC_TYPE_E
{
    AAC_TYPE_AACLC      = 0,
    AAC_TYPE_EAAC       = 1,
```

```
    AAC_TYPE_EAACPLUS   = 2,
    AAC_TYPE_BUTT,
}AAC_TYPE_E;
```

【成员】

成员名称	描述
AAC_TYPE_AACLC	AACLC 格式。
AAC_TYPE_EAAC	eAAC 格式（也称为 HEAAC、AAC+或 aacPlusV1）。
AAC_TYPE_EAACPLUS	eAACPLUS 格式（也称为 AAC++或 aacPlusV2）。

【注意事项】

无。

【相关数据类型及接口】

略。

AAC_BPS_E

【说明】

定义 AAC 音频编码码率。

【定义】

```
typedef enum hiAAC_BPS_E
{
    AAC_BPS_16K = 16000,
    AAC_BPS_22K = 22000,
    AAC_BPS_24K = 24000,
    AAC_BPS_32K = 32000,
    AAC_BPS_48K = 48000,
    AAC_BPS_64K = 64000,
    AAC_BPS_96K = 96000,
    AAC_BPS_128K= 128000,
    AAC_BPS_BUTT
}AAC_BPS_E;
```

【成员】

成员名称	描述
AAC_BPS_16K	16kbit/s。
AAC_BPS_22K	22kbit/s。

成员名称	描述
AAC_BPS_24K	24kbit/s。
AAC_BPS_32K	32kbit/s。
AAC_BPS_48K	48kbit/s。
AAC_BPS_64K	64kbit/s。
AAC_BPS_96K	96kbit/s。
AAC_BPS_128K	128kbit/s。

【注意事项】

无。

【相关数据类型及接口】

略。

3.5.2 音频编码

音频编码相关数据类型、数据结构定义如下：

- AENC_ATTR_AMR_S：定义 AMR 编码协议属性结构体。
- AENC_ATTR_G711_S：定义 G.711 编码协议属性结构体。
- AENC_ATTR_G726_S：定义 G.726 编码协议属性结构体。
- AENC_ATTR_ADPCM_S：定义 ADPCM 编码协议属性结构体。
- AENC_ATTR_AAC_S：定义 AAC 编码协议属性结构体。
- AENC_CHN_ATTR_S：定义音频编码通道属性结构体。

AENC_ATTR_AMR_S

【说明】

定义 AMR 编码协议属性结构体。

【定义】

```
typedef struct hiAENC_ATTR_AMR_S
{
    AMR_MODE_E      enMode;
    AMR_FORMAT_E    enFormat;
    HI_S32          s32Dtx;
}AENC_ATTR_AMR_S;
```

【成员】

成员名称	描述
enMode	AMR 编码模式。
enFormat	AMR 编码格式。
s32Dtx	Dtx 启用标志。 取值范围：{0, 1} 0：不启用。 1：启用。

【注意事项】

无。

【相关数据类型及接口】

略。

AENC_ATTR_G711_S

【说明】

定义 G.711 编码协议属性结构体。

【定义】

```
typedef struct hiAENC_ATTR_G711_S
{
    HI_U32 resv;
}AENC_ATTR_G711_S;
```

【成员】

成员名称	描述
resv	待扩展用（目前暂未使用）。

【注意事项】

无。

【相关数据类型及接口】

略。

AENC_ATTR_G726_S

【说明】

定义 G.726 编码协议属性结构体。

【定义】

```
typedef struct hiAENC_ATTR_G726_S
{
    G726_BPS_E  enG726bps;
}AENC_ATTR_G726_S;
```

【成员】

成员名称	描述
enG726bps	G.726 协议码率。

【注意事项】

无。

【相关数据类型及接口】

G726_BPS_E

AENC_ATTR_ADPCM_S

【说明】

定义 ADPCM 编码协议属性结构体。

【定义】

```
typedef struct hiAENC_ATTR_ADPCM_S
{
    ADPCM_TYPE_E    enADPCMType;
}AENC_ATTR_ADPCM_S;
```

【成员】

成员名称	描述
enADPCMType	ADPCM 类型。

【注意事项】

无。

【相关数据类型及接口】

ADPCM_TYPE_E

AENC_ATTR_AAC_S

【说明】

定义 AAC 编码协议属性结构体。

【定义】

```
typedef struct hiAENC_ATTR_AAC_S
{
    AAC_TYPE_E          enAACType;
    AAC_BPS_E           enBitRate;
    AUDIO_SAMPLE_RATE_E enSmpRate;
    AUDIO_BIT_WIDTH_E   enBitWidth;
    AUDIO_SOUND_MODE_E  enSoundMode;
}AENC_ATTR_AAC_S;;
```

【成员】

成员名称	描述
enAACType	AAC 编码类型（Profile）。
enBitRate	编码码率。 取值范围： LC：48～128； EAAC：22～48； EAAC+：16～32； 以 kbit/s 为单位。
enSmpRate	音频数据的采样率。 取值范围： LC：16～48； EAAC：32～48； EAAC+：32～48。 以 kHz 为单位。
enBitWidth	音频数据采样精度，只支持 16bit。
enSoundMode	输入数据的声道模式。支持输入为单声道或双声道。

【注意事项】

无。

【相关数据类型及接口】

略。

AENC_CHN_ATTR_S

【说明】

定义音频编码通道属性结构体。

【定义】

```
typedef struct hiAENC_CHN_ATTR_S
{
    PAYLOAD_TYPE_E  enType;
    HI_U32  u32BufSize; /*buffer size，以帧为单位，[1～MAX_AUDIO_FRAME_NUM]*/
    HI_VOID *pValue;
}AENC_CHN_ATTR_S;
```

【成员】

成员名称	描述
enType	音频编码协议类型。 静态属性。
u32BufSize	音频编码缓存大小。 取值范围：[1, MAX_AUDIO_FRAME_NUM]，以帧为单位。 静态属性。
pValue	具体协议属性指针。 静态属性。

【注意事项】

无。

【相关数据类型及接口】

AENC_ATTR_AMR_S

3.5.3 音频解码

音频解码相关数据类型、数据结构定义如下：

- ADEC_ATTR_AMR_S：定义 AMR 解码协议属性结构体。
- ADEC_ATTR_G711_S：定义 G.711 解码协议属性结构体。
- ADEC_ATTR_G726_S：定义 G.726 解码协议属性结构体。
- ADEC_ATTR_ADPCM_S：定义 ADPCM 编码协议属性结构体。
- ADEC_ATTR_AAC_S：定义 AAC 解码协议属性结构体。
- ADEC_MODE_E：定义解码方式。
- ADEC_CHN_ATTR_S：定义解码通道属性结构体。

ADEC_ATTR_AMR_S

【说明】

定义 AMR 解码协议属性结构体。

【定义】

```
typedef struct hiADEC_ATTR_AMR_S
{
    AMR_FORMAT_E    enFormat;
}ADEC_ATTR_AMR_S;
```

【成员】

成员名称	描述
enFormat	AMR 编码格式。

【注意事项】

无。

【相关数据类型及接口】

略。

ADEC_ATTR_G711_S

【说明】

定义 G.711 解码协议属性结构体。

【定义】

```
typedef struct hiADEC_ATTR_G711_S
{
    HI_U32  resv;
}ADEC_ATTR_G711_S;
```

【成员】

成员名称	描述
resv	待扩展用（目前暂未使用）。

【注意事项】

无。

【相关数据类型及接口】

略。

ADEC_ATTR_G726_S

【说明】

定义 G.726 解码协议属性结构体。

【定义】

```
typedef struct hiADEC_ATTR_G726_S
{
    G726_BPS_E  enG726bps;
}ADEC_ATTR_G726_S;
```

【成员】

成员名称	描述
enG726bps	G.726 协议码率。

【注意事项】

无。

【相关数据类型及接口】

G726_BPS_E

ADEC_ATTR_ADPCM_S

【说明】

定义 ADPCM 编码协议属性结构体。

【定义】

```
typedef struct hiADEC_ATTR_ADPCM_S
{
    ADPCM_TYPE_E    enADPCMType;
}ADEC_ATTR_ADPCM_S;
```

【成员】

成员名称	描述
enADPCMType	ADPCM 类型。

【注意事项】

无。

【相关数据类型及接口】

ADPCM_TYPE_E

ADEC_ATTR_AAC_S

【说明】

定义 AAC 解码协议属性结构体。

【定义】

```
typedef struct hiADEC_ATTR_AAC_S
{
    HI_U32 resv;
}ADEC_ATTR_AAC_S;
```

【成员】

成员名称	描述
resv	待扩展用（目前暂未使用）。

【注意事项】

无。

【相关数据类型及接口】

略。

ADEC_MODE_E

【说明】

定义解码方式。

【定义】

```
typedef enum hiADEC_MODE_E
{
    ADEC_MODE_PACK = 0,/*require input is valid dec pack(a
                        complete frame encode result),
                        e.g.the stream get from AENC is a
                        valid dec pack, the stream know actually
                        pack len from file is also a dec pack.
                        this mode is high-performative*/
    ADEC_MODE_STREAM ,/*input is stream,low-performative,
                        if you couldn't find out whether a stream is
                        vaild dec pack,you could use
                        this mode*/
    ADEC_MODE_BUTT
```

```
}ADEC_MODE_E;
```

【成员】

成员名称	描述
ADEC_MODE_PACK	pack 模式解码。
ADEC_MODE_STREAM	stream 模式解码。

【注意事项】

- pack 模式用于用户确认当前码流包为一帧数据编码结果的情况下，解码器会直接进行对其解码，如果不是一帧，解码器会出错。这种模式的效率比较高，在使用 AENC 模块编码的码流包如果没有破坏，均可以使用此方式解码。
- stream 模式用于用户不能确认当前码流包是不是一帧数据的情况下，解码器需要对码流进行判断并缓存，此工作方式的效率低下，一般用于读文件码流送解码或者不确定码流包边界的情况。当然由于语音编码码流长度固定，很容易确定在码流中的帧边界，推荐使用 pack 模式解码。

【相关数据类型及接口】

略。

ADEC_CHN_ATTR_S

【说明】

定义解码通道属性结构体。

【定义】

```
typedef struct hiADEC_CH_ATTR_S
{
    PAYLOAD_TYPE_E  enType;
    HI_U32          u32BufSize;
    ADEC_MODE_E     enMode;
    HI_VOID         *pValue;
}ADEC_CHN_ATTR_S;
```

【成员】

成员名称	描述
enType	音频解码协议类型。 静态属性。
u32BufSize	音频解码缓存大小。 取值范围：[1, MAX_AUDIO_FRAME_NUM]，以帧为单位。 静态属性。

成员名称	描述
enMode	解码方式。 静态属性。
pValue	具体协议属性指针。

【注意事项】

无。

【相关数据类型及接口】

略。

3.6 音视频复合编码数据类型

音视频复合编码相关数据类型、数据结构定义如下：

- AVENC_LIST_S：定义链表节点结构体。
- AVENC_STREAM_S：定义音视频复合通道码流帧结构体。
- AVENC_OPERATE_OBJECT_E：定义音视频复合编码操作对象类型。

AVENC_LIST_S

【说明】

定义链表节点结构体。

【定义】

```
typedef struct hiAVENC_LIST_S
{
    HI_VOID *pData;
    Struct  hiAVENC_LIST_S  *pNext;
} AVENC_LIST_S;
```

【成员】

成员名称	描述
pData	数据指针。
pNext	链表下一个指针。

【注意事项】

无。

【相关数据类型及接口】

略。

AVENC_STREAM_S

【说明】

定义音视频复合通道码流帧结构体。

【定义】

```
typedef struct hiAVENC_STREAM_S
{
    AVENC_LIST_S    *pAudioListHead;
    AVENC_LIST_S    *pVideoListHead;
    HI_U32          u32Seq;
} AVENC_LIST_S;
```

【成员】

成员名称	描述
pAudioListHead	音频队列首部。
pVideoListHead	视频队列首部。
u32Seq	复合编码码流序列标示。

【注意事项】

无。

【相关数据类型及接口】

略。

AVENC_OPERATE_OBJECT_E

【说明】

定义音视频复合编码操作对象类型。

【定义】

```
typedef enum hiAVENC_OPERATE_OBJECT_E
{
    AVENC_OPERATION_BOTH  = 0,
    AVENC_OPERATION_AUDIO = 1,
    AVENC_OPERATION_VIDEO = 2,
    AVENC_OPERATION_BUTT
} AVENC_LIST_S;
```

【成员】

成员名称	描述
AVENC_OPERATION_BOTH	音频和视频都编码。
AVENC_OPERATION_AUDIO	音频编码。
AVENC_OPERATION_VIDEO	视频编码。

【注意事项】

无。

【相关数据类型及接口】

略。

4 错误码

4.1 概述

Hi3507 媒体处理软件开发包的 MPI 错误码分为系统控制类错误码、视频功能类错误码和音频功能类错误码。当错误码的值在-4096～0 之间时，为 Linux 操作系统的错误码，可以使用 perror 来查看具体错误原因。

4.2 系统控制错误码

错误代码	宏定义	描述
0xA0028003	HI_ERR_SYS_ILLEGAL_PARAM	参数设置无效
0xA0028006	HI_ERR_SYS_NULL_PTR	空指针错误
0xA0028009	HI_ERR_SYS_NOT_PERM	操作不允许
0xA0028010	HI_ERR_SYS_NOTREADY	系统控制属性未配置
0xA0028012	HI_ERR_SYS_BUSY	系统忙

4.3 视频功能类错误码

视频缓存池错误码

错误代码	宏定义	描述
0xA0018003	HI_ERR_VB_ILLEGAL_PARAM	参数设置无效
0xA0018005	HI_ERR_VB_UNEXIST	视频缓存池不存在
0xA0018006	HI_ERR_VB_NULL_PTR	参数空指针错误

错误代码	宏定义	描述
0xA0018009	HI_ERR_VB_NOT_PERM	操作不允许
0xA001800c	HI_ERR_VB_NOMEM	分配内存失败
0xA001800d	HI_ERR_VB_NOBUF	分配缓存失败
0xA0018010	HI_ERR_VB_NOTREADY	系统控制属性未配置
0xA0018012	HI_ERR_VB_BUSY	系统忙
0xA0018040	HI_ERR_VB_2MPOOLS	创建缓存池多

视频输入错误码

错误代码	宏定义	描述
0xA0108001	HI_ERR_VI_INVALID_DEVID	视频输入设备号无效
0xA0108002	HI_ERR_VI_INVALID_CHNID	视频输入通道号无效
0xA0108003	HI_ERR_VI_INVALID_PARA	视频输入参数设置无效
0xA0108006	HI_ERR_VI_INVALID_NULL_PTR	输入参数空指针错误
0xA0108007	HI_ERR_VI_FAILED_NOTCONFIG	视频输入通道属性未配置
0xA0108008	HI_ERR_VI_NOT_SUPPORT	操作不支持
0xA0108009	HI_ERR_VI_NOT_PERM	操作不允许
0xA010800c	HI_ERR_VI_NOMEM	分配内存失败
0xA010800E	HI_ERR_VI_BUF_EMPTY	视频输入缓存为空
0xA010800F	HI_ERR_VI_BUF_FULL	视频输入缓存为满
0xA0108010	HI_ERR_VI_SYS_NOTREADY	视频输入系统未初始化
0xA0108012	HI_ERR_VI_BUSY	视频输入系统忙
0xA0108040	HI_ERR_VI_FAILED_NOTENABLE	视频输入设备或通道未启用
0xA0108041	HI_ERR_VI_FAILED_NOTDISABLE	视频输入设备未禁用
0xA0108042	HI_ERR_VI_FAILED_CHNOTDISABLE	视频输入通道未禁用

视频输出错误码

错误代码	宏定义	描述
0xA00F8002	HI_ERR_VO_INVALID_CHNID	通道 ID 超出合法范围
0xA00F8003	HI_ERR_VO_ILLEGAL_PARAM	参数超出合法范围
0xA00F8006	HI_ERR_VO_NULL_PTR	函数参数中有空指针
0xA00F8010	HI_ERR_VO_SYS_NOTREADY	系统未初始化
0xA00F800C	HI_ERR_VO_NOMEM	内存不足
0xA00F8040	HI_ERR_VO_FAILED_NOTENABLE	未使能视频输出设备
0xA00F8041	HI_ERR_VO_FAILED_NOTCONFIG	未配置视频输出设备
0xA00F8042	HI_ERR_VO_FAILED_NOTDISABLE	未禁用视频输出设备
0xA00F8043	HI_ERR_VO_FAILED_CHNOTDISABLE	有通道未禁用
0xA00F8044	HI_ERR_VO_FAILED_TIMEOUT	通道操作超时
0xA00F8045	HI_ERR_VO_FAILED_CHNOTENABLE	有通道未使能
0xA00F8046	HI_ERR_VO_FAILED_CHNNOTCONFIG	通道未配置属性
0xA00F8012	HI_ERR_VO_BUSY	系统忙
0xA00F8047	HI_ERR_VO_GRP_INVALID_NUMBER	同步组号无效
0xA00F8048	HI_ERR_VO_GRP_NOT_CREATE	同步组没有创建
0xA00F8049	HI_ERR_VO_GRP_HAS_CREATED	同步组已经存在
0xA00F804A	HI_ERR_VO_GRP_CHN_FULL	同步组已经注册满
0xA00F804B	HI_ERR_VO_GRP_CHN_EMPTY	同步组为空
0xA00F804C	HI_ERR_VO_GRP_CHN_NOT_EMPTY	同步组通道未空
0xA00F804D	HI_ERR_VO_GRP_INVALID_SYN_MODE	同步组模式无效
0xA00F804E	HI_ERR_VO_GRP_INVALID_BASE_PTS	同步组基准时间戳无效
0xA00F804F	HI_ERR_VO_GRP_NOT_START	同步组没有启动
0xA00F8050	HI_ERR_VO_GRP_INVALID_FRMRATE	同步组帧率无效

错误代码	宏定义	描述
0xA00F8051	HI_ERR_VO_GRP_CHN_NOT_UNREG	同步组通道未注销
0xA00F8053	HI_ERR_VO_INVALID_RECT_PARA	放大窗矩形区参数错误
0xA00F8054	HI_ERR_VO_GRP_CHN_HAS_REG	同步组中通道已经注册
0xA00F8055	HI_ERR_VO_GRP_CHN_NOT_REG	同步组中通道未注册
0xA00F8056	HI_ERR_VO_SETBEGIN_ALREADY	设置属性已经开始
0xA00F8057	HI_ERR_VO_SETBEGIN_NOTYET	设置属性还未开始
0xA00F8058	HI_ERR_VO_SETEND_ALREADY	设置属性已经结束
0xA00F8059	HI_ERR_VO_SETEND_NOTYET	设置属性还未结束

视频前处理错误码

错误代码	宏定义	描述
0xA0078001	HI_ERR_VPP_INVALID_DEVID	设备 ID 超出合法范围
0xA0078002	HI_ERR_VPP_INVALID_CHNID	通道 ID 超出合法范围
0xA0078003	HI_ERR_VPP_ILLEGAL_PARAM	参数超出合法范围
0xA0078004	HI_ERR_VPP_EXIST	重复创建已存在的设备、通道或资源
0xA0078005	HI_ERR_VPP_UNEXIST	试图使用或者销毁不存在的设备、通道或者资源
0xA0078006	HI_ERR_VPP_NULL_PTR	函数参数中有空指针
0xA0078007	HI_ERR_VPP_NOT_CONFIG	模块没有配置
0xA0078008	HI_ERR_VPP_NOT_SUPPORT	不支持的参数或者功能
0xA0078009	HI_ERR_VPP_NOT_PERM	该操作不允许，如试图修改静态配置参数
0xA007800C	HI_ERR_VPP_NOMEM	分配内存失败，如系统内存不足
0xA007800D	HI_ERR_VPP_NOBUF	分配缓存失败，如申请的数据缓冲区太大
0xA007800E	HI_ERR_VPP_BUF_EMPTY	缓冲区中无数据
0xA007800F	HI_ERR_VPP_BUF_FULL	缓冲区中数据满

错误代码	宏定义	描述
0xA0078010	HI_ERR_VPP_NOTREADY	系统没有初始化或没有加载相应模块
0xA0078011	HI_ERR_VPP_BADADDR	地址非法
0xA0078012	HI_ERR_VPP_BUSY	系统忙

视频编码错误码

错误代码	宏定义	描述
0xA0068001	HI_ERR_VENC_INVALID_DEVID	设备 ID 超出合法范围
0xA0068002	HI_ERR_VENC_INVALID_CHNID	通道 ID 超出合法范围
0xA0068003	HI_ERR_VENC_ILLEGAL_PARAM	参数超出合法范围
0xA0068004	HI_ERR_VENC_EXIST	试图申请或者创建已经存在的设备、通道或者资源
0xA0068005	HI_ERR_VENC_UNEXIST	试图使用或者销毁不存在的设备、通道或者资源
0xA0068006	HI_ERR_VENC_NULL_PTR	函数参数中有空指针
0xA0068007	HI_ERR_VENC_NOT_CONFIG	使用前未配置
0xA0068008	HI_ERR_VENC_NOT_SUPPORT	不支持的参数或者功能
0xA0068009	HI_ERR_VENC_NOT_PERM	该操作不允许，如试图修改静态配置参数
0xA006800C	HI_ERR_VENC_NOMEM	分配内存失败，如系统内存不足
0xA006800D	HI_ERR_VENC_NOBUF	分配缓存失败，如申请的数据缓冲区太大
0xA006800E	HI_ERR_VENC_BUF_EMPTY	缓冲区中无数据
0xA006800F	HI_ERR_VENC_BUF_FULL	缓冲区中数据满
0xA0068010	HI_ERR_VENC_SYS_NOTREADY	系统没有初始化或没有加载相应模块

移动侦测错误码

错误代码	宏定义	描述
0xA0088001	HI_ERR_MD_INVALID_DEVID	设备 ID 超出合法范围
0xA0088002	HI_ERR_MD_INVALID_CHNID	通道 ID 超出合法范围
0xA0088003	HI_ERR_MD_ILLEGAL_PARAM	参数超出合法范围
0xA0088004	HI_ERR_MD_EXIST	试图申请或者创建已经存在的设备、通道或者资源
0xA0088005	HI_ERR_MD_UNEXIST	试图使用或者销毁不存在的设备、通道或者资源
0xA0088006	HI_ERR_MD_NULL_PTR	参数中有空指针
0xA0088007	HI_ERR_MD_NOT_CONFIG	使用前未配置
0xA0088008	HI_ERR_MD_NOT_SUPPORT	不支持的参数或者功能
0xA0088009	HI_ERR_MD_NOT_PERM	该操作不允许，如试图修改静态配置参数
0xA008800C	HI_ERR_MD_NOMEM	分配内存失败，如系统内存不足
0xA008800D	HI_ERR_MD_NOBUF	分配缓存失败，如申请的数据缓冲区太大
0xA008800E	HI_ERR_MD_BUF_EMPTY	缓冲区中无数据
0xA008800F	HI_ERR_MD_BUF_FULL	缓冲区中数据满
0xA0088010	HI_ERR_MD_SYS_NOTREADY	系统没有初始化或没有加载相应模块
0xA0088012	HI_ERR_MD_BUSY	系统忙

视频解码错误码

错误代码	宏定义	描述
0xA00C8001	HI_ERR_VDEC_INVALID_DEVID	设备 ID 超出合法范围
0xA00C8002	HI_ERR_VDEC_INVALID_CHNID	通道 ID 超出合法范围
0xA00C8003	HI_ERR_VDEC_ILLEGAL_PARAM	参数超出合法范围
0xA00C8004	HI_ERR_VDEC_EXIST	试图申请或者创建已经存在的设备、通道或者资源
0xA00C8006	HI_ERR_VDEC_NULL_PTR	函数参数中有空指针

错误代码	宏定义	描述
0xA00C8007	HI_ERR_VDEC_NOT_CONFIG	使用前未配置
0xA00C8008	HI_ERR_VDEC_NOT_SUPPORT	不支持的参数或者功能
0xA00C8009	HI_ERR_VDEC_NOT_PERM	该操作不允许，如试图修改静态配置参数
0xA00C8005	HI_ERR_VDEC_UNEXIST	试图使用或者销毁不存在的设备、通道或者资源
0xA00C800C	HI_ERR_VDEC_NOMEM	分配内存失败，如系统内存不足
0xA00C800D	HI_ERR_VDEC_NOBUF	分配缓存失败，如申请的数据缓冲区太大
0xA00C800E	HI_ERR_VDEC_BUF_EMPTY	缓冲区中无数据
0xA00C800F	HI_ERR_VDEC_BUF_FULL	缓冲区中数据满
0xA00C8010	HI_ERR_VDEC_SYS_NOTREADY	系统未初始化
0xA00C8012	HI_ERR_VDEC_BUSY	系统忙

4.4 音频功能类错误码

音频输入错误码

错误代码	宏定义	描述
0xA0158001	HI_ERR_AI_INVALID_DEVID	音频输入设备号无效
0xA0158002	HI_ERR_AI_INVALID_CHNID	音频输入通道号无效
0xA0158003	HI_ERR_AI_ILLEGAL_PARAM	音频输入参数设置无效
0xA0158006	HI_ERR_AI_NULL_PTR	输入参数空指针错误
0xA0158007	HI_ERR_AI_NOT_CONFIG	音频输入设备属性未设置
0xA0158008	HI_ERR_AI_NOT_SUPPORT	操作不被支持
0xA0158009	HI_ERR_AI_NOT_PERM	操作不允许
0xA015800B	HI_ERR_AI_NOT_ENABLED	音频输入设备或通道未启用
0xA015800C	HI_ERR_AI_NOMEM	分配内存失败
0xA015800D	HI_ERR_AI_NOBUF	音频输入缓存不足

错误代码	宏定义	描述
0xA015800E	HI_ERR_AI_BUF_EMPTY	音频输入缓存为空
0xA015800F	HI_ERR_AI_BUF_FULL	音频输入缓存为满
0xA0158010	HI_ERR_AI_SYS_NOTREADY	音频输入系统未初始化
0xA0158012	HI_ERR_AI_BUSY	音频输入系统忙

音频输出错误码

错误代码	宏定义	描述
0xA0168001	HI_ERR_AO_INVALID_DEVID	音频输出设备号无效
0xA0168002	HI_ERR_AO_INVALID_CHNID	音频输出通道号无效
0xA0168003	HI_ERR_AO_ILLEGAL_PARAM	音频输出参数设置无效
0xA0168006	HI_ERR_AO_NULL_PTR	输出空指针错误
0xA0168007	HI_ERR_AO_NOT_CONFIG	音频输出设备属性未设置
0xA0168008	HI_ERR_AO_NOT_SUPPORT	操作不被支持
0xA0168009	HI_ERR_AO_NOT_PERM	操作不允许
0xA016800B	HI_ERR_AO_NOT_ENABLED	音频输出设备未启用
0xA016800C	HI_ERR_AO_NOMEM	系统内存不足
0xA016800D	HI_ERR_AO_NOBUF	音频输出缓存不足
0xA016800E	HI_ERR_AO_BUF_EMPTY	音频输出缓存为空
0xA016800F	HI_ERR_AO_BUF_FULL	音频输出缓存为满
0xA0168010	HI_ERR_AO_SYS_NOTREADY	音频输出系统未初始化
0xA0168012	HI_ERR_AO_BUSY	音频输出系统忙

音频编码错误码

错误代码	宏定义	描述
0xA0178001	HI_ERR_AENC_INVALID_DEVID	音频设备号无效
0xA0178002	HI_ERR_AENC_INVALID_CHNID	音频编码通道号无效
0xA0178003	HI_ERR_AENC_ILLEGAL_PARAM	音频编码参数设置无效

错误代码	宏定义	描述
0xA0178004	HI_ERR_AENC_EXIST	音频编码通道已经创建
0xA0178005	HI_ERR_AENC_UNEXIST	音频编码通道未创建
0xA0178006	HI_ERR_AENC_NULL_PTR	输入参数空指针错误
0xA0178007	HI_ERR_AENC_NOT_CONFIG	编码通道未配置
0xA0178008	HI_ERR_AENC_NOT_SUPPORT	操作不被支持
0xA0178009	HI_ERR_AENC_NOT_PERM	操作不允许
0xA017800C	HI_ERR_AENC_NOMEM	系统内存不足
0xA017800D	HI_ERR_AENC_NOBUF	编码通道缓存分配失败
0xA017800E	HI_ERR_AENC_BUF_EMPTY	编码通道缓存空
0xA017800F	HI_ERR_AENC_BUF_FULL	编码通道缓存满
0xA0178010	HI_ERR_AENC_SYS_NOTREADY	系统没有初始化
0xA0178040	HI_ERR_AENC_ENCODER_ERR	音频编码器内部错误

音频解码错误码

错误代码	宏定义	描述
0xA0188001	HI_ERR_ADEC_INVALID_DEVID	音频解码设备号无效
0xA0188002	HI_ERR_ADEC_INVALID_CHNID	音频解码通道号无效
0xA0188003	HI_ERR_ADEC_ILLEGAL_PARAM	音频解码参数设置无效
0xA0188004	HI_ERR_ADEC_EXIST	音频解码通道已经创建
0xA0188005	HI_ERR_ADEC_UNEXIST	音频解码通道未创建
0xA0188006	HI_ERR_ADEC_NULL_PTR	输入参数空指针错误
0xA0188007	HI_ERR_ADEC_NOT_CONFIG	解码通道属性未配置
0xA0188008	HI_ERR_ADEC_NOT_SUPPORT	操作不被支持
0xA0188009	HI_ERR_ADEC_NOT_PERM	操作不允许
0xA018800C	HI_ERR_ADEC_NOMEM	系统内存不足
0xA018800D	HI_ERR_ADEC_NOBUF	解码通道缓存分配失败
0xA018800E	HI_ERR_ADEC_BUF_EMPTY	解码通道缓存空
0xA018800F	HI_ERR_ADEC_BUF_FULL	解码通道缓存满

错误代码	宏定义	描述
0xA0188010	HI_ERR_ADEC_SYS_NOTREADY	系统没有初始化
0xA0188040	HI_ERR_ADEC_DECODER_ERR	音频解码器内部错误

音视频复合编码错误码

错误代码	宏定义	描述
0xA0198001	HI_ERR_AVENC_INVALID_CHN	视音频或复合编码通道号无效
0xA0198002	HI_ERR_AVENC_VCHN_USING	视频编码通道对应的复合通道已经存在
0xA0198003	HI_ERR_AVENC_ACHN_USING	音频编码通道对应的复合通道已经存在
0xA0198004	HI_ERR_AVENC_INVALID_PTR	输入指针无效
0xA0198005	HI_ERR_AVENC_CHN_NOT_CREATE	复合编码通道未创建
0xA0025006	HI_ERR_AVENC_CHN_NOT_STOP	复合编码通道有正在运行的编码任务
0xA0198007	HI_ERR_AVENC_CHN_RESTART	通道已经启动
0xA0198008	HI_ERR_AVENC_AUDIO_NOT_EXIST	音频通道不存在
0xA0198009	HI_ERR_AVENC_VIDEO_NOT_EXIST	视频通道不存在
0xA019800A	HI_ERR_AVENC_AUDIO_USEDBYOTHER	音频编码通道已经被单独启动过
0xA019800B	HI_ERR_AVENC_VIDEO_USEDBYOTHER	视频编码通道已经被单独启动过
0xA019800C	HI_ERR_AVENC_CHN_NOT_START	通道未启动
0xA019800D	HI_ERR_AVENC_BUFFER_NOTFREE	复合编码通道的缓冲区还在使用
0xA019800E	HI_ERR_AVENC_INVALID_OBJECT	操作对象无效
0xA019800F	HI_ERR_AVENC_NO_STREAM	复合流缓冲队列中没有数据
0xA0198010	HI_ERR_AVENC_AUDIONONBLOCK	音频通道是非阻塞方式

错误代码	宏定义	描述
0xA0198011	HI_ERR_AVENC_INVALID_WATER LEVEL	无效的水线
0xA0198012	HI_ERR_AVENC_TIMEOUT	获取数据超时
0xA0198013	HI_ERR_AVENC_INVALID_AUSIZE	无效的音频缓存块大小
0xA0198014	HI_ERR_AVENC_INVALID_PARA	传入的参数无效