SmartEye 客户端开发手册

文档名称	SmartEye 客户端开发手册	**版本号**	0.0.2		
		文档编号			
文档类别	开发手册	**文档阶段**			
项目名称		**作者**	方后继		
承担部门	软件部	**批准**			
文档日期	2013-12-18	**使用范围**	GA	页数	25

合肥优视科技 中国·合肥市高新区天通路 14 号软件园 5 号楼东侧 5 层 (230088)
联系电话：+86-551-5317012 5317072 5321270 传真:+86-551-5317532 E-mail:market@besovideo.com

目录

合肥优视科技 中国·合肥市高新区天通路 14 号软件园 5 号楼东侧 5 层 (230088)
联系电话：+86-551-5317012 5317072 5321270 传真:+86-551-5317532 E-mail:market@besovideo.com

1 文档版本

日期	版本号	简要描述
2013-12-18	0.0.2	完善了结构体、函数说明，修改了 0.0.1 中部分说明不明确的内容
2012-04-16	0.0.1	初稿

2 简介

2.1 概述

SmartEye 系统是优视科技的视频网络监控平台，包括 Server 端和 Client 端。BVCU SDK 是开发 SmartEye Client 端的开发套件。

BVCU SDK 支持如下功能：

1、 通过访问 Server，可以访问各种前端监控设备（下面简称 PU），包括浏览音视频、配置 PU 等；
2、 支持 Server 的内建功能，例如权限管理、报警联动等；
3、 支持 Client 的本地功能，例如本地录像、抓图、录像回放等。

BVCU SDK 包括如下组件：

组件	文件	说明
libBVCU 网络库	BVCU.h	主头文件
	BVCUConst.h	常量头文件
	BVCUCommon.h	通用结构体头文件
	PUConfig.h	PU 配置参数头文件
	BVEvent.h	事件相关参数头文件
	BVTask.h	任务相关参数头文件
	RecordPlay.h	回放相关参数头文件
	libBVCU.dll	DLL 文件

	libBVCU.lib	导入库
libSAV 音视频编解码和文件库	SAVCommon.h	通用结构体头文件
	SAVTypes.h	数据类型头文件
	SAVUtil.h	辅助头文件
	SAVCodec.h	编解码头文件
	SAVContainer.h	多媒体文件处理头文件
	libSAV.dll	DLL 文件
Bvdisplay 视频显示及 OSD 库	BVDisplay.h	头文件
	Bvdisplay.dll	DLL 文件
	Pixel.psh	Shader 文件
Portaudio 音频采集和播放库	Portaudio.dll	DLL 文件
Acl 线程辅助库	Acl.dll	DLL 文件
AAAC 客户端安全管理库	aaac.dll	DLL 文件
FTP 文件上传下载库	BVFtp.dll	DLL 文件

libBVCU 是 SDK 的核心库，开发者主要调用这个库完成客户端功能。其他库都是被 libBVCU 调用的，开发者通常不需要直接调用。唯一的例外是 BVDisplay，开发者可能需要用 BVDisplay 做 OSD（也可以不调用 BVDisplay 而直接使用 Windows GDI 等做 OSD）。

2.2 开发环境

BVCU SDK 使用 MSVC 2010 开发， 运行时需要 MSVC 的运行库 msvcr100.dll、msvcp100.dll。目标平台是 Windows XP 及以上版本

SDK 发布包的目录结构如下：

2.3 基本概念

BVCU SDK 涉及了几个概念：Session、Dialog、Command，了解了这几个概念，也就掌握了整个 SDK。

2.3.1 Session

作为客户端，BVCU 必须登录 Server，支持同时登录多个 Server。每次登录创建一个 **Session** 对象，在 BVCU.h 中用 BVCU_HSession 表示。

BVCU_Login 函数登录 Server 并创建一个 Session 对象。

BVCU_Logout 函数退出登录并销毁 Session 对象。

2.3.2 Dialog

BVCU 的最重要功能是浏览（或者叫做监视）Server 传过来的 PU 音视频。一个 PU 可以有多个通道，每个通道可能有音频或者视频。浏览 PU 通道的音视频时，BVCU 需要向 Server 发起浏览请求，这种请求称为“会话”（Dialog），在 BVCU.**h** 中用 BVCU_HDialog 表示。

在实际使用中，根据媒体流方向的不同，Dialog 有几种常见的类型：

类型 1：**Dialog** 请求接收 1 路视频流，或者额外再接收 1 路音频流。这是最常见的情况。

类型 2： **Dialog** 请求接收 PU 的 1 路音频流，并发送 1 路本地音频流给 PU，这被称为语音对讲。

类型 3：**Dialog** 请求发送 1 路本地音频流给多个 PU。这被称为语音广播。

开发者应该注意，类型 2/3（语音对讲）的 **Dialog** 在整个 BVCU 库中只能有一个。

BVCU_Dialog_Open 向 Server 请求发起一个 Dialog 并创建一个 **Dialog** 对象。**Dialog** 的目标由 **BVCU_DialogParam** 结构体中的 **iTargetCount/pTarget** 指定。对类型 1/2 的 **Dialog**，**iTargetCount** 总是 1，对类型 3，**iTargetCount** 可以大于 1。**Dialog** 类型由 **BVCU_DialogParam.iAVStreamDir** 决。

BVCU_DialogControlParam_Render 参数决定了是否在窗口中播放音视频，**BVCU_DialogControlParam_Storage** 决定了是否存储接收到的音视频数

据及存储路径。

BVCU_Dialog_Update 函数通过更改 **BVCU_DialogParam.iAVStreamDir** 来改变一个已经建立的 **Dialog** 的媒体流方向，这只对类型 1 有意义。例如可以去掉/增加一个音频流等。

BVCU_Dialog_Control 函数更改会话的本地设置，包括接收/存储/回放等。

BVCU_Dialog_Snapshot 函数抓取会话中接收到的一帧视频，存为图像文件

BVCU_Dialog_Close 函数关闭 **Dialog** 并销毁 **Dialog** 对象

2.3.3 Command

除了浏览音视频外，BVCU 还需要通过和 **Server** 通讯来支持其他功能。BVCU 需要向 **Server** 发送命令包，每条命令对应一个 **Command** 对象，在 BVCU.h 中用 **BVCU_Command** 表示。

BVCU_SendCmd 函数向 Server 发送命令。

Session 对象包括 **Dialog** 对象和 **Command** 对象，当 **Session** 销毁时，所有已有的 **Dialog**/**Command** 都将被销毁。

3 接口定义

BVCU 库的使用原则：

1．所有需要发送网络数据的接口函数(包括 **BVCU_Login** / **BVCU_Logout** / **BVCU_Dialog_Open** / **BVCU_Dialog_Update** / **BVCU_Dialog_Close** / **BVCU_SendCmd**)，都支持异步回调。如果函数返回失败，则意味着函数在本地发生失败，并没有发出网络命令包给 **Server**，也不会调用回调函数。如果函数返回成功，则意味着函数发出了网络命令包给 **Server**，但命令执行结果通过回调函数来通知应用程序。

2．不可以在回调函数中调用除了 **BVCU_GetSessionInfo / BVCU_GetDialogInfo / BVCU_SetLogLevel / BVCU_GetLogLevel** 之外的任何 BVCU 函数。

3．不可以在回调函数中执行任何可能导致阻塞的操作，例如执行 **Windows API MessageBox** 等。

3.1 常量

由于常量较多，不再这里赘述，请参考 **SDK** 发布包目录下的 **include/BVCUConst.h** 文件。

3.2 结构体

3.2.1 BVCU_GlobalParam

```
typedef struct _BVCU_GlobalParam{
	int iSize;
	void (*OnEvent)(int iEventCode, void* pParam);
}BVCU_GlobalParam;
```

名称	说明	
iSize	本结构体的大小，分配者应初始化为 sizeof(**BVCU_GlobalParam**)	
OnEvent	回调函数。库把库内部检测到的全局事件通知应用程序	
	iEventCode	事件码。参见 **BVCU_EVENT_*** 中的“全局事件”
	pParam	每个事件对应的参数，具体类型参考各个事件码的说明。如果 **pParam** 是 **NULL**，表示无参数

3.2.2 BVCU_ServerParam

```
typedef struct _BVCU_ServerParam{
	int iSize;
	char szServerAddr[BVCU_MAX_HOST_NAME_LEN+1];
	short iServerPort;
	char szClientID[BVCU_MAX_ID_LEN+1];//必须以"CU_"开始
	char szUserAgent[128];
	char szUserName[BVCU_MAX_NAME_LEN+1];
	char szPassword[BVCU_MAX_PASSWORD_LEN+1];
	int iCmdProtoType;
	int  iTimeOut;
	int (*OnNotify)(BVCU_HSession hSession,
BVCU_NotifyMsgContent* pData);
	void (*OnEvent)(BVCU_HSession hSession, int iEventCode,
void* pParam);
```

```
}BVCU_ServerParam;
```

名称	说明	
iSize	本结构体的大小，分配者应初始化为 sizeof(BVCU_ServerParam)	
szServerAddr	Server 地址，域名或者 IP	
iServerPort	Server 端口号	
szClientID	Client ID。必须以"CU_"开始，调用者应选择一种 ID 分配方式，尽量使每台计算机上的 ID 不同且对同一台计算机的 ID 保持不变。	
szUserAgent	应用程序名称。该名称被 Server 端记录到 Log 中	
szUserName	登录用户名	
szPassword	登录密码	
iCmdProtoType	CU 与 Server 之间命令通道使用的传输层协议类型，参见 BVCU_PROTOTYPE_*。目前仅支持 TCP	
iTimeOut	登录过程如果超过 iTimeOut 未收到 Server 回响则认为失败，单位毫秒。该值必须 >0，推荐不小于 60*1000。	
OnNotify	CU 收到 Server 的 Notify 通知后的回调函数	
	hSession	Session 对象
	pData	BVCU_NotifyMsgContent 类型的通知消息内容
OnEvent	回调函数。库把库内部检测到的 Session 事件通知应用程序	
	hSession	Session 对象
	iEventCode	Session 事件码 包括 BVCU_EVENT_SESSION_OPEN 和 BVCU_EVENT_SESSION_CLOSE
	pParam	每个事件对应的参数，具体类型参考各个事件码的说明。如果 pParam 是 NULL，表示无参数

3.2.3 BVCU_Command

```
typedef struct _BVCU_Command BVCU_Command;
struct _BVCU_Command{
        int iSize;
        int iMethod;
        int iSubMethod;
        char szTargetID[BVCU_MAX_ID_LEN+1];
        int iTargetIndex;
        int iTimeOut;
        BVCU_CmdMsgContent stMsgContent;
        void  (*OnEvent)(BVCU_HSession  hSession,  BVCU_Command*
pCommand, int              iEventCode, void* pParam);
};
```

名称	说明

iSize	本结构体的大小，分配者应初始化为 sizeof(BVCU_Command)	
iMethod	命令类型，参见 BVCU_METHOD_*	
iSubMethod	子命令类型，参见 BVCU_SUBMETHOD_	
szTargetID	目标 ID，通常是 PU ID，PU ID 必须以“PU_”开始。设置为空表示命令目标是 Server	
iTargetIndex	从 0 开始的目标附属设备的索引，例如 PU 的云台/通道/音视频 IO 等。设为-1 表示无意义	
iTimeOut	命令超过 iTimeOut 未收到回响则认为失败，单位毫秒。如果设置为 0，则采用 BVCU_ServerParam.iTimeout	
stMsgContent	发送的命令负载	
OnEvent	回调函数。Command 事件通知	
	hSession	Session 对象
	pCommand	本 Command 对象
	iEventCode	Command 事件码 包括 BVCU_EVENT_SESSION_CMD_*
	pParam	每个事件对应的参数，具体类型参考各个事件码的说明。如果 pParam 是 NULL，表示无参数

3.2.4 BVCU_DialogParam

```
typedef struct _BVCU_DialogParam
{
    int iSize;
    void* pUserData;
    BVCU_HSession hSession;
    int iTargetCount;
    const BVCU_DialogTarget* pTarget;
    int iAVStreamDir;
    BVCU_Result      (*afterDecode)(BVCU_Hdialog      hDialog,
SAVCodec_Context* pCodec, SAV_Frame* pFrame);
    BVCU_Result      (*afterRender)(BVCU_HDialog      hDialog,
SAVCodec_Context* pCodec,SAV_Frame* pFrame);
    void (*OnEvent)(BVCU_HDialog hDialog, int iEventCode, void*
pParam);
    int iReserved[4];
}BVCU_DialogParam;
```

名称	说明
iSize	本结构体的大小，分配者应初始化为 sizeof(BVCU_DialogParam)
pUserData	用户自定义数据。通常用于回调通知
hSession	登录 Session
iTargetCount	会话目标个数

pTarget	会话目标数组。pTarget 内存由调用者分配/释放。调用 BVCU_Dialog_Open/BVCU_Dialog_Update 时，SDK 会保留 pTarget 的拷贝。调用 BVCU_GetDialogInfo 时，pTarget 会指向 SDK 内部的拷贝，调用者不可以对拷贝做任何修改	
iAVStreamDir	会话的数据流方向	
afterDecode	CodecThread	对音视频数据，解码完成后调用。调用者可以对解码后的数据进行各种处理。对于视频数据，必须做备份后修改备份，并把pFrame中ppData数据指针指向修改后的备份。对纯数据(SAVCodec_Context.eMediaType==SAV_MEDIATYPE_DATA)，从缓冲区得到数据后立即回调
	pCodec	Codec信息，详见SAVCodec.h
	pFrame	音视频数据:解码得到的原始媒体数据;纯数据:组好帧后的数据
	返回	对纯数据无意义。对音视频数据: BVCU_RESULT_S_OK: pFrame 被显示/播放。 BVCU_RESULT_E_FAILED: pFrame 不被显示/播放。
afterRender	VideoRender Thread/ AudioRender Thread	显示/播放完一个音频或视频帧后调用。用户可以在此处叠加其他效果如文字等。
	pCodec	Codec 信息，详见 SAVCodec.h
	pFrame	解码得到的原始媒体数据
	返回	目前库会忽略返回值
OnEvent	iEventCode	事件码，参见 Dialog 事件
	pParam	每个事件对应的参数,具体类型参考各个事件码的说明。如果 pParam 是 NULL，表示无参数。
iReserved	保留	

3.2.5 BVCU_DialogControlParam

```
typedef struct _BVCU_DialogControlParam{
   int iSize;
   BVCU_DialogControlParam_Network  stNetwork;
```

```
    BVCU_DialogControlParam_Render  stRender;
    BVCU_DialogControlParam_Storage stStorage;
}BVCU_DialogControlParam;
```

名称	说明
iSize	本结构体的大小，分配者应初始化为 sizeof(BVCU_DialogControlParam)
stNetwork	网络
stRender	提取
stStorage	存储

3.2.6 BVCU_NotifyMsgContent

```
typedef struct _BVCU_NotifyMsgContent BVCU_NotifyMsgContent;
struct _BVCU_NotifyMsgContent {
    BVCU_NotifyMsgContent* pNext;
    int iSubMethod;
    char szSourceID[BVCU_MAX_ID_LEN+1];
    int iSourceIndex;
    char szTargetID[BVCU_MAX_ID_LEN+1];
    int iTargetIndex;
    int iDataCount;
    void* pData;
};
```

名称	说明
pNext	一个通知可能包含多条信息。pNext 指向下一条信息。最后一条信息的 pNext 应指向 NULL。
iSubMethod	通知内容的类型，BVCU_SUBMETHOD_
szSourceID	信息源（系统中的网络实体)ID。为空表示是当前登录的 Server
iSourceIndex	信息源的附属设备的索引，从 0 开始，例如 PU 的云台/通道/音视频 IO 等。设为-1 表示无意义
szTargetID	目标 ID。为空表示命令目标是当前登录的 Server
iTargetIndex	从 0 开始的目标附属设备的索引，例如 PU 的云台/通道/音视频 IO 等。设为-1 表示无意义
iDataCount	信息数目
pData	信息数组，数组元素个数等于 iDataCount，pData[0]表示第一个成员，pData[1]表示第 2 个成员。 类型由 iSubMethod 决定

3.2.7 BVCU_CmdMsgContent

```
typedef struct _BVCU_CmdMsgContent BVCU_CmdMsgContent;
struct _BVCU_CmdMsgContent {
    BVCU_CmdMsgContent* pNext;
    int iDataCount;
    void* pData;
};
```

CU 发送的通知包和收到的命令回响内容

名称	说明
pNext	一个通知/回响可能包含多条信息。pNext 指向下一条信息。最后一条信息的 pNext 应指向 NULL, 每个通知/回响的信息类型和顺序是固定的。大多数通知/回响只支持一种数据类型(pNext 是 NULL)
iDataCount	信息数目
pData	信息数组，数组元素个数等于 iDataCount, pData[0]表示第一个成员，pData[1]表示第 2 个成员。类型由具体命令决定

3.3 函数

3.3.1 SDK 初始化与注销

3.3.1.1 BVCU_Initialize

函数名称	BVCU_Result BVCU_Initialize(const BVCU_GlobalParam* pParam)	
功能描述	初始化库。在调用 BVCU 任何接口函数之前调用，并且只能调用一次	
参数说明	pParam	库的全局参数
返回值	BVCU_Result	

3.3.1.2 BVCU_Finish

函数名称	BVCU_Result BVCU_Finish()	
功能描述	销毁库。调用该函数后，不能再调用任何 BVCU 函数	
参数说明		
返回值	BVCU_Result	

3.3.2 登入/退出服务器

1 登录

LIBBVCU_API BVCU_Result BVCU_Login(BVCU_HSession* phSession, BVCU_ServerParam* pParam);

函数名称	BVCU_Login	
功能描述	登录服务器	
参数说明	phSession	输出参数，登录对应的session
	pParam	输入参数，要登录Server信息，包括IP、用户名、密码等，参考BVCU_ServerParam结构(示例)
返回值		BVCU_Result

3. 3.3 实时音视频浏览

相关函数：

1、 LIBBVCU_API BVCU_Result BVCU_Dialog_Open(BVCU_HDialog* phDialog, BVCU_DialogParam* pParam, BVCU_DialogControlParam* pControl);

//打开音视频会话

2、LIBBVCU_API BVCU_Result BVCU_Dialog_Control(BVCU_HDialog hDialog, BVCU_DialogControlParam* pParam);

//音视频会话的控制

函数名称	BVCU_Dialog_Open	
功能描述	打开会话	
参数说明	phDialog	输出参数，会话句柄
	pParam	打开会话的会话参数
	pControl	打开会话的控制参数
常见返回值	BVCU_RESULT_S_OK	调用正确完成。结果通过 pParam 中提供的回调函数 OnEvent 通知调用者
	BVCU_RESULT_E_UNSUPPORTED	不支持的操作，例如在不支持对讲的通道上要求对讲
	BVCU_RESULT_E_FAILED	其他错误导致失败

函数名称	BVCU_Dialog_Control
功能描述	更改会话的本地设置，包括接收/存储/回放等。此函数不需要与 Server 通讯。

参数说明	hDialog	会话句柄，通过 BVCU_Dialog_Open 获得
	pParam	新会话的控制参数
常见返回值	BVCU_RESULT_S_OK	成功
	BVCU_RESULT_E_NOTEXIST	会话不存在
	BVCU_RESULT_E_UNSUPPORTED	不支持的操作
	BVCU_RESULT_E_FAILE 等	其他错误导致失败

3. 3.4 各种命令

大部分的功能都是通过发送命令完成任务，例如：设备列表，设备信息，云台等信息的获取和设置都需要通过客户端发送命令。

1、发送命令函数：

```
LIBBVCU_API BVCU_Result BVCU_SendCmd(BVCU_HSession hSession, BVCU_Command* pCommand);
```

函数名称	BVCU_SendCmd	
功能描述	CU发送命令。该函数是异步的，命令完成后触发BVCU_Command.OnEvent回调通知。	
参数说明	hSession	对服务器的一个连接，由BVCU_Login获得
	pCommand	命令参数，参见BVCU_Command
返回值	BVCU_RESULT_S_OK	调用正确完成。结果通过OnEvent通知调用者
	BVCU_RESULT_E_NOTEXIST	登录Session不存在，即未登录
	BVCU_RESULT_E_FAILED等	其他错误导致失败

3. 3.5 其他函数

待补充

4 代码示例

4.1 SDK 初始化

下面的代码演示了初始化和销毁 SDK。

```
#include <BVCU.h>
void OnEventGlobal(int iEventCode, void *pParam)
```

```
{
    switch(iEventCode)
    {
    case BVCU_EVENT_AUDIOINPUT_DISCONNECTED:
        printf("检测到未接麦克风\n");
        break;
    case BVCU_EVENT_AUDIOINPUT_DISCONNECTED:
        printf("接入了麦克风\n");
        break;
    default
            break;
    }
}

int main(int argc, char *argv[])
{
    BVCU_Result bvResult;

    //初始化 SDK
    BVCU_GlobalParam gp;

    memset(&gp, 0, sizeof(gp));
    gp.iSize = sizeof(gp);
    gp.OnEvent = OnEventGlobal;
    bvResult = BVCU_Initialize(&gp);
    if(BVCU_Result_FAILED(bvResult))
    {
        printf("初始化 SDK 失败\n");
        exit(0);
    }

    //配置 log 级别

    BVCU_SetLogLevel(BVCU_LOG_DEVICE_CONSOLE, BVCU_LOG_LEVEL_WARN);//配置控制台输出级别，通常仅对 windows console 程序有效

    BVCU_SetLogLevel(BVCU_LOG_DEVICE_FILE, BVCU_LOG_LEVEL_WARN);//配置 log 文件输出级别

    //....
    //可以调用其他 BVCU 函数
    //....

    //退出
```

```
    BVCU_Finish();

    return 0;
}
```

4.2 Session 登录/退出

下面的代码片段演示了登录/退出 Server 及各种事件和 PU 上下线通知的处理。

```
#include <BVCU.h>

enum
{
    SESSION_STATUS_IDLE = 0,
    SESSION_STATUS_LOGIN,
    SESSION_STATUS_LOGOUT
};

static volatile int iSessionStatus = SESSION_STATUS_IDLE;
static void OnEventSession(BVCU_HSession hSession, int iEventCode, void *pParam)
{
    BVCU_Event_Common *pEvent = (BVCU_Event_Common *)pParam;
    BVCU_SessionInfo info;
    BVCU_GetSessionInfo(hSession, &info);
    printf("收到 Server 的 Event\n");
    iSessionStatus = SESSION_STATUS_LOGOUT;
    if (pEvent->iResult == BVCU_RESULT_E_DISCONNECTED)
    {
        printf("Session 断线\n");
    }
    else if(iEventCode == BVCU_EVENT_SESSION_CLOSE)
    {
        printf("Session Logout\n");
    }
    else if(iEventCode == BVCU_EVENT_SESSION_OPEN)
    {
        if(SAV_Result_SUCCEEDED(pEvent->iResult))
        {
            printf("Session Login 成功\n");
            iSessionStatus = SESSION_STATUS_LOGIN;
        }
        else if(pEvent->iResult == BVCU_RESULT_E_TIMEOUT)
```

```
        {
            printf("Session Login 失败：超时\n");
        }
        else if(pEvent->iResult == BVCU_RESULT_E_AUTHORIZE_FAILED)
            printf("Session Login 失败：验证未通过\n\n");
    }
    else
    {
        printf("Session Login 失败\n");
    }
}
}

static int OnNotifySession(BVCU_HSession hSession, BVCU_NotifyMsgContent *pData)
{
    printf("收到 Server 的 Notify\n");
    if (pData)
    {
        if (pData->iSubMethod == BVCU_SUBMETHOD_PU_CHANNELINFO &&
pData->iDataCount > 0)
        {
            BVCU_PUChannelInfo *puChannelInfo = (BVCU_PUChannelInfo *)(pData->pData
);
            printf("index      puID        puName       status \r\n");
            printf("     channelName mediaDir PTZIndex \r\n");
            for (int i = 0; i < pData->iDataCount; i ++)
            {
                printf("[% 4d]  %s : %s : %d \r\n", i + 1, puChannelInfo[i].szPUID,
 puChannelInfo[i].szPUName, puChannelInfo[i].iOnlineStatus);
                for (int j = 0; j < puChannelInfo->iChannelCount; j ++)
                {
                    printf("     %s : %d : %d\r\n", puChannelInfo[i].pChannel[j].sz
Name, puChannelInfo[i].pChannel[j].iMediaDir, puChannelInfo[i].pChannel[j].iPTZInde
x);
                }
            }
        }
    }
    return 0;
}

int main(int argc, char *argv[])
{
```

```
//初始化 SDK 等
//....

BVCU_Result bvResult;
BVCU_HSession hSession = NULL;
BVCU_ServerParam sp;
memset(&sp, 0, sizeof(sp));
sp.iSize = sizeof(sp);
strcpy(sp.szServerAddr, "127.0.0.1");
sp.iServerPort = 9701;
strcpy(sp.szClientID, "CU_F39324DA25"); //CU ID
strcpy(sp.szUserAgent, "test for BVCU");
strcpy(sp.szUserName, "besovideo");
strcpy(sp.szPassword, "123456");
sp.iCmdProtoType = BVCU_PROTOTYPE_TCP;
sp.iTimeout = 60 * 1000;
sp.OnNotify = OnNotifySession;
sp.OnEvent = OnEventSession;

bvResult = BVCU_Login(&hSession, &sp);
if(BVCU_Result_FAILED(bvResult))
{
    printf("Login 失败\n");
    exit(0);
}
//....做其他事情...
//等待登录成功
while(iSessionStatus == SESSION_STATUS_IDLE)
{
    Sleep(100);
}


//....做其他事情...

//退出登录
BVCU_Logout(hSession);

//....做其他事情...

//...清理 SDK...
return 0;
```

```
}
```

4.3 Command 示例

```
static void cmd_OnEvent(BVCU_HSession hSession, BVCU_Command *pCommand, int iEventC
ode, void *pParam)
{
    BVCU_Event_SessionCmd *pEvent = (BVCU_Event_SessionCmd *)pParam;

    switch(pCommand->iMethod)
    {
    case BVCU_METHOD_QUERY:
        if(pCommand->iSubMethod == BVCU_SUBMETHOD_GETPULIST)
        {
            printf("Received GetPuList callback,data : \r\n");
            if (pEvent->stContent.iDataCount > 0)
            {
                BVCU_PUChannelInfo *puChannelInfo = (BVCU_PUChannelInfo *)(pEvent->
stContent.pData);
                printf("index puID puName status \r\n");
                printf(" channelName mediaDir PTZIndex \r\n");
                set_console_color(10);
                for (int i = 0; i < pEvent->stContent.iDataCount; i ++)
                {
                    printf("[%4d]%s:%s:%d\r\n", i + 1, puChannelInfo[i].szPUID, puC
hannelInfo[i].szPUName, puChannelInfo[i].iOnlineStatus);
                    for (int j = 0; j < puChannelInfo[i].iChannelCount; j ++)
                    {
                        printf("%s:%d:%d\r\n", puChannelInfo[i].pChannel[j].szName,
 puChannelInfo[i].pChannel[j].iMediaDir, puChannelInfo[i].pChannel[j].iPTZIndex);
                    }
                }
            }
        }
        else if(pCommand->iSubMethod == BVCU_SUBMETHOD_PU_DEVICEINFO)
        {
            if (BVCU_Result_SUCCEEDED(pEvent->iResult) && pEvent->stContent.iDataCo
unt > 0)
            {
                BVCU_PUCFG_DeviceInfo *lpDevInfo = (BVCU_PUCFG_DeviceInfo *)pEvent-
>stContent.pData;
                printf("deviceID : %s \r\n", lpDevInfo->szID);
```

```
                printf("productName : %s \r\n", lpDevInfo->szProductName);
                printf("softwareVersion : %s \r\n", lpDevInfo->szSoftwareVersion);
                printf("hardwareVersion : %s \r\n", lpDevInfo->szHardwareVersion);
                printf("puType : %d \r\n", lpDevInfo->iPUType);
                printf("languageIndex : %d \r\n", lpDevInfo->iLanguageIndex);
                printf("name : %s \r\n", lpDevInfo->szName);
                printf("wifiCount : %d \r\n", lpDevInfo->iWIFICount);
                printf("radioCount : %d \r\n", lpDevInfo->iRadioCount);
                printf("channelCount : %d \r\n", lpDevInfo->iChannelCount);
                printf("videoInCount : %d \r\n", lpDevInfo->iVideoInCount);
                printf("AudioInCount : %d \r\n", lpDevInfo->iAudioInCount);
                printf("AudioOutCount: %d \r\n", lpDevInfo->iAudioOutCount);
                printf("ptzCount : %d \r\n", lpDevInfo->iPTZCount);
                printf("serialPortCount : %d \r\n", lpDevInfo->iSerialPortCount);
                printf("AlertInCount : %d \r\n", lpDevInfo->iAlertInCount);
                printf("AlertOutCount: %d \r\n", lpDevInfo->iAlertOutCount);
                printf("GPSCount : %d \r\n", lpDevInfo->iGPSCount);
                printf("StorageCount : %d \r\n", lpDevInfo->iStorageCount);
            }
        }
        else if(pCommand->iSubMethod == BVCU_SUBMETHOD_PU_PTZATTR)
        {
            if (pEvent->stContent.iDataCount > 0)
            {
                BVCU_PUCFG_PTZAttr *ptzAttr = (BVCU_PUCFG_PTZAttr *)pEvent->stConte
nt.pData;
                printf("puID : %s \r\n", pCommand->szTargetID);
                printf("ptzIndex : %d \r\n", pCommand->iTargetIndex);
                printf("支持的协议列表：  \r\n");
                for (int i = 0; i < BVCU_PTZ_MAX_PROTOCOL_COUNT; ++i)
                {
                    if(ptzAttr->iPTZProtocolAll[i] == BVCU_PTZ_PROTO_INVALID)
                        break;
                    printf("         : %d \r\n", ptzAttr->iPTZProtocolAll[i]);
                }
                printf("ptzProtoIndex : %d \r\n", ptzAttr->iPTZProtocolIndex);
                printf("iAddress : %d \r\n", ptzAttr->iAddress);
                printf("iBaudRate : %d \r\n", ptzAttr->stRS232.iBaudRate);
                printf("iDataBit : %d \r\n", ptzAttr->stRS232.iDataBit);
                printf("iFlowControl : %d \r\n", ptzAttr->stRS232.iFlowControl);
                printf("iParity : %d \r\n", ptzAttr->stRS232.iParity);
                printf("iStopBit : %d \r\n", ptzAttr->stRS232.iStopBit);
            }
```

```
        }
        break;
    case BVCU_METHOD_CONTROL:
        if(pCommand->iSubMethod == BVCU_SUBMETHOD_PU_DEVICEINFO)
        {
            printf("Control : BVCU_SUBMETHOD_PU_DEVICEINFO : %d \r\n", pEvent->iRes
ult);
        }
        printf(" method_control : %d \r\n", pEvent->iResult);
        break;
    case BVCU_METHOD_SUBSCRIBE:
        printf("cmd_Onfinish: method_subscribe : %d \r\n", pEvent->iResult);
        break;
    default:
        printf("Received an unknown command callback ! %d \r\n", pEvent->iResult);
    }
}

int main(int argc, char *argv[])
{
    BVCU_HSession hSession;

    //...登录 Server....

    //发送 GetPUList 命令
    BVCU_Result bvResult;
    BVCU_Command cmdInfo;
    memset(&cmdInfo, 0, sizeof(cmdInfo));
    cmdInfo.iSize = sizeof(cmdInfo);
    cmdInfo.iMethod = BVCU_METHOD_QUERY;
    cmdInfo.iSubMethod = BVCU_SUBMETHOD_GETPULIST;
    cmdInfo.OnEvent = cmd_OnEvent;
    bvResult = BVCU_SendCmd(hSession, &cmdInfo);
    if(BVCU_Result_FAILED(bvResult))
    {
        printf("发送命令失败\n");
        exit(0);
    }

    //...等待命令结果，这期间可以执行其他任务...

    ...
```

```
}
```

4.4 Dialog 示例

```
static void OnEventDialog(BVCU_HDialog hDialog, int iEventCode, void *pParam)
{
    switch(iEventCode)
    {
    case BVCU_EVENT_DIALOG_OPEN:
        if(BVCU_Result_SUCCEEDED(((BVCU_Event_Common *)pParam)->iResult))
        {
            BVCU_DialogInfo info;
            BVCU_GetDialogInfo(hDialog, &info);
            printf("\r\nOpen dialog succeeded ! Device ID = %s \r\n", info.stParam.
pTarget[0].szID);
        }
        else
        {
            printf("\r\nOpen dialog failed!  \r\n");
        }
        break;
    case BVCU_EVENT_DIALOG_CLOSE:
    {
        BVCU_DialogInfo info;
        BVCU_GetDialogInfo(hDialog, &info);
        printf("\r\nClose dialog ! Device ID = %s \r\n", info.stParam.pTarget[0].sz
ID);

    }
    break;
    case BVCU_EVENT_STORAGE_FILE_REQUIRENAME:
        if(((BVCU_Event_Storage *)pParam)->iResult == BVCU_RESULT_S_OK)
        {

            printf("\r\nrequire file name %s succeeded !", ((BVCU_Event_Storage *)p
Param)->szFileName);
            printf(" time : %lld \r\n", ((BVCU_Event_Storage *)pParam)->iTimestamp)
;
        }
        break;
    case BVCU_EVENT_STORAGE_FILE_OPEN:
        if(((BVCU_Event_Storage *)pParam)->iResult == BVCU_RESULT_S_OK)
        {
```

```
            printf("\r\nopen file %s succeeded !", ((BVCU_Event_Storage *)pParam)->
szFileName);
        }
        else if(((BVCU_Event_Storage *)pParam)->iResult == BVCU_RESULT_E_OUTOFSPACE
)
        {
            printf("\r\nopen file %s no space error !", ((BVCU_Event_Storage *)pPar
am)->szFileName);
        }
        else
            printf("\r\nopen file %s unknown error !", ((BVCU_Event_Storage *)pPara
m)->szFileName);
        printf(" time : %lld \r\n", ((BVCU_Event_Storage *)pParam)->iTimestamp);
        break;
    case BVCU_EVENT_STORAGE_FILE_CLOSE:
        if(((BVCU_Event_Storage *)pParam)->iResult == BVCU_RESULT_S_OK)
        {
            printf("\r\nclose file %s succeeded !", ((BVCU_Event_Storage *)pParam)-
>szFileName);
        }
        else
            printf("\r\nclose file %s unknown error !", ((BVCU_Event_Storage *)pPar
am)->szFileName);
        printf(" time : %lld \r\n", ((BVCU_Event_Storage *)pParam)->iTimestamp);
        break;
    case BVCU_EVENT_STORAGE_ERROR:
        if(((BVCU_Event_Storage *)pParam)->iResult == BVCU_RESULT_E_OUTOFSPACE)
        {
            printf("\r\nwrite Error file: %s no space error !", ((BVCU_Event_Storag
e *)pParam)->szFileName);
        }
        else
            printf("\r\nwrite Error file: %s unknown error !", ((BVCU_Event_Storage
 *)pParam)->szFileName);
        printf(" time : %lld \r\n", ((BVCU_Event_Storage *)pParam)->iTimestamp);
        break;
    }
}


int main(int argc, char *argv[])
{
    BVCU_HSession hSession;
```

```
    BVCU_HDialog hDialog;
    //... 登录 Server 并获取 PU 列表....

    BVCU_DialogParam dlgParam;
    BVCU_DialogControlParam dlgCtlParam;
    BVCU_DialogTarget targetPU;

    memset(&dlgParam, 0, sizeof(dlgParam));
    memset(&dlgCtlParam, 0, sizeof(dlgCtlParam));
    dlgCtlParam.iSize = sizeof(dlgCtlParam);
    dlgParam.iSize = sizeof(dlgParam);

    dlgParam.hSession = hSession;
    dlgParam.OnEvent = OnEventDialog;
    dlgParam.iTargetCount = 1;
    dlgParam.pTarget = &targetPU;
    targetPU.iIndexMajor = atoi(0);//通道号
    targetPU.iIndexMinor = -1;//流号
    strcpy(targetPU.szID, "PU_55aa0000");//这里应该填写 PU 列表中存在的 PU ID

    //网络参数
    dlgCtlParam.stNetwork.iTimeOut = 60 * 1000;//60 秒超时
    dlgCtlParam.stNetwork.iDelayMin = 1000;//最小延时 1 秒钟
    dlgCtlParam.stNetwork.iDelayMax = 6000;//最大延时 6 秒钟
    dlgCtlParam.stNetwork.iDelayVsSmooth = 3;//平滑设置

    //存储参数
    char dir[128] = "c:";
    strcpy_s(dlgCtlParam.stStorage.szFilePath, sizeof(dlgCtlParam.stStorage.szFileP
ath), dir); //配置录像
    dlgCtlParam.stStorage.iFileLenInSeconds = 600;//以 10 分钟为单位生成文件
    dlgParam.iAVStreamDir = BVCU_MEDIADIR_VIDEORECV | BVCU_MEDIADIR_AUDIORECV; //类
型 1 的 Dialog

    //音视频播放参数
    dlgCtlParam.stRender.iPlackbackVolume = 50;//播放音量
    dlgCtlParam.stRender.iCaptureVolume   = 0;//采集音量

    //dlgCtlParam.stRender.hWnd = NULL;//如果在窗口播放视频，这里设置为窗口句柄
    //GetClientRect(hWnd, &dlgCtlParam.stRender.rcDisplay);//如果在窗口播放视频，这里
设置为播放区域

    BVCU_HDialog dlg;
```

```
    bvResult = BVCU_Dialog_Open(&hDialog, &dlgParam, &dlgCtlParam);
    if(BVCU_Result_FAILED(bvResult))
    {
        printf("打开对话失败\n");
        //...失败处理逻辑
    }

    //...等待 OnEvent...

    //...现在应该在存储文件或者听到声音...

    //关闭 Dialog
    BVCU_Dialog_Close(hDialog);
}
```