# 矩阵上墙和融合屏相关SDK接口说明

1. 函数调用顺序
2. SDK初始化, 设备登陆等
3. 获取设备通道信息, 由于矩阵的通道号是非连续的, 所以不但要得到通道的通道数, 还要各个编码通道和解码通道的通道号. CLIENT\_QueryProductionDefinition可以得到设备的通道数, CLIENT\_QueryMatrixCardInfo得到通道具体信息.
4. 获取电视墙和融合屏配置. 融合屏可以看成是一种特殊的输出通道, 所以每个融合屏都可以用一个通道号标识. 融合屏的通道号是在原有输出通道号的基础上往上加1, 比如用CLIENT\_QueryProductionDefinition接口查询得到设备的输出通道数(nVideoOutChannel)是n, 那么融合屏的起始通道号就是n, 之后的各个融合屏的通道号分别是n+1, n+2, …
5. 分割控制, 电视墙配置等
6. 设备登出, SDK退出清理
7. 配置接口
8. BOOL CALL\_METHOD CLIENT\_GetNewDevConfig(LONG lLoginID, char\* szCommand, int nChannelID, char\* szOutBuffer, DWORD dwOutBufferSize, int \*error, int waittime);

函数说明：获取配置，按照字符串格式，各个字符串包含的信息由CLIENT\_ParseData解析。

参数说明：

[in]lLoginID，CLIENT\_Login的返回值

[in]szCommand，命令参数，见CLIENT\_ParseData命令参数说明。

[in]nChannelID，通道号

[out]szOutBuffer，输出缓冲，

[out]dwOutBufferSize，输出缓冲大小

[out]error，错误码

|  |  |
| --- | --- |
| 0 | 成功 |
| 1 | 失败 |
| 2 | 数据不合法 |
| 3 | 暂时无法设置 |
| 4 | 没有权限 |

[in]waittime，等待超时时间

1. BOOL CLIENT\_ParseData(char \*szCommand, char \*szInBuffer, LPVOID lpOutBuffer, DWORD dwOutBufferSize, void\* pReserved);

函数说明：此接口配合CLIENT\_GetNewDevConfig使用，在使用CLIENT\_GetNewDevConfig获取字符串配置信息后，使用此接口从配置信息中提取想要的信息。

参数说明：

[in]szCommand，命令参数。

|  |  |
| --- | --- |
| 命令参数 | 相应结构体 |
| CFG\_CMD\_MONITORWALL | 电视墙配置, 对应AV\_CFG\_MonitorWall数组 |
| CFG\_CMD\_SPLICESCREEN | 融合屏配置，对应AV\_CFG\_SpliceScreen数组 |

[in]szInBuffer，输入缓冲，字符配置缓冲。

[out]lpOutBuffer ，输出缓冲，交通结构体类型

[out]dwOutBufferSize ，输出缓冲的大小

[in]pReserved，返回数据长度

1. BOOL CLIENT\_PacketData(char \*szCommand, LPVOID lpInBuffer, DWORD dwInBufferSize, char \*szOutBuffer, DWORD dwOutBufferSize);

函数说明：此接口配合CLIENT\_SetNewDevConfig使用，使用CLIENT\_PacketData后，将打包的信息通过CLIENT\_SetNewDevConfig设置到设备上。

参数说明：

[out]szCommand

命令参数，参见CLIENT\_ParseData中表格

[in]lpInBuffer

输入缓冲，结构体类型参见CLIENT\_ParseData中表格

[in]dwInBufferSize

输入缓冲大小

[out]szOutBuffer

输出缓冲

[in]dwOutBufferSize

输出缓冲大小

1. BOOL CLIENT\_SetNewDevConfig(LONG lLoginID, char\* szCommand, int nChannelID, char\* szInBuffer, DWORD dwInBufferSize, int \*error, int \*restart, int waittime=500);

函数说明：设置配置，按照字符串格式，各个字符串包含的信息由CLIENT\_PacketData组包。

参数说明：

[in]lLoginID

CLIENT\_Login的返回值

[in]szCommand，请参考CLIENT\_ParseData中的说明。

[in]nChannelID，通道号。

[in]szInBuffer，输入缓冲。

[in]dwInBufferSize，输入缓冲大小。

[out]error，错误码。

|  |  |
| --- | --- |
| 0 | 成功 |
| 1 | 失败 |
| 2 | 数据不合法 |
| 3 | 暂时无法设置 |
| 4 | 没有权限 |

[out]restart，配置设置后是否需要重启设备，1表示需要重启，0表示不需要重启。

[in]waittime，等待超时时间。

1. 配置接口使用示例

电视墙和融合屏的配置类似, 这里只介绍电视墙配置.

获取配置:

const int nMaxJsonBuf = 64 \* 1024;

char\* pszJsonBuf = new char[nMaxJsonBuf];

ZeroMemory(pszJsonBuf, nMaxJsonBuf);

int nRetLen = 0;

int i = 0;

// 取电视墙配置, 这里取到的是字符串信息

if (CLIENT\_GetNewDevConfig(m\_lLoginID, CFG\_CMD\_MONITORWALL, -1, pszJsonBuf, nMaxJsonBuf, NULL))

{

// 电视墙输出结构体, 需要进行初始化

const int nMaxMonitorWall = 4;

AV\_CFG\_MonitorWall\* pstuWall = new AV\_CFG\_MonitorWall[nMaxMonitorWall];

for (i = 0; i < nMaxMonitorWall; ++i)

{

pstuWall[i].nStructSize = sizeof(AV\_CFG\_MonitorWall);

for (int j = 0; j < AV\_CFG\_Max\_Block\_In\_Wall; ++j)

{

pstuWall[i].stuBlocks[j].nStructSize = sizeof(AV\_CFG\_MonitorWallBlock);

for (int k = 0; k < AV\_CFG\_Max\_TV\_In\_Block; ++k)

{

pstuWall[i].stuBlocks[j].stuTVs[k].nStructSize = sizeof(AV\_CFG\_MonitorWallTVOut);

}

}

}

// 解析字符串, 得到电视墙配置

if (CLIENT\_ParseData(CFG\_CMD\_MONITORWALL, pszJsonBuf, pstuWall, pstuWall->nStructSize \* nMaxMonitorWall, &nRetLen))

{

for (i = 0; i < nRetLen / pstuWall->nStructSize; ++i)

{

m\_vecWall.push\_back(pstuWall[i]);

}

}

delete[] pstuWall;

}

delete[] pszJsonBuf;

设置配置:

BOOL bResult = FALSE;

const int nMaxJsonLen = 64 \* 1024;

char\* pszJsonBuf = new char[nMaxJsonLen];

*memset(pszJsonBuf, 0, sizeof(nMaxJsonLen));*

*// 对电视墙结构体赋值*

*AV\_CFG\_MonitorWall\* pstuWall = new AV\_CFG\_MonitorWall;*

*memset(pstuWall, 0, sizeof(AV\_CFG\_MonitorWall));*

*pstuWall->nStructSize = sizeof(AV\_CFG\_MonitorWall);*

*strcpy(pstuWall->szName, m\_wndWall.GetName());*

*pstuWall->nColumn = m\_wndWall.GetGridLayout().cx;*

*pstuWall->nLine = m\_wndWall.GetGridLayout().cy;*

*pstuWall->nBlockCount = \_\_min(\_countof(pstuWall->stuBlocks), m\_wndWall.GetBlockCount());*

*for (int i = 0; i < pstuWall->nBlockCount; ++i)*

*{*

*const WMBlock\* pBlock = m\_wndWall.GetBlock(i);*

*AV\_CFG\_MonitorWallBlock& block = pstuWall->stuBlocks[i];*

*block.nStructSize = sizeof(AV\_CFG\_MonitorWallBlock);*

*strcpy(block.szName, pBlock->szName);*

*block.nColumn = pBlock->stuTVSize.cx;*

*block.nLine = pBlock->stuTVSize.cy;*

*block.stuRect.nStructSize = sizeof(AV\_CFG\_Rect);*

*RECT rcGrid = m\_wndWall.GridRect(pBlock);*

*block.stuRect.nLeft = rcGrid.left;*

*block.stuRect.nTop = rcGrid.top;*

*block.stuRect.nRight = rcGrid.right;*

*block.stuRect.nBottom = rcGrid.bottom;*

*block.nTVCount = \_\_min(\_countof(block.stuTVs), pBlock->vecOutputs.size());*

*for (int j = 0; j < block.nTVCount; ++j)*

*{*

*AV\_CFG\_MonitorWallTVOut& output = block.stuTVs[j];*

*output.nStructSize = sizeof(AV\_CFG\_MonitorWallTVOut);*

*int nOutputID = pBlock->vecOutputs[j];*

*output.nChannelID = nOutputID;*

*sprintf(output.szName, "output %d", nOutputID);*

*}*

*for (int k = 0; k < WEEK\_DAY\_NUM; ++k)*

*{*

*block.stuTimeSection[k][0].nBeginHour = 2;*

*block.stuTimeSection[k][0].nEndHour = 3;*

*block.stuTimeSection[k][1].nBeginHour = 12;*

*block.stuTimeSection[k][1].nEndHour = 13;*

*}*

*}*

*// 将结构体打包成字符串信息*

*if (CLIENT\_PacketData(CFG\_CMD\_MONITORWALL, pstuWall, sizeof(\*pstuWall), pszJsonBuf, nMaxJsonLen))*

*{*

*// 设置到设备*

*if (!CLIENT\_SetNewDevConfig(m\_lLoginID, CFG\_CMD\_MONITORWALL, -1, pszJsonBuf, strlen(pszJsonBuf), NULL, NULL, 3000))*

*{*

*//MsgBoxErr("Set config (monitor wall) error.");*

*}*

*else*

*{*

*// 完成电视墙设置, 再接着设置融合屏*

*int nMaxSpliceScreen = m\_wndWall.GetBlockCount();*

*AV\_CFG\_SpliceScreen\* pScreens = NULL;*

*if (nMaxSpliceScreen > 0)*

*{*

*pScreens = new AV\_CFG\_SpliceScreen[nMaxSpliceScreen];*

*memset(pScreens, 0, sizeof(AV\_CFG\_SpliceScreen) \* nMaxSpliceScreen);*

*for (int i = 0; i < nMaxSpliceScreen; ++i)*

*{*

*pScreens[i].nStructSize = sizeof(AV\_CFG\_SpliceScreen);*

*strcpy(pScreens[i].szWallName, pstuWall->szName);*

*strcpy(pScreens[i].szName, pstuWall->stuBlocks[i].szName);*

*pScreens[i].nBlockID = i;*

*}*

*}*

*if (CLIENT\_PacketData(CFG\_CMD\_SPLICESCREEN, (char\*)pScreens, sizeof(AV\_CFG\_SpliceScreen) \* nMaxSpliceScreen,*

*pszJsonBuf, nMaxJsonLen))*

*{*

*if (CLIENT\_SetNewDevConfig(m\_lLoginID, CFG\_CMD\_SPLICESCREEN, -1, pszJsonBuf, strlen(pszJsonBuf), NULL, NULL, 3000))*

*{*

*bResult = TRUE;*

*}*

*}*

*delete[] pScreens;*

*}*

*}*

*delete pstuWall;*

*delete[] pszJsonBuf;*

1. 控制接口
2. BOOL CLIENT\_QueryMatrixCardInfo(LLONG lLoginID, DH\_MATRIX\_CARD\_LIST\* pstuCardList, int nWaitTime = 1000);

函数说明：获取子卡信息.

参数说明：

[in]lLoginID, CLIENT\_Login的返回值

[out] pstuCardList，子卡信息列表。

[in] nWaitTime，等待超时时间。

注意: 所有结构体的第一个参数如果是dwSize, 那就必须给这个字段赋值, 表示该结构体的大小.

如DH\_MATRIX\_CARD\_LIST的定义如下:

// 矩阵子卡列表

typedef struct tagDH\_MATRIX\_CARD\_LIST

{

DWORD dwSize;

int nCount; // 子卡数量

DH\_MATRIX\_CARD stuCards[DH\_MATRIX\_MAX\_CARDS]; // 子卡列表

} DH\_MATRIX\_CARD\_LIST;

使用时必须对结构体对象进行初始化:

DH\_MATRIX\_CARD\_LIST stuCards = {0};

stuCards.dwSize = sizeof(stuCards);

for (int i = 0; i < \_countof(stuCards.stuCards); ++i)

{

stuCards.stuCards[i].dwSize = sizeof(DH\_MATRIX\_CARD);

}

1. BOOL CLIENT\_GetSplitCaps(LLONG lLoginID, int nChannel, DH\_SPLIT\_CAPS\* pstuCaps, int nWaitTime = 1000);

函数说明：获取通道分割能力.

参数说明：

[in]lLoginID, CLIENT\_Login的返回值

[in] nChannel，通道号。

[out] pstuCaps, 分割能力.

[in] nWaitTime，等待超时时间。

1. BOOL CLIENT\_GetSplitSource(LLONG lLoginID, int nChannel, int nWindow, DH\_SPLIT\_SOURCE\* pstuSplitSrc, int nMaxCount, int\* pnRetCount, int nWaitTime = 1000);

函数说明：获取分割显示源信息, 可以获取多个窗口, 也可以是单个窗口

参数说明：

[in]lLoginID, CLIENT\_Login的返回值

[in] nChannel，通道号。

[in] nWindow, 窗口号, -1表示当前所有窗口

[out] pstuSplitSrc, 显示源信息指针.

[in] nMaxCount, pstuSplitSrc对应的显示源数组的大小

[out] pnRetCount, 返回的显示源数量

[in] nWaitTime，等待超时时间。

1. BOOL CALL\_METHOD CLIENT\_SetSplitSource(LLONG lLoginID, int nChannel, int nWindow, const DH\_SPLIT\_SOURCE\* pstuSplitSrc, int nSrcCount, int nWaitTime = 1000);

函数说明：设置分割显示源, 只有解码器的物理屏能够进行多窗口设置.

参数说明：

[in]lLoginID, CLIENT\_Login的返回值

[in] nChannel，通道号。

[in] nWindow, 窗口号, >=0

[out] pstuSplitSrc, 显示源信息指针.

[in] nSrcCount, pstuSplitSrc对应的显示源数组的大小, 这个参数只对解码器的物理屏有效

[in] nWaitTime，等待超时时间。

1. BOOL CLIENT\_GetSplitMode(LLONG lLoginID, int nChannel, DH\_SPLIT\_MODE\_INFO\* pstuSplitInfo, int nWaitTime = 1000);

函数说明: 获取分割模式.

参数说明:

[in]lLoginID, CLIENT\_Login的返回值

[in] nChannel，通道号。

[out] pstuSplitInfo, 分割模式信息

[in] nWaitTime，等待超时时间。

1. BOOL CLIENT\_SetSplitMode(LLONG lLoginID, int nChannel, const DH\_SPLIT\_MODE\_INFO\* pstuSplitInfo, int nWaitTime = 1000);

函数说明: 设置分割模式

参数说明:

[in]lLoginID, CLIENT\_Login的返回值

[in] nChannel，通道号。

[in] pstuSplitInfo, 分割模式信息

[in] nWaitTime，等待超时时间。

1. BOOL CLIENT\_SetSplitWindowRect(LLONG lLoginID, const DH\_IN\_SPLIT\_SET\_RECT\* pInParam, DH\_OUT\_SPLIT\_SET\_RECT\* pOutParam, int nWaitTime = 1000);

函数说明：设置窗口位置

参数说明：

[in]lLoginID, CLIENT\_Login的返回值

[in] pInParam，接口输入参数

[out] pOutParam, 接口输出参数, 可以为NULL

[in] nWaitTime，等待超时时间。

// CLIENT\_SetSplitWindowRect输入参数(设置窗口位置)

typedef struct tagDH\_IN\_SPLIT\_SET\_RECT

{

DWORD dwSize;

int nChannel; // 通道号(屏号)

UINT nWindowID; // 窗口序号

DH\_RECT stuRect; // 窗口位置, 0~8192

BOOL bDirectable; // 坐标是否满足直通条件, 直通是指拼接屏方式下，此窗口区域正好为物理屏区域

} DH\_IN\_SPLIT\_SET\_RECT;

1. BOOL　CLIENT\_GetSplitWindowRect(LLONG lLoginID, const DH\_IN\_SPLIT\_GET\_RECT\* pInParam, DH\_OUT\_SPLIT\_GET\_RECT\* pOutParam, int nWaitTime = 1000);

函数说明：获取窗口位置

参数说明：

[in]lLoginID, CLIENT\_Login的返回值

[in] pInParam，接口输入参数

[out] pOutParam, 接口输出参数

[in] nWaitTime，等待超时时间。

// CLIENT\_GetSplitWindowRect接口输入参数(获取窗口位置)

typedef struct tagDH\_IN\_SPLIT\_GET\_RECT

{

DWORD dwSize;

int nChannel; // 通道号(屏号)

UINT nWindowID; // 窗口序号

} DH\_IN\_SPLIT\_GET\_RECT;

// CLIENT\_GetSplitWindowRect接口输出参数(获取窗口位置)

typedef struct tagDH\_OUT\_SPLIT\_GET\_RECT

{

DWORD dwSize;

DH\_RECT stuRect; // 窗口位置, 0~8192

} DH\_OUT\_SPLIT\_GET\_RECT;

1. BOOL　CLIENT\_OpenSplitWindow(LLONG lLoginID, const DH\_IN\_SPLIT\_OPEN\_WINDOW\* pInParam, DH\_OUT\_SPLIT\_OPEN\_WINDOW\* pOutParam, int nWaitTime = 1000);

函数说明：打开／增加窗口

参数说明：

[in]lLoginID, CLIENT\_Login的返回值

[in] pInParam，接口输入参数

[out] pOutParam, 接口输出参数

[in] nWaitTime，等待超时时间。

// CLIENT\_OpenSplitWindow接口输入参数(开窗)

typedef struct tagDH\_IN\_SPLIT\_OPEN\_WINDOW

{

DWORD dwSize;

int nChannel; // 通道号(屏号)

DH\_RECT stuRect; // 窗口位置, 0~8192

BOOL bDirectable; // 坐标是否满足直通条件, 直通是指拼接屏方式下，此窗口区域正好为物理屏区域

} DH\_IN\_SPLIT\_OPEN\_WINDOW;

// CLIENT\_OpenSplitWindow接口输出参数(开窗)

typedef struct tagDH\_OUT\_SPLIT\_OPEN\_WINDOW

{

DWORD dwSize;

unsigned int nWindowID; // 窗口序号

unsigned int nZOrder; // 窗口次序

} DH\_OUT\_SPLIT\_OPEN\_WINDOW;

1. BOOL　CLIENT\_CloseSplitWindow(LLONG lLoginID, const DH\_IN\_SPLIT\_CLOSE\_WINDOW\* pInParam, DH\_OUT\_SPLIT\_CLOSE\_WINDOW\* pOutParam, int nWaitTime = 1000);

函数说明：关闭窗口

参数说明：

[in]lLoginID, CLIENT\_Login的返回值

[in] pInParam，接口输入参数

[out] pOutParam, 接口输出参数，可以为ＮＵＬＬ

[in] nWaitTime，等待超时时间。

// CLIENT\_CloseSplitWindow接口输入参数(关窗)

typedef struct tagDH\_IN\_SPLIT\_CLOSE\_WINDOW

{

DWORD dwSize;

int nChannel; // 通道号(屏号)

UINT nWindowID; // 窗口序号

} DH\_IN\_SPLIT\_CLOSE\_WINDOW;

// CLIENT\_CloseSplitWindow接口输出参数(关窗)

typedef struct tagDH\_OUT\_SPLIT\_CLOSE\_WINDOW

{

DWORD dwSize;

} DH\_OUT\_SPLIT\_CLOSE\_WINDOW;

1. BOOL　CLIENT\_SetSplitTopWindow(LLONG lLoginID, const DH\_IN\_SPLIT\_SET\_TOP\_WINDOW\*　pInParam, DH\_OUT\_SPLIT\_SET\_TOP\_WINDOW\* pOutParam, int nWaitTime = 1000);

函数说明：设置顶层窗口，返回所有窗口的Ｚ次序．

参数说明：

[in]lLoginID, CLIENT\_Login的返回值

[in] pInParam，接口输入参数

[out] pOutParam, 接口输出参数，可以为ＮＵＬＬ

[in] nWaitTime，等待超时时间。

// 窗口层叠次序

typedef struct tagDH\_WND\_ZORDER

{

DWORD dwSize;

unsigned int nWindowID; // 窗口序号

unsigned int nZOrder; // Z次序

} DH\_WND\_ZORDER;

// CLIENT\_SetSplitWindowLevels接口输入参数(设置窗口次序)

typedef struct tagDH\_IN\_SPLIT\_SET\_TOP\_WINDOW

{

DWORD dwSize;

int nChannel; // 通道号(屏号)

int nWindowID; // 窗口序号

} DH\_IN\_SPLIT\_SET\_TOP\_WINDOW;

// CLIENT\_SetSplitWindowLevels接口输出参数(设置窗口次序)

typedef struct tagDH\_OUT\_SPLIT\_SET\_TOP\_WINDOW

{

DWORD dwSize;

DH\_WND\_ZORDER\* pZOders; // 窗口次序数组

int nMaxWndCount; // 窗口次序数组大小

int nWndCount; // 返回的窗口数量

} DH\_OUT\_SPLIT\_SET\_TOP\_WINDOW;

1. BOOL　CLIENT\_GetSplitWindowsInfo(LLONG lLoginID, const DH\_IN\_SPLIT\_GET\_WINDOWS\* pInParam, DH\_OUT\_SPLIT\_GET\_WINDOWS\* pOutParam, int nWaitTime = 1000);

函数说明：获取当前显示的所有窗口信息, 包括分割模式, 各窗口的位置, Z次序, 显示源等.

参数说明：

[in]lLoginID, CLIENT\_Login的返回值

[in] pInParam，接口输入参数

[out] pOutParam, 接口输出参数

[in] nWaitTime，等待超时时间。

// 区块窗口信息

typedef struct tagDH\_WINDOW\_COLLECTION

{

DWORD dwSize;

int nWindowID; // 窗口ID

BOOL bWndEnable; // 窗口是否有效

DH\_RECT stuRect; // 窗口区域, 自由分割模式下有效

BOOL bDirectable; // 坐标是否满足直通条件

int nZOrder; // 窗口Z次序

BOOL bSrcEnable; // 显示源是否有效

char szDeviceID[DH\_DEV\_ID\_LEN\_EX]; // 设备ID

int nVideoChannel; // 视频通道号

int nVideoStream; // 视频码流类型

int nAudioChannel; // 音频通道

int nAudioStream; // 音频码流类型

} DH\_WINDOW\_COLLECTION;

// 区块收藏

typedef struct tagDH\_BLOCK\_COLLECTION

{

DWORD dwSize;

DH\_SPLIT\_MODE emSplitMode; // 分割模式

DH\_WINDOW\_COLLECTION stuWnds[DH\_MAX\_SPLIT\_WINDOW];// 窗口信息数组

int nWndsCount; // 窗口数量

} DH\_BLOCK\_COLLECTION;

// CLIENT\_GetSplitWindowsInfo接口输入参数

typedef struct tagDH\_IN\_SPLIT\_GET\_WINDOWS

{

DWORD dwSize;

int nChannel; // 通道号

} DH\_IN\_SPLIT\_GET\_WINDOWS;

// CLIENT\_GetSplitWindowsInfo接口输出参数

typedef struct tagDH\_OUT\_SPLIT\_GET\_WINDOWS

{

DWORD dwSize;

DH\_BLOCK\_COLLECTION stuWindows; // 窗口信息

} DH\_OUT\_SPLIT\_GET\_WINDOWS;

1. BOOL CLIENT\_LoadMonitorWallCollection(LLONG lLoginID, const DH\_IN\_WM\_LOAD\_COLLECTION\* pInParam, DH\_OUT\_WM\_LOAD\_COLLECTION\* pOutParam, int nWaitTime = 1000);

函数说明: 加载预案.

参数说明：

[in]lLoginID, CLIENT\_Login的返回值

[in] pInParam，接口输入参数

[out] pOutParam, 接口输出参数, 可以为NULL

[in] nWaitTime，等待超时时间。

// CLIENT\_LoadMonitorWallCollection接口输入参数(载入电视墙预案)

typedef struct tagDH\_IN\_WM\_LOAD\_COLLECTION

{

DWORD dwSize;

int nMonitorWallID; // 电视墙序号

const char\* pszName; // 预案名称

} DH\_IN\_WM\_LOAD\_COLLECTION;

// CLIENT\_LoadMonitorWallCollection接口输出参数(载入电视墙预案)

typedef struct tagDH\_OUT\_WM\_LOAD\_COLLECTION

{

DWORD dwSize;

} DH\_OUT\_WM\_LOAD\_COLLECTION;

1. BOOL CLIENT\_SaveMonitorWallCollection(LLONG lLoginID, const DH\_IN\_WM\_SAVE\_COLLECTION\* pInParam, DH\_OUT\_WM\_SAVE\_COLLECTION\* pOutParam, int nWaitTime = 1000);

函数说明: 保存预案.

参数说明：

[in]lLoginID, CLIENT\_Login的返回值

[in] pInParam，接口输入参数

[out] pOutParam, 接口输出参数, 可以为NULL

[in] nWaitTime，等待超时时间。

// CLIENT\_SaveMonitorWallCollection接口输入参数(保存电视墙预案)

typedef struct tagDH\_IN\_WM\_SAVE\_COLLECTION

{

DWORD dwSize;

int nMonitorWallID; // 电视墙序号

const char\* pszName; // 预案名称

} DH\_IN\_WM\_SAVE\_COLLECTION;

// CLIENT\_SaveMonitorWallCollection接口输出参数(保存电视墙预案)

typedef struct tagDH\_OUT\_WM\_SAVE\_COLLECTION

{

DWORD dwSize;

} DH\_OUT\_WM\_SAVE\_COLLECTION;

1. BOOL CLIENT\_GetMonitorWallCollections(LLONG lLoginID, const DH\_IN\_WM\_GET\_COLLECTIONS\* pInParam, DH\_OUT\_WM\_GET\_COLLECTIONS\* pOutParam, int nWaitTime = 1000);

函数说明: 获取预案信息

参数说明：

[in]lLoginID, CLIENT\_Login的返回值

[in] pInParam，接口输入参数

[out] pOutParam, 接口输出参数

[in] nWaitTime，等待超时时间。

// 区块窗口信息

typedef struct tagDH\_WINDOW\_COLLECTION

{

DWORD dwSize;

int nWindowID; // 窗口ID

BOOL bWndEnable; // 窗口是否有效

DH\_RECT stuRect; // 窗口区域, 自由分割模式下有效

BOOL bDirectable; // 坐标是否满足直通条件

int nZOrder; // 窗口Z次序

BOOL bSrcEnable; // 显示源是否有效

char szDeviceID[DH\_DEV\_ID\_LEN\_EX]; // 设备ID

int nVideoChannel; // 视频通道号

int nVideoStream; // 视频码流类型

int nAudioChannel; // 音频通道

int nAudioStream; // 音频码流类型

} DH\_WINDOW\_COLLECTION;

// 区块收藏

typedef struct tagDH\_BLOCK\_COLLECTION

{

DWORD dwSize;

DH\_SPLIT\_MODE emSplitMode; // 分割模式

DH\_WINDOW\_COLLECTION stuWnds[DH\_MAX\_SPLIT\_WINDOW];// 窗口信息数组

int nWndsCount; // 窗口数量

} DH\_BLOCK\_COLLECTION;

// 电视墙预案

typedef struct tagDH\_MONITORWALL\_COLLECTION

{

DWORD dwSize;

char szName[DH\_DEVICE\_NAME\_LEN]; // 电视墙预案名称

DH\_BLOCK\_COLLECTION stuBlocks[DH\_MAX\_BLOCK\_NUM];// 区块数组

int nBlocksCount; // 区块数量

} DH\_MONITORWALL\_COLLECTION;

// CLIENT\_GetMonitorWallCollections接口输入参数(获取电视墙预案信息)

typedef struct tagDH\_IN\_WM\_GET\_COLLECTIONS

{

DWORD dwSize;

int nMonitorWallID; // 电视墙ID

} DH\_IN\_WM\_GET\_COLLECTIONS;

// CLIENT\_GetMonitorWallCollections接口输出参数(获取电视墙预案信息)

typedef struct tagDH\_OUT\_WM\_GET\_COLLECTIONS

{

DWORD dwSize;

DH\_MONITORWALL\_COLLECTION\* pCollections; // 电视墙预案数组

int nMaxCollectionsCount; // 电视墙预案数组大小

int nCollectionsCount; // 电视墙预案数量

} DH\_OUT\_WM\_GET\_COLLECTIONS;

1. BOOL CLIENT\_RenameMonitorWallCollection(LLONG lLoginID, const DH\_IN\_WM\_RENAME\_COLLECTION\* pInParam, DH\_OUT\_WM\_RENAME\_COLLECTION\* pOutParam, int nWaitTime = 1000);

函数说明: 预案重命名

参数说明：

[in]lLoginID, CLIENT\_Login的返回值

[in] pInParam，接口输入参数

[out] pOutParam, 接口输出参数

[in] nWaitTime，等待超时时间。

// CLIENT\_RenameMonitorWallCollection接口输入参数(预案重命名)

typedef struct tagDH\_IN\_WM\_RENAME\_COLLECTION

{

DWORD dwSize;

int nMonitorWallID; // 电视墙序号

const char\* pszOldName; // 原名称

const char\* pszNewName; // 新名称

} DH\_IN\_WM\_RENAME\_COLLECTION;

// CLIENT\_RenameMonitorWallCollection接口输出参数(预案重命名)

typedef struct tagDH\_OUT\_WM\_RENAME\_COLLECTION

{

DWORD dwSize;

} DH\_OUT\_WM\_RENAME\_COLLECTION;

1. 控制接口使用示例
2. 获取子卡通道信息

由于矩阵通道是不连续的, 即使知道通道总数也不能确定实际有效的通道号, 所以需要通过获取子卡信息得到实际有效的通道号.

DH\_MATRIX\_CARD\_LIST stuCards = {0};

stuCards.dwSize = sizeof(stuCards);

for (int i = 0; i < \_countof(stuCards.stuCards); ++i)

{

stuCards.stuCards[i].dwSize = sizeof(DH\_MATRIX\_CARD);

}

// 获取所有子卡信息

if (CLIENT\_QueryMatrixCardInfo(lLoginID, &stuCards))

{

for (int i = 0; i < stuCards.nCount; ++i)

{

const DH\_MATRIX\_CARD& stuCard = stuCards.stuCards[i];

if (!stuCard.bEnable)

continue;

// 编码卡

if (stuCard.dwCardType & DH\_MATRIX\_CARD\_ENCODE)

{

for (int j = stuCard.nVideoEncChnMin; j <= stuCard.nVideoEncChnMax; ++j)

{

MatrixChannel stuChannel(j, i, j - stuCard.nVideoEncChnMin);

mapVideoEncode[j] = stuChannel;

}

}

// 解码卡

else if (stuCard.dwCardType & DH\_MATRIX\_CARD\_DECODE)

{

// 非连续通道号做法

for (int j = stuCard.nVideoDecChnMin; j <= stuCard.nVideoDecChnMax; ++j)

{

// 获取每个通道的分割能力

DH\_SPLIT\_CAPS stuCaps = { sizeof(DH\_SPLIT\_CAPS) };

CLIENT\_GetSplitCaps(lLoginID, j, &stuCaps);

MatrixChannel stuChannel(j, i, j - stuCard.nVideoDecChnMin, stuCaps.nMaxSourceCount);

for (int k = 0; k < min(stuCaps.nModeCount, \_countof(stuChannel.nSplitModes)); ++k)

{

stuChannel.nSplitModes[k] = stuCaps.emSplitMode[k];

}

mapVideoDecode[j] = stuChannel;

}

}

}

}

1. 设置顶层窗口

DH\_IN\_SPLIT\_SET\_TOP\_WINDOW stuInParam = { sizeof(DH\_IN\_SPLIT\_SET\_TOP\_WINDOW), nTVID, nWindow };

DH\_OUT\_SPLIT\_SET\_TOP\_WINDOW stuOutParam = { sizeof(DH\_OUT\_SPLIT\_SET\_TOP\_WINDOW) };

// pZOders表示返回的窗口Z次序信息数组, 需要用户分配内存

// nMaxWndCount是数组的大小, 必须赋值

stuOutParam.nMaxWndCount = 64;

stuOutParam.pZOders = new DH\_WND\_ZORDER[stuOutParam.nMaxWndCount];

memset(stuOutParam.pZOders, 0, sizeof(DH\_WND\_ZORDER) \* stuOutParam.nMaxWndCount);

for (int i = 0; i < stuOutParam.nMaxWndCount; ++i)

{

stuOutParam.pZOders[i].dwSize = sizeof(DH\_WND\_ZORDER);

}

if (!CLIENT\_SetSplitTopWindow(m\_Matrix.lLoginID, &stuInParam, &stuOutParam))

{

MsgBoxErr("Set top window error.");

}