Windows发射端SDK接口文档
当前版本：V1.0

苏州必捷网络有限公司

修订记录

版本号	拟制/修改人	拟制/修改日期	评审人	修改内容要点
0.9	吴刚	2018/4		
1.0	吴刚	2018/11		
文档初始拟定时，可不填“评审人”以及“修改内容要点”				

目 录

1 概述

1.1 目的

用于指导使用必捷无线投屏SDK开发Windows发送端应用程序

1.2 读者对象

本文档适用于Windows发送端应用程序的开发人员和测试人员

1.3 缩略语定义

缩写名称	英文	中文
BJCast		必捷无线投屏协议

2 基本概念

2.1 无线投屏系统中的发射端

发射端为无线投屏系统的视频源，它通过发现协议寻找到接收端设备，通过BJCast协议发起投屏会话，负责录制WINDOS本地的屏幕视频和声音，经过编码并传输到接收端进行播放和呈现。

2.2 无线投屏系统中的接收端

接收端负责接入发射端发起的无线投屏会话，接收发射端的音视频数据并进行解码和呈现。

接收端类型:

必捷投屏硬件盒子：必捷网络研发的BJ系列投屏设备。实现了BJcast的接收端功能，必捷网络自定义的设备发现协议。

基于BJCast接收端SDK开发的应用或硬件设备：可以使用BJCast投屏协议发起投屏。

2.3 发现协议

发射端在发起投屏会话到接收端时，首要需要确定接收端的通信IP和通信端口。发现协议用于自动发现在无线投屏系统的接收端设备。

BJCast SDK中预定义了一种的发现协议，可用于发现在同一局域网内的BJCast接收端。

客户也可以根据自身业务系统的需要自定义投屏协议。

2.4 BJCast无线投屏协议

BJCast无线投屏协议包括BJCast发现协议和BJCas投屏协议。BJCast发现协议的接口将在4.2中描述，BJCas投屏协议在4.3中进行描述。

3 范围

3.1 BJCast投屏应用程序描述

总体框架分为两层

BJCast SDK： 实现投屏控制协议和媒体传输和处理协议部分。使用c/c++开发，以动态链接库的形式提供服务。

应用层：开发的具体应用部分，我司交付DEMO源代码，具体与客户应用集成，可做针对性开发。

本文档主要描述BJCast SDK的接口

3.2 SDK交付物

- Dll库
- 头文件
- SDK接口文档

4 接口

BJCast SDK的API接口包括以下部分：

1) 发现协议接口：实现了BJCast发现协议。参考4.2，其相关接口在ControlManagerIntf类中定义。
2) 投屏协议接口：SDK提供的投屏会话控制接口。参考4.3，其相关接口在ControlManagerIntf类中定义。
3) 用户回调接口：SDK中的ControlManagerNotify类定义了投屏会话相关的通知回调接口，用户程序需要实现该接口。参考4.4。

上述接口都在ControlManagerIntf.h中定义。

4.1 SDK初始化

ControlManagerIntf *createControlManager(ControlManagerNotify *);

功能:

创建BJCast SDK管理实例

输入参数：

ControlManagerNotify*，用户实现的ControlManagerNotify接口实例。

输出参数：

无

返回参数：

ControlManagerIntf实例，后续所有的投屏协议接口都需要调用该实例的成员方法。应用程序需要维护好该实例。当应用退出时析构该实例。

其它：

该函数不能多次调用，全局只允许有一个ControlManagerIntf实例。

4.2 BJCast发现协议接口

用户在选择发现协议时，可选择SDK自带的BJCast发现协议，也可以选择不启用BJCast发现协议。当前BJCast发现协议只能发现同一局域网子网内的接收端设备，跨局域网等应用场景不适用。

用户可根据自身业务需要来决定是否启用BJCast发现协议，举例：

某集团需要将投屏功能集成到现有的办公管理系统中，通过统一的管理平台来管理投屏盒子，此时在该场景下客户就可以自定义投屏设备的管理和发现协议，达到由现有的管理平台统一管理投屏设备。

若用户不需要使用BJCast发现协议，则不需要关心该部分接口。

BJCast发现协议接口在ControlManagerIntf类中定义，下列接口是ControlManagerIntf类的成员函数，用户需要使用createControlManager返回的实例来调用相关接口。

4.2.1 开始自动发现

S32 startServerDetecting()
功能：
发射端调用该接口后，SDK内部将定时在局域网内发送探测消息，探测是否有BJCast接收端。BJCast接收端收到探测消息后会回复探测响应消息。BJCast发射端收到探测响应消息后会通过onServerDetected()接口通知发现的接收端信息。
输入参数
无
输出参数：
无
返回值：
0：成功

4.2.2 停止自动发现

S32 stopServerDetecting()
功能:
发射端调用该接口后，SDK内部将停止自动发现接收端设备。
输入参数
无
输出参数：
无
返回值：
0：成功

4.2.3 探测接收端

S32 probe(const ServerInfo &info)
功能:
在已经知道IP，PORT等信息后，向该端口发送探测包，探测该IP和端口上是否运行有BJCast接收端服务。可用于探测接收端当前是否正常运行。BJCast发射端收到探测响应消息后会通过onServerDetected()接口通知发现的接收端信息。
输入参数
无
输出参数:
无
返回值：
0：成功

4.3 BJCast投屏协议接口

BJCast投屏协议接口在ControlManagerIntf类中定义，下列接口是ControlManagerIntf类的成员函数，用户需要使用createControlManager返回的实例来调用相关接口。

4.3.1 登录接收端

S32 login(const S8 *ip, U16 port, const S8* pin = NULL)
功能：
用于登录及连接接收端。
输入：
ip: 接收端IP地址
port: BJCast服务端口，默认BJCast接收端使用8188端口提供投屏接入服务。
pin：投屏PIN码，如果没有PIN码该参数输入NULL或空字符串，默认为NULL。

输出：
无
返回值：
0：成功 其它：失败

4.3.2 设置投屏模式

S32 setMirrorMode(ControlManagerMirrorMode mode)
功能:
设置投屏模式，投屏模式有如下几种:
CMMM_Custom, //普通模式
CMMM_Screensharing, //办公模式，适合播放PPT之类应用,表现为延时较低
CMMM_RealtimeVideo, //影音模式,表现为播放流畅
CMMM_Performance //高性能模式,表现为帧率较高
输入参数
无
输出参数：
无
返回值：
0：成功

4.3.3 分辨率变化通知接口模式

S32 onDisplayChange()
功能:
应用程序监控到桌面分辨率变化等情况需要调用该接口通知SDK进行相应处理
输入参数
无
输出参数：
无
返回值：
0：成功

4.3.4 获取屏幕个数

int screenCount()
功能:
获取可录制的屏幕个数，一般为1，如果接了双显示屏时为2
输入参数
无
输出参数：
无
返回值：
屏幕个数，一般为1，如果接了双显示屏时为2

4.3.5 设置屏幕

void setScreen(S32 screen)
功能:
设置当前发射端录制哪一个屏幕,此接口可以动态生效。默认为主屏幕。
输入参数
screen：屏幕索引，其取值范围为0- (screenCount() -1)
当存在多个屏幕时，0为主屏幕，1为辅屏幕

输出参数：
无
返回值：
无

4.3.6 获取音频采集接口数

S32 micphoneCount()
功能:
获取音频采集接口个数
输入参数
无
输出参数：
无
返回值：
音频采集接口个数，取值范围为>=0

4.3.7 获取音频采集接口名称

void micphoneName(S32 index, char name[128])
功能:
获取某个音频采集接设备的名称
输入参数
index：音频采集设备索引，其取值范围为0- (micphoneCount()-1)
输出参数：
name：音频采集设备的名称
返回值：
无

4.3.8 选择音频采集口

void selectMicphone(S32 index)
功能：
投屏功能中用于选择录制具体某个音频采集接口的音频数据
输入参数：
index：音频采集设备索引，其取值范围为0- (micphoneCount()-1)
输出参数：
无
返回值：
无

4.3.9 发起投屏

void startMirror()
功能：
投屏功能中用于选择录制具体某个音频采集接口的音频数据
输入参数：
无
输出参数：
无
返回值：
无
其它：

如果发起投屏失败，则会通过应用通知接口onCallEnd通知应用投屏会话结束原因

4.3.10 结束投屏

void stopMirror()

功能：

结束投屏会话

输入参数：

index：音频采集设备索引

输出参数：

无

返回值：

无

4.3.11 申请全屏

S32 requestFullscreen()

功能：

申请全屏。若接收端为必捷投屏盒子且实现了多路投屏功能，当多路投屏情况下，可通过该接口申请全屏，此时该发射端对应画面在接收端会占据所有的屏幕。

输入参数：

无

输出参数：

无

返回值：

0：成功　其它:失败

其它：

若无多路发射端同时投屏到一个接收端的场景，发射端无需关注此接口。若接收端未实现全屏控制逻辑，则该接口可能不生效。

4.3.12 退出全屏

S32 exitFullscreen()

功能：

退出全屏。若接收端为必捷投屏盒子且实现了多路投屏功能，当多路投屏情况下且已经通过申请全屏接口，可通过该接口退出全屏，此时该发射端对应画面在接收端会退出全屏状态。

输入参数：

无

输出参数：

无

返回值：

0：成功　其它:失败

其它：

若无多路发射端同时投屏到一个接收端场景，发射端无需关注此接口。若接收端未实现全屏控制逻辑，则该接口可能不生效。

4.4 应用通知接口

应用通知接口在ControlManagerNotify类中定义，下列接口是ControlManagerNotify类的成员函数，用户需要实现该接口，并在SDK初始化时传入其实例。SDK内部会调用该实例的成员方法通知应用程序会话状态。

4.4.1 发现到接收端通知

void onServerDetected(ServerInfo *info)

功能：

发现到接收端之后，通知应用层接口，可能会搜索到多个接收端,则该函数会调用多次。

输入参数：

ServerInfo*：接收端的IP地址，服务端口，版本号等信息

输出参数：

无

返回值：

无

其它：

若用户未启用BJCast默认的发现协议，则不用关心该接口。

4.4.2 投屏结束通知

void onCallEnd(const Reason &)

功能：

通知当前的投屏会话结束。应用需要根据该接口做相应处理，如结束底层投屏会话，并更新界面状态。

输入参数：

Reason：Reason为投屏会话结束的原因。参考第6章错误码说明。

输出参数：

无

返回值：

无

4.4.3 全屏通知接口

void onCallUpdate(bool flag)

功能：

全屏通知接口状态，通知反射端是否处于全屏状态。应用程序需要根据该状态更新界面状态和全屏控制相关状态。

输入参数：

flag：true表示当前发射端画面处于全屏状态，false表示当前发射端画面退出全屏状态

输出参数：

无

返回值：

无

其它：

若无多路发射端同时投屏到一个接收端场景，发射端无需关注和实现此接口。若接收端未实现多络投屏和全屏控制逻辑，该接口可能不生效。

5 最简DEMO代码说明

```
#include "ControlManagerIntf.h"      //包含ControlManagerIntf.h

#include <string>
#include <stdint.h>

#define TEST_ServerDetecting 1
```

```
class CMN : public ControlManagerNotify   // CMN类实现ControlManagerNotify接口
{
   public:
      void onServerDetected(ServerInfo *info)   //返现到接收端信息会通过此接口返回
      {
            printf("%s %s %d %s %u\n", info->serial, info->ip, info->port, info->name, info->version);
      }

      void onCallEnd(const Reason &reason) {   //会话结束
           cm->stopMirror();          //是否SDK内部投屏资源
      }
      void onCallUpdate(bool fullscreen) {}

      ControlManagerIntf *cm;
};

int main(int argc, char **argv)
{
      std::string svc_ip("192.168.9.147");   //指定接收端IP
      if (argc > 1)
      {
            svc_ip.assign(argv[1]);
      }
      printf("svc_ip is %s\n", svc_ip.c_str());

      uint16_t svc_port = 8188;   //接收端端口
      CMN c;
      ControlManagerIntf *cm = createControlManager(&c); // SDK初始化

      #if TEST_ServerDetecting        //宏开关为1时，启用
      ServerInfo i;
      i.port = svc_port;

      strcpy(i.ip, svc_ip.c_str());

      for (;;)
      {
            cm->startServerDetecting();
            //    cm->probe(i);
            system("pause");
            break;
      }
      #endif

       cm->setMirrorMode(CMMM_Performance);     //设置为影音模式

      //接入接收端，无PIN码则填""或者NULL，否则填实际的PIN码
      S32 ret = cm->login(svc_ip.c_str(), svc_port,"");
      if (ret != E_OK)
      {
            printf("login svc_ip    %s failed\n", svc_ip.c_str());
            delete cm;
```

```
        return 0;
    }

    //开始发起投屏
    cm->startMirror();

    system("pause");

    //结束投屏
    cm->stopMirror();

    delete cm;    //程序退出，释放SDK控制实例

    return 0;
}
```

参考demo，一次投屏的典型流程如下：

- 用户需要实现ControlManagerNotify接口，如CMN，其实例作为createControlManager方法的入参。
- 调用createControlManager方法初始化SDK，获取ControlManagerIntf实例
- 调用login接口发起接收端的连接会话
- 调用startMirror发起投屏
- 调用stopMirror结束投屏
- 析构ControlManagerIntf实例，退出应用程序

6 错误码说明

应用程序涉及的常见错误码的含义：

错误码值	描述	出现场景
0	成功	正常结束投屏的情况
-1	未知原因失败	
-6	连接接收端失败，检查网络是否正确	网络异常
-7	会话超时	Login后，15秒内未发起投屏会话，此时会出现超时
-14	投屏会话被拒绝	接收端拒绝投屏会话接入
-15	接收端屏幕已满，无法接入	接收端屏幕已满，不允许接入
-16	投屏码不正确	投屏码错误