聯陽半導體
ITE Tech. Inc.

iTE 可視對講系統

Control Server SDK

V0.9

ITE TECH. INC.

修訂記錄

修訂日期	**修訂說明**	頁次
2014/10/23	初建版本 V1.0	

1. 前言

本 Control Server 系统是为了配合嵌入式系统架构开发的，Control Server 可有效的管理门口机、室内机、管理机、小门口机设备以及手机 App，使上述的这些设备之间友好的协作，完成整个系统框架下的任务。

控制服务器（Control Server）是为了配合整个可视对讲系统开发的一套 PC 软件。再功能上，可视对讲/监视、门禁管理是两项主要的应用，它同时也有管理系统有对各个终端设备的交互管理，如：升级管理，设备管理，本地系统管理。同时也可提供更多的社区服务模块（社区信息发布，社区管理，三表抄送等）。本文档主要介绍整个系统的框架，以及各个模块的 SDK 说明，旨在让各开发者快速了解系统，迅速开发。

1.1 编写目的

本文档的编写为下阶段的设计、开发提供依据，为项目组成员对需求的详尽理解，以及在开发开发过程中的协同工作提供强有力的保证。同时本文档也作为项目测试评审验收的依据之一。

1.2 本产品涉及用户与角色

外来访客：例如快递员

管理员：社区工作人员，负责管理和维护整套社区系统。

住户：室内机终端和门铃机终端的使用者

1.3 文档名词约定

门口机：通常指单元门口机，一栋房子的入口处安防的机器。也是文档中指定的大门口机。

室内机：住户家中的设备终端。

小门口机：也称为别墅机或二次确认机。

管理中心机：类似于室内机的管理终端，通常是管理人员用来接听和拨出使用。用于日常事务处理。

管理中心 sever：数据存储和管理，社区管理的核心设备。同时也担任管理中心机的职责：呼叫，监视，应答。

名称换算：

大门口机/门口机 --> 大厅机 (Lobby Phone)

小门口机/室外机/门前机/门外机/别墅机/二次确认机 --> 门铃机 (Door Camera)

室内机 --> 室内机 (Indoor Phone)

管理机/管理中心机--> 管理机 (Administrator Unit)

PC 管理中心/管理中心 sever --> 控制服务器 (Control Server)

1.4 Control Server SDK 开发环境

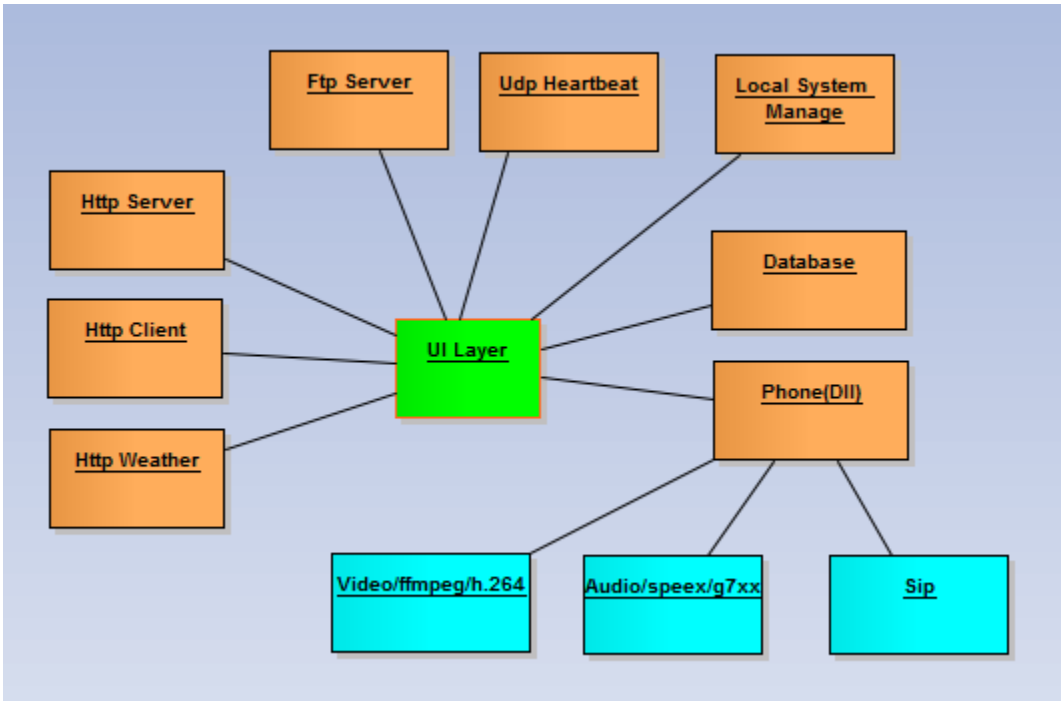
系统：Windows 7 操作系统（双核 4G 以上）/windows xp 操作系统 带有声卡

工具：Microsoft Visual Studio 2010 开发工具

数据库支持：MySQL 关系型数据库管理系统

1.5 设计思想

1.6 SDK 架构



Sdk 架构图

Http Server：主要是处理 web Server 消息处理，里面有很多 http 请求在里面，向逻辑层返回收到的 http 请求消息。

Http Clinet: 主要处理 Control Server 请求命令封装。

Http Weather：主要处理用 Http 方式获取 Rss 天气预报源。

Ftp Server：控制服务器构建的 Ftp 服务，提供终端机升级服务。

UDP Heartbeat：心跳包，控制服务器主动下发 UDP 心跳包，检测设备在线状态。

Phone Mode:：封装的一个模块主要处理呼叫、响铃、视频/音频接受、播放等接口。

Audio/speex/g711：提供 Audio 编解码。

Video/h.264：提供 Video 编解码。

SIP：提供 sip 操作接口。

Database :：提供数据库操作接口

Local system Manage ：本地设置接口。

UI Layer：UI 层。

1.7 SDK 模块介绍

1. Http Server

该模块主要提供 http 服务，与自定义 http 协议详见 Protocol 文档主要定义。代码在 HttpProtocolIO.h 和 HttpProtocolIO.cpp、AppInstance.h 和 AppInstance.cpp, AppInstance 是对 HttpProtocolIO 的管理。HttpProtocolIO.h 中代码如下：

```
#pragma once
class CNetTermBase;
#include "../Http/curl_http/http_parser.h"
#include "../Common/csthread.h"
typedef struct ST_HTTPURL
{
	int nPort;
	string sHost;
	string sPath;
	string sSchema;
	string sQuery;
	string sFragment;
	string sUserInfo;
	char * pBody;
}stHttp_Url;

typedef struct ST_HTTPMSG
{
```

```
        int nMsg;
        string sip;
        CNetTermBase * pNet;
        void * param1;
        void * param2;
}stHttpMsg;

typedef void( CALLBACK *pHttpCallBack) (stHttpMsg msg);
class CHttpProtocolIO:
csthread
{
public:
        CHttpProtocolIO(void);
        ~CHttpProtocolIO(void);
        //CHttpProtocolIO(pHttpCallBack pFun)
        //{
        //      httpcallback = pFun;
        //}
        //typedef int (*http_data_cb) (http_parser*, const char *at, size_t length);
        //int On_Http_Url(http_parser*, const char *at, size_t length);

        CRITICAL_SECTION m_cs;
        HANDLE m_hThread;
        typedef vector<char> TvcBuffer;
        typedef struct ST_S
        {
                TvcBuffer vcbuffer;
                int nLen;
                string sip;
                CNetTermBase * pNet;
        }stBlockInfo;
        std::list<stBlockInfo> m_ltRecv;
        //std::vector<char> m_vcRecv;

public:
        void __stdcall HttpReceive(CNetTermBase *,int nLen);
        static bool  HTTP_URL_Get(char * pUrl, stHttp_Url & http_url);
        void  HTTP_Analyse(char * pHttp,int len,SOCKET socket);
void run();
```

```
Public:   /*向外提供接口*/
        static bool SendHttpResponse(stHttpMsg msg,char * buffer,int len);


        static string GetResponseHead(int bodysize);
        static string GetHttpServerIp();
        static void RegCallFun(pHttpCallBack pFun);
/*end*/

private:
string ReadHttpHeaderLine(char * pRequest, int Len);
        string GetValue(string str,string sKey);
        int GetHttpType(string s);
        bool FileDownLoad(stHttpMsg msg);
        DWORD ReadFileToBuffer( string szFile, char *buff, DWORD nFileSize );
        bool GetFileAttr(char * ,WIN32_FIND_DATA & wfd);

        void  PostCommand(stHttpMsg msg,stHttp_Url rq);
        void  GetCommand(stHttpMsg msg,stHttp_Url rq);
        bool ToSend(stHttpMsg msg,char * buffer,int len,bool bconn = false);
        bool ToClose(stHttpMsg msg);

    };
```

该类主要提供三个接口供使用者调用，如下所示：

```
/*
功能：Http Server 向请求端 发送 Response 消息。
函数名称： SendHttpResponse
参数：stHttpMsg  :包装的 http 消息结构体
                typedef struct ST_HTTPMSG
                {    int nMsg;  //消息类型
                     string sip;//http请求ip
                     CNetTermBase * pNet; // 请求CNetTermBase 指针，使用者不需要关心
                     void * param1;      //附加指针1
                     void * param2;      //附加指针2
                }stHttpMsg;
    Char * buffer:返回response字符串口
```

Int len :字符串长度

返回值：bool 发送成功与否。

*/

```
static bool SendHttpResponse(stHttpMsg msg,char * buffer,int len);

/*
功能：获取 Response Heade 字符串
函数名称：GetResponseHead
参数：int bodysize: Response body 长度
返回值：string 类型的字符
*/
static string GetResponseHead(int bodysize);
/
功能：获取 http Server 端的 ip
函数名称：GetHttpServerIp
参数：NULL
返回值：string 类型的 ip 地址
*/
static string GetHttpServerIp();
/*
功能：注册回调函数 ，向上返回 Http 消息信息
函数名：RegCallbackFun
参数：typedef void( CALLBACK *pHttpCallBack) (stHttpMsg msg)类型的函数指针。
返回值：NULL
*/
static void RegCallbackFun(pHttpCallBack pFun)
```

如要添加新的协议可在 HttpProtocolIO.cpp 中可在以下三个函数中去修改。

```
Int GetHttpType(string s)
```

该函数将 http Request 字段映射为 int 型的 id 消息类型，以后判断消息用 id 来区分。如有新的消息可在此中添加消息 id 和映射。

```
void  PostCommand(stHttpMsg msg,stHttp_Url rq);
void  GetCommand(stHttpMsg msg,stHttp_Url rq);
```

上面两个函数是处理 Post 和 Get 的 http 请求的，如要将有效内容返回给回调函数，就在此处添加消

息处理程序。

如下为 AppInstance.h 文件代码：

```
class CHttpInstance
{
protected:
	CHttpInstance(void);

public:
	virtual ~CHttpInstance();
Public:/*向外提供的三个接口*/
	static CHttpInstance * CreateHttpServer(int nPort =0);//创建 Http 服务
	/*初始 Http Server 信息*/
	void   InitionlizeServer();
	/*注销 Http Server 信息*/
	void   DeInitializeServer();
/*end*/
Public:
	//NetList manager
	bool RegistrNetTerm(CNetTermBase *pTermContext);
	bool UnRegistrNetTerm(CNetTermBase *pTermContext){ return
m_pNetTermList->RemoveNewItem(pTermContext);}
	void RemoveNetTerm(CNetTermBase *pTerm){};
	CNetTermList* GetNetTermList(){return m_pNetTermList;}
	bool FindNetTerm(CNetTermBase * pNet);


protected:
	static CHttpInstance * m_pHttpThis;
	CNetTermList        * m_pNetTermList;
	CIOCPModel          * m_pIocp;
	CHttpProtocolIO     * m_HttpIO;

};
```

该类主要提供三个接口：

/*

功能：创建 Http 服务，创建一个 CHttpInstance 对象，且有且仅有一个，当再次调用的时候，返回的是已经存在的对象指针。

函数名：CreateHttpServer

参数：int nPort :http server 端口

返回值：CHttpInstance 对象指针

*/

static CHttpInstance * CreateHttpServer(int nPort =0);//创建 Http 服务 调用此函数，创建 CHttpProtocolIO 对象。

/*

功能：初始 Http Server 信息并启动 server

函数名：InitionlizeServer

参数：NULL

返回值：void

*/

void InitionlizeServer();

/*

功能：停止 Http Server

函数名：DeInitializeServer

参数：NULL

返回值：void

*/

void DeInitializeServer();

详细说明：调用 CreateHttpServer 创建一个 Http Server 服务对象，InitionlizeServer 初始化并启动 Http 服务， 具体调用就在 CHttpProtocolIO 中去实现。当结束服务的时候，调用 InitionlizeServer 函数，即可释放所有服务资源。

2. Http Client

该模块中主要封装了 http 客户端请求，HttpClientIO.h 代码如下所示：

```
#pragma once
#include "../Http/curl_http/http.h"
class CHttpClientIo
{
```

```
public:
 CHttpClientIo(void);
 ~CHttpClientIo(void);
public:
 enum {
        HTTP_REQUEST_UPGRADE = 0,
        HTTP_REQUEST_NEWMESSAGE

 };
private:
 int m_Error;
 CURL* curl_obj;
 string m_sResponse;
public:
 void SendHttpMsg(string ip,char * pBuffer, int nType);
 bool SendHttpNewMessage(string ClientIp,int nCount, int nType);
 bool SendOpenLock(string ClientIp, string ro);
 bool SendNewCardMessage(string ClientIp, string sVer);
 bool SendRebootMessage(string ClientIp);
 bool SendUpdateAddrBookMsg(string ClientIp, string sFtp);
 bool SendUpdateResMsg(string ClientIp, string sFtp);

 bool SendCancelSecurityMsg(string ClientIp);
 bool GetDeviceCfgInfo(string ClientIp,string & sInfo);
 string GetHttpResponse();
 bool GetImgFormServer(string sUrl, string sImgAddr);
private:

 string CommandUpgrade(string ClientIp, string ServerIp,string sVer);
 string CommandNewMessage(string ClientIp,int nCount, int nType);
 //string CommandOpenLock();

 int GetLastError();

 int Send(string ip,string sUrl, string &sRet);
 int Send(string ip,string sUrl, char * pBuffer,int& nLen);
};
```

可调用接口如下：

void SendHttpMsg(string ip,char * pBuffer, int nType);

```
    /*发送 http 提示消息*/
bool SendHttpNewMessage(string ClientIp,int nCount, int nType);
    /*发送开锁消息*  /
bool SendOpenLock(string ClientIp, string ro);
    /*发送门禁卡更新消息*/
bool SendNewCardMessage(string ClientIp, string sVer);
    /*发送重启消息*/
bool SendRebootMessage(string ClientIp);
    /*发送更细地址薄消息*/
bool SendUpdateAddrBookMsg(string ClientIp, string sFtp);
    /*发送更新资源文件消息*/
bool SendUpdateResMsg(string ClientIp, string sFtp);
    /*发送撤销安防报警消息*/
bool SendCancelSecurityMsg(string ClientIp);
    /*向 device 获取 cfg 文件*/
bool GetDeviceCfgInfo(string ClientIp,string & sInfo);
    /*获取当前响应消息*/
string GetHttpResponse();
    /*从服务器下载图片*/
bool GetImgFormServer(string sUrl, string sImgAddr);
```

详细说明：详细见 HttpClientIO.cpp 文件

3. Ftp Server

在 TCP/IP 网络中，客户机可通过文件传输协议 FTP(File Transport Protocol)下载或加载文件服务器上的文件，以实现资源共享，各个终端设备 FTP Server 相连接，可访问服务器上的大量程序和信息。FTP Server 已成为互联网上的一种重要资源。在本控制服务器中，FTP Server 主要是放置升级软件文件，和各种 UPG 文件，终端机可通过 ftp url 来访问服务器，获取相应的资源。见“FtpserverIO.h”头文件：

```
#pragma once
/*
    簡單的 FTP 服務器
    僅支持 FTP 下載服務
*/
#include "../Common/csthread.h"
typedef DWORD （CALLBACK* pFtpCallBackFun)(LPVOID lparam, LPVOID wparam);
class CFtpServerIO: public csthread
```

```
{
public:
 CFtpServerIO(void);
 ~CFtpServerIO(void);
private:
 HANDLE m_hThread ;
 bool m_bRunFlag;

private:
 char m_cLocalAddr[32];

public:
 //回調函數指針
 static pFtpCallBackFun mpOnBeginFun;
 static pFtpCallBackFun mpOnProgressFun;
 static pFtpCallBackFun mpOnEndFun;
 static string m_strCurDir;
public:/*向外提供接口*/
 bool RunFtpServer();
 bool StopFtpServer();
 void SetCurrentDir(string sDir);

 //注冊函數
 void RegBeginFun(pFtpCallBackFun pFun);
 void RegProgressFun(pFtpCallBackFun pFun);
 void RegEndFun(pFtpCallBackFun pFun);

private:
 void run();
};
```

在该模块主要提供如下函数：

```
/*启动 ftp service*/
bool RunFtpServer();
/*停止 ftp Service*/
 bool StopFtpServer();
/*设置 ftp 根目录*/
 void SetCurrentDir(string sDir);
```

```
//注册函數
/*注册升级开始回调消息函数*/
void RegBeginFun(pFtpCallBackFun pFun);
 void RegProgressFun(pFtpCallBackFun pFun);
/*注册升级结束回调消息函数*/
 void RegEndFun(pFtpCallBackFun pFun);
```

详细说明：

在使用此服务的时候,需要 RunFtpServer() 函数启动 ftp service ,然后 SetCurrentDir(string sDir) 设置 ftp 根目录，如果调用设置，ftp service 会默认为当前程序的执行目录，当程序结束的时候调用 StopFtpServer()，服务将被终止。

4. Database

每个数据库表类都有一个基类 CADODatabase, 应用到的数据表类都是有基类派生出来的，CADORecordset 是 Ado 操作产生的数据集接口包装，详细见代码。

数据库表设计：（以下数据字段类型都是参考 mysql 数据支持类型，如果使用其他关系型或者非关系型数据库，请自行转换）

1、数据表：设备表（device)

字段	_id	_ip	_roomid	_alias	_mac	_status	_type	_sm	_gw	_aVer	_cVer
描述	PRIMARY KEY	设备 ip	设备地址	设备别名	设备 mac 地址	在线状态	设备类型	子网掩码	网关	设备地址薄版本号	设备门禁卡版本号
类型	INTEGER	Varchar(20)	Varchar(20)	varchar(50)	varchar(50)	INTEGER	INTEGER	varchar(50)	varchar(50)	varchar(50)	varchar(50)
其他	NOT NULL AUTO_INCREMENT	NOT NULL									此字段在门口机上才有效

设备类型（定义了 7 种设备类型，可以自行在此基础上增加）：

0：管理中心。1：小门口机。2：单元门口机。3：栋门口机。4：小区门口机（围墙机）5：室内机；6 管理中心机

2、数据表：留影留言记录（leaveword)

字段	_id	_filenames	_src_addr	_dst_addr	_ttime	_readflag
描述	Primary key	留影留言文件路径	Lobby phone address	Indoor Address(无分机）	留影留言时间点	读标记
类型	INTEGER	Varchar(250)	Varchar(20)	Varchar(20)	Varchar(20)	Varchar(20)
其他	NOT NULL AUTO_INCREMENT					

3、数据表：门禁卡信息（iccard）

字段	_id	_icno	_roomid	_username	_ictype	_icpassword	_available	_time	_work_begin	_work_end
描述	Primary key	Ic 卡序列号	开卡地址	开卡人	开卡类型		有效	开卡时间	有效时段其实时间	有效时段结束时间
类型	INTEGER	VARCHAR(20)	VARCHAR(20)	VARCHAR(20)	VARCHAR(20)	VARCHAR(20)	VARCHAR(20)	VARCHAR(20)	VARCHAR(20)	VARCHAR(20)
其他										

4、数据表：安防事件表（EventWarn)

字段	_id	_src_addr	_ttime	_handle status	_handle time	_type	_channel	_action	_handler
描述	Primary key	上报地	上报时	处理状	处理事	事件类	事件通	动作	处理人

		址	间	态	件时间	型	道		
类型	INTEGER	VARCHAR(20)	VARCHAR(20)	INTEGER	VARCHAR(20)	INTEGER	INTEGER	VARCHAR(20)	VARCHAR(20)
其他									

5、数据表：对讲记录上报（EventCallout)

字段	_id	_from_addr	_to_addr	_ttime	_type	_img_addr	
描述	Primary key	主叫地址	被叫地址	记录时间点	记录类型	记录影像图片	
类型	INTEGER	VARCHAR(20)	VARCHAR(20)	VARCHAR(20)	INTEGER	VARCHAR(255)	
其他							

6、数据表：普通事件表（EventCommon)

字段	_id	_src_addr	_ttime	_handlestatus	_handletime	_type	_content	_action	_handler
描述	Primary key	事件上报地址	事件上报时间	处理状态	处理时间	事件类型	上报内容	动作	处理人
类型	INTEGER	VARCHAR(20)	VARCHAR(20)	INTEGER	VARCHAR(20)	INTEGER	VARCHAR(255)	VARCHAR(20)	VARCHAR(255)
其他									

7、数据表：升级文件表（Upgrade)

字段	_id	_filepath	_ver	_type	_time
描述	Primary key	升级文件保存地址	升级文件版本号	升级文件类型	保存时间

类型	INTEGER	VARCHAR (255)	VARCHAR (20)	INTEGER	VARCHAR (20)
其他					

8、数据表：发布信息（publishinfo)

字段	_id	_topic	_dst_addr	_ttime	_type	_fmt	_readflag	
描述	Primary key	发布主题	发布目的地址	发布时间点	发布信息类型	格式	读标记	
类型	INTEGER	VARCHAR (250)	VARCHAR (20)	VARCHAR (20)	VARCHAR (20)	VARCHAR (20)	INTEGER	
其他								

9、数据表：住户信息表（HolderInfo)

字段	_id	_name	_sex	_phoneno	_roomid	_isholder	
描述	Primary key	住户姓名	住户性别	住户手机号	住户房号	是否房主	
类型	INTEGER	VARCHAR(20)	INTEGER	VARCHAR(20)	VARCHAR(20)	INTEGER	
其他							

10、数据表：楼栋属性表（buildproperty)

字段	_id	_qu	_dong	_type		
描述	Primary key	区	栋	类型		
类型	INTEGER	VARCHAR(20)	VARCHAR(20)	INTEGER		
其他						

说明：类型（三种类型）：独栋无单元，多单元型，别墅型

11、数据表：门口机密码表（doorbellpassword)

字段	_id	_roomid	_password	_ttime			
描述	Primary key	门口机地址	门口机密码	上传时间点			
类型	INTEGER	VARCHAR(20)	VARCHAR(20)	VARCHAR(20)			
其他							

12、数据表：系统用户表（user)

字段	_id	_userno	_usename	_powerid	_password
描述	Primary key	用户 id	用户名称	权限 id	用户登录密码
类型	INTEGER	VARCHAR(20)	VARCHAR(20)	INTEGER	VARCHAR(20)
其他					

13、数据表：用户权限表（authority)

字段	_id	_name	_authority
描述	Primary key	权限名称	权限
类型	INTEGER	VARCHAR(20)	INTEGER
其他			

14、数据表：地址薄规则表（AddrAssociate）

字段	_id	_addrA	_typeA	_addrB	_typeB	_des

描述	Primary key	源地址	规则类型	目标地址	目标地址设备类型	描述
类型	INTEGER	VARCHAR(20)	INTEGER	VARCHAR(20)	INTEGER	VARCHAR(20)
其他						

“ado.h”文件中代码如下：

```
class CADODatabase
{
public:
 CString GetDatabaseSet();
 ////////////////////////////////////////////////////////////////////////
 // 構造和析構函數
 CADODatabase()
 {
        //初始 COM 庫
        ::CoInitialize(NULL);

        m_pConnection = NULL;
        m_strConnection = _T("");
        m_pConnection.CreateInstance(__uuidof(Connection));
 }

 ~CADODatabase()
 {
        Close();
        m_pConnection.Release();
        m_pConnection = NULL;
        ::CoUninitialize();
 }
 ////////////////////////////////////////////////////////////////////////
 //打開關閉連接、判斷是否打開、執行不返回記錄的 SQL 命令
 BOOL Open(LPCTSTR lpstrConnection = _T(""));
 void Close();
 BOOL IsOpen();
```

```
BOOL Execute(LPCTSTR lpstrExec);

//////////////////////////////////////////////////////////////////////
// 事務處理
long BeginTransaction() ;
BOOL CommitTransaction();
BOOL RollbackTransaction();
//////////////////////////////////////////////////////////////////////
//設置和獲得連接串、獲得連接指針、錯誤信息
_ConnectionPtr GetActiveConnection() {return m_pConnection;};
void SetConnectionString(LPCTSTR lpstrConnection)
        {m_strConnection = lpstrConnection;};
CString GetConnectionString()
        {return m_strConnection;};
CString GetLastError()
        {return m_strLastError;};
protected:
void dump_com_error(_com_error &e);

protected:
_ConnectionPtr m_pConnection;
CString m_strConnection;
CString m_strLastError;
private:
CString m_strDatabaseSet;
};

class CADORecordset
{
public:
//Get Float Value
BOOL GetFieldValue(LPCTSTR lpFieldName,float& fValue);
//未知、SQL 語句、表、存儲過程
enum cadoOpenEnum
{
        openUnknown = 0,
        openQuery = 1,
        openTable = 2,
        openStoredProc = 3
```

```
};
enum cadoPositionEnum
{

	positionUnknown = -1,
	positionBOF = -2,
	positionEOF = -3
};
//搜索方向，對應于 ADO 的
enum cadoSearchEnum
{
	searchForward = 1,
	searchBackward = -1
};
////////////////////////////////////////////////////////////////////////////
//構造、析構函數
CADORecordset()
{
	m_pRecordset = NULL;
	m_strQuery = _T("");
	m_pRecordset.CreateInstance(__uuidof(Recordset));
	m_nSearchDirection = CADORecordset::searchForward;
}
//利用 CADODatabase 類參數初始化
CADORecordset(CADODatabase* pAdoDatabase);
~CADORecordset()
{
	Close();
	m_pRecordset.Release();
	m_pRecordset = NULL;
	m_strQuery = _T("");
}
////////////////////////////////////////////////////////////////////////////
//打開關閉記錄集
BOOL Open(_ConnectionPtr mpdb, LPCTSTR lpstrExec = _T(""), int nOption = CADORecordset::openUnknown);
BOOL Open(LPCTSTR lpstrExec = _T(""), int nOption = CADORecordset::openUnknown);
void Close();
////////////////////////////////////////////////////////////////////////////
```

```
//獲得記錄個數，判斷記錄集是否打開
DWORD GetRecordCount();
BOOL IsOpen();
//////////////////////////////////////////////////////////////////////////
//存取字段
BOOL SetFieldValue(int nIndex, CString strValue);
BOOL SetFieldValue(LPCTSTR lpFieldName, CString strValue);
BOOL SetFieldValue(int nIndex, int nValue);
BOOL SetFieldValue(LPCTSTR lpFieldName, int nValue);
BOOL SetFieldValue(int nIndex, long lValue);
BOOL SetFieldValue(LPCTSTR lpFieldName, long lValue);
BOOL SetFieldValue(int nIndex, double dblValue);
BOOL SetFieldValue(LPCTSTR lpFieldName, double dblValue);
BOOL SetFieldValue(int nIndex, COleDateTime time);
BOOL SetFieldValue(LPCTSTR lpFieldName, COleDateTime time);
BOOL SetFieldValue(LPCTSTR lpFieldName, vector<BYTE>);

BOOL GetFieldValue(LPCTSTR lpFieldName, double& dbValue);
BOOL GetFieldValueMoney(LPCTSTR lpFieldName, double& fValue);
BOOL GetFieldValue(int nIndex, double& dbValue);
BOOL GetFieldValue(LPCTSTR lpFieldName, long& lValue);
BOOL GetFieldValue(int nIndex, long& lValue);
BOOL GetFieldValue(LPCTSTR lpFieldName, int& nValue);
BOOL GetFieldValue(int nIndex, int& nValue);
BOOL GetFieldValue(LPCTSTR lpFieldName, CString& strValue);
BOOL GetFieldValue(int nIndex, CString& strValue);
BOOL GetFieldValue(LPCTSTR lpFieldName, COleDateTime& time);
BOOL GetFieldValue(int nIndex, COleDateTime& time);

BOOL GetFieldValue(LPCTSTR lpFieldName, vector<BYTE> &vb);
//////////////////////////////////////////////////////////////////////////
//更新記錄集的記錄
BOOL Update();
BOOL AddNew();
BOOL Requery(long Options);
BOOL Delete();
/*
BOOL IsFieldNull(LPCTSTR lpFieldName);
BOOL IsFieldNull(int nIndex);
```

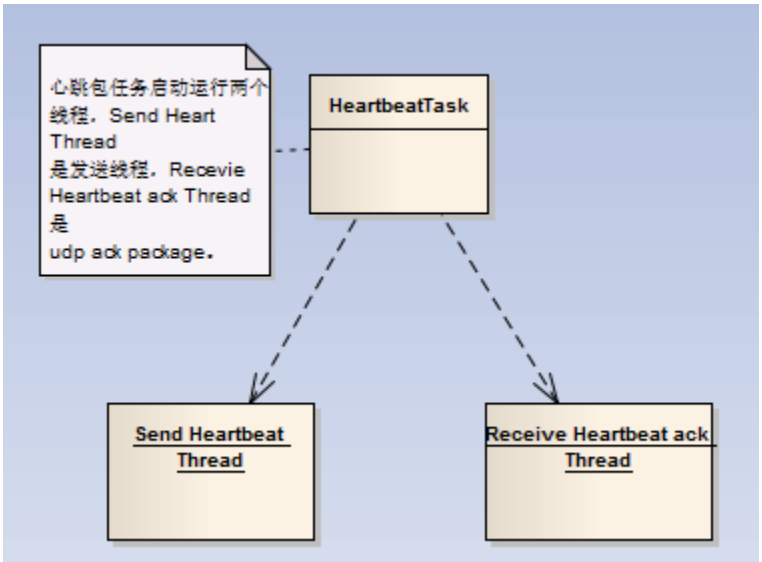
```
BOOL IsFieldEmpty(LPCTSTR lpFieldName);
BOOL IsFieldEmpty(int nIndex);    */
//////////////////////////////////////////////////////////////////////
//判斷位置
BOOL IsEOF();
BOOL IsBOF();
//////////////////////////////////////////////////////////////////////
//遍歷記錄
BOOL MoveFirst();
BOOL MoveNext();
BOOL MovePrevious();
BOOL MoveLast();
//////////////////////////////////////////////////////////////////////
//搜索記錄，獲得當前記錄的位置序號
BOOL Find(LPCTSTR lpFind, int nSearchDirection);
long GetAbsolutePosition();
/*          void SetAbsolutePosition(int nPosition)
            {m_pRecordset->PutAbsolutePosition((enum PositionEnum)nPosition);};
      */
//////////////////////////////////////////////////////////////////////
//其他
CString GetQuery()
{return m_strQuery;};
void SetQuery(LPCSTR strQuery)
{m_strQuery = strQuery;};
//////////////////////////////////////////////////////////////////////
//書簽操作
void GetBookmark();
BOOL SetBookmark();
//////////////////////////////////////////////////////////////////////
//錯誤處理
CString GetLastError() {return m_strLastError;};
protected:
void dump_com_error(_com_error &e);

protected:
_ConnectionPtr m_pConnection;
_RecordsetPtr m_pRecordset;
```

```
int m_nSearchDirection;
CString m_strFind;
//      _variant_t m_varBookFind;
_variant_t m_varBookmark;
CString m_strLastError;
CString m_strQuery;
};
```

详细说明：使用请参考工程中 “Database”目录下的数据库表代码文件。

5. UDP Heartbeat

该功能是一个任务，主要来检测设备在线状态。具体代码见工程中 HeartBeatTask.h 和 HeartBeatTask.cpp 文件中，使用 udp 端口为 49201。



任务架构

规则：

3s 钟发送一个检测包，收到回复了就停止发送，最多发送 3 次，如果三次都没收到终端设备的回复，那么就认定该设备已经离线。

代码见“HeartbeatTask.h”中：

```
#pragma once
#include <vector>
#include <list>
#include <map>
using namespace std;
```

```
#define UDP_HEARTBEAT_ON_LINE  1  /*on line*/
#define UDP_HEARTBEAT_OFF_LINE 0  /*off line*/
class CHeartBitTask
{
public:
	CHeartBitTask(void);
	~CHeartBitTask(void);
public:
	typedef struct _HeartBitInfo
	{
		string ip;
		int count;

	}HeartBitInfo;

	static DWORD __stdcall ThreadSendProc(LPVOID lparam);
	static DWORD __stdcall ThreadRecvProc(LPVOID lparam);

	list<HeartBitInfo> m_vSendList;
	map<string,HeartBitInfo> m_vRecvList;
	list<HeartBitInfo> m_vAllList;

	bool m_bSendAuto;
	bool m_bRecvAuto;
	SOCKET local;

private:
	HANDLE m_hSendThread;
	HANDLE m_hRecvThread;


	typedef void (__stdcall *CallBackFunc)(void *,int *,char *);
	CallBackFunc m_pCallBackFunc;
	void *m_Object;
```

```
public:
    int  InitUdpSocket(int nPort);


public:
template <class ClassName> void RegRecvFunc(void *pObject, void (__stdcall ClassName::*pFunc)(int ,char
*));
    /*初始化任務發送 List*/
    void InitDeviceList(vector<string> vip_address);
    void BeginTask();
    void EndTask();
};
```

用户使用的时候只需要调用一下 4 个接口即可，如下：

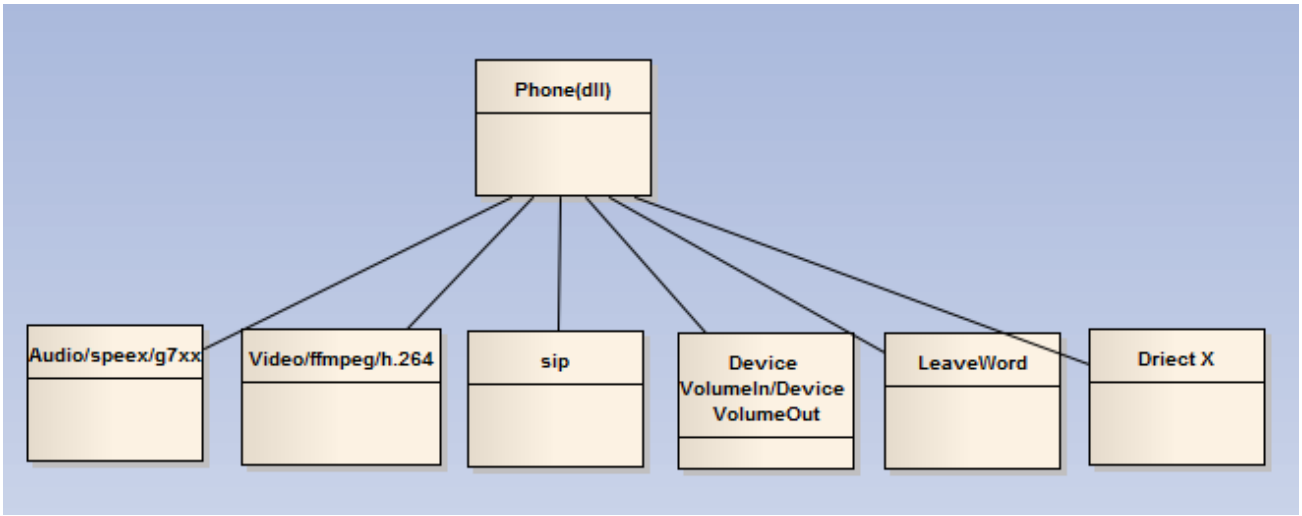
```
/*
功能：模版函数，注册回调函数，返回设备在线消息
函数名：RegRecvFunc
参数：void * pObject： 回调函数对象指针
      void (__stdcall ClassName::*pFunc)(int ,char *):回调函数
返回值：void
说明：只有调用此函数注册了，才能返回设备状态消息
*/
template <class ClassName> void RegRecvFunc(void *pObject, void (__stdcall ClassName::*pFunc)(int ,char
*));
/*
功能：初始化任務發送 List
函数名：InitDeviceList
参数：vector<string> vip_address:设备 ip list
返回值：void
说明：
*/
    void InitDeviceList(vector<string> vip_address);
/*启动任务*/
```

```
    void BeginTask();
/*结束任务*/
    void EndTask();
```

备注：启动任务前要先 RegRecvFunc 注册回调函数，然后 RegRecvFunc 启动任务，并 InitDeviceList 初始化系统设备表，当结束任务时候，请调用 EndTask 释放所有任务资源。

6. Phone (SolutionDLL.dll)

该层模块集成了 Video /Audio 解码部分、Sip 通信部分、Direct X、留影留言处理 Module 、Device Volume In/Device Volume Out 如下图所示：



模块图（Phone.dll)

a、Video/Audio 采用最新的 fmpeg 库，audio 引入了 Speex 和 g7xx 库。

b、直接调用 Phone.dll 接口可以完成基本的 Sip 动作：sip 呼叫，sip 呼叫停止、sip 监视，sip 停止监视

c、在 Phone.dll 中已经建立了一个视频 Show Task，sip 处理->视频接受->视频显示 都是在 dll 中去处理，只需要在调用初始化 dll 的时候，将显示视频的窗口的句柄（hwnd 或者 Cwnd *)传入即可。

solutionDll 接口说明：

```
/*
功能：初始化对讲功能
函数名：dll_init
参数：TUICommand::phoneCallBack func  回调函数
返回：0：初始化成功  -1:初始化失败
说明：所有的消息都是通过回调函数返回， 不要在回调函数里面添加阻塞事件。
typedef  void (WINAPI * phoneCallBack)(int AType ,const mm_comm::TDataBase&, void*);

    返回消息消息包括：
typedef enum {
          emMEET_INVAL        = 0,      //标括?志?无T效§
         /*meet operator */
          emMEET_REQUEST      = 1,      //请?求ó对?讲2
          emMEET_ACCEPT       = 2,      //接ó受酣?对?讲2
          emMEET_REFUSE       = 3,      //拒ú绝?对?讲2
          emMEET_TIMEOVER          = 4,      //Request Time Over
          emMEET_STOP              = 5,      //挂ò断?对?讲2
          emMEET_OPERA        = 6,      //辅¨助ú操ù作痢?
          emMEET_FAILED       = 7,      //操ù作痢?失骸?败悃?

         /*watch operator*/
          emWATCH_REQUEST          = 8,      //请?求ó监à视酣?
          emWATCH_BUSY        = 9,      //监à视酣?正y忙|
          emWATCH_TIMEOVER    = 10,          //监à视酣?超?时骸?
          emWATCH_STOP        = 11,          //停  ?止1监à视酣?
          emWATCH_ACCEPT      = 12,          //接ó受酣?监à视酣?

          /*leave word operator*/
          emLW_STOP           = 14,
          emLW_REQUEST        = 15,
          emLW_DOWNLOAD_REQUEST = 16,
         /*Phone Records upload*/
          emPHONERECORDS_FROM = 17,
```

```
        emPHONERECORDS_TO    = 18

    } TMeetDataFlag;

*/
extern "C" __declspec(dllexport) int __stdcall dll_init(TUICommand::phoneCallBack func)
```

Sip 动作提供了一个 dll_Dataout 接口，flag 参数包含了 sip 所有的动作，如下所见。

```
/*
功能：提供 sip 呼叫接口
函数名：dll_Dataout
参数：const mm_comm::TDataBase& data   包含设备信息
      void* pt=0  预留，暂无实际功能
返回值：true or false
说明：data.mValue.flag: 操作标记
                        = emMEET_REQUEST        //请求对讲
                        = emMEET_ACCEPT         //接受对讲
                        = emMEET_REFUSE         //拒绝对讲
                        = emMEET_STOP       //挂断对讲
                        = emWATCH_REQUEST   //请求监视
                        = emWATCH_STOP      //停止监视
      data.mValue.device.HouseNo :房间号
      data.mValue.device.IP     :设备 Ip
    传参只需要关注上面结构体字段即可。
*/
extern "C" __declspec(dllexport) bool __stdcall dll_DataOut(const mm_comm::TDataBase& data, void* pt= 0)


/*
void Handle(HWND); 绑定视频输出窗口的句柄

//音量控制操作函数
/*
功 能：设置麦克风声音
函数名：SetVolumeImport
参数类型: DWORD  输入音量值
```

返回值：void

*/

void SetVolumeImport(DWORD);

/*

功能：设置输出音量

函数名：SetVolumeExport

参数类型：DWORD 输出音量值

返回值：void

*/

void SetVolumeExport(DWORD);

/*

功能：获取当前输入音量的值

函数名：GetVolumeImport

参数类型：void

返回值：DWORD 返回当前输入音量值

*/

DWORD GetVolumeImport();

/*

功能：获取当前输出音量值

函数名：GetVolumeExport

参数：void

返回值：DWORD 返回当前输出音量值

*/

DWORD GetVolumeExport();

/*

功能：开启音频设备 当对讲有语音输入输出是需要调用该接口

函数名：OpenVoiceDev

参数：void

返回值：bool 返回 true：开启成功 false 开启失败

/*

功能：获取当前对讲的通话状态

函数名：dll_CallState

参数：NULL

返回值：nsMM::TMeetStateFlag 当前通话状态的值

```
说明: typedef enum {
            emMEET_STATE_FREE,              //空闲状态
            emMEET_STATE_REQUESTING,        //正在呼叫
            emMEET_STATE_CALLING,           //正在通话
      } TMeetStateFlag;
*/
extern "C" __declspec(dllexport) nsMM::TMeetStateFlag __stdcall dll_CallState()

/*
功能：获取当前的通话类型
函数名：dll_CallType
参数：NULL
返回值：nsMM::TCallType  呼叫类型
说明：typedef enum {
                  emNOCALL,//无通话状态
                  emTOCALL,//被呼叫状态
                  emFORCALL,//主动呼叫状态
                  emWATCH //监视状态
                  }     TCallType;

*/
extern "C" __declspec(dllexport) nsMM::TPhone::TCallType __stdcall dll_CallType()
/*
功能：监视状态返回
函数名：dll_WatchState
参数：NULL
返回值：TWatchStateFlag  监视状态
说明：      typedef enum {
            emWATCH_STATE_FREE,             //空闲状态
            emWATCH_STATE_BUSY,             //正在监视
      } TWatchStateFlag;ype;

*/
extern "C" __declspec(dllexport) nsMM::TWatchStateFlag__stdcall dll_WatchState()
/*play record file*/
```

7. Sip

SIP(Session Initiation Protocol)是一个应用层的信令控制协议。用于创建、修改和释放一个或多个参与者的会话。本系统中采用 SIP 方式来，协商会话，建立通话通道。Sip 控制协议具体参见标准的 SIP 资料，系统中使用的 SIP Protocol 详细见 Protocol 协议文档。

下面就外置的主要几个 sip 封装接口做一下说明（详细参见项目解决法中 sip.h 文件）：

```
/*
功能：sip 呼叫请求函数
函数名：SIP_CallStart
参数 char * 终端设备的ip地址
返回值：int -1操作失败， 0操作成功
说明：底层调用osip exsip等相关操作接口。
*/
int SIP_CallStart(char *pcCallNum);

/*
功能：sip 停止呼叫
函数名：SIP_CallStop
参数：NULL
返回值：int
说明：当处于主动呼叫or被动呼叫状态，调用此接口可终端呼叫。
*/
int SIP_CallStop();
/*
功能：接听当前呼叫
函数名：SIP_CallAccept
参数：NULL
返回值：int
说明：
*/
int SIP_CallAccept();
/*
```

```
功能：取消当前呼叫
函数名：SIP_CallCancel
参数：NULL
返回值：int
说明：
*/
int  SIP_CallCancel();
/*
功能：监视设备
函数名：SIP_WatchStart
参数：char * 监视设备的IP
返回值：int
说明：
*/
int  SIP_WatchStart(char * pcWatchedHost);

/*
功能：停止监视
函数名：SIP_WatchStop
参数：bool
返回值：int
说明：
*/
int  SIP_WatchStop (bool fEndMsg = true);
/*
功能：接听留影留言
函数名：sip_handle_accept_lw
参数：char * cip    :接听留影留言设备ip
    int nVideoPort :视频协商端口
    int nAudioPort : 音频协商端口
返回值：int
说明：
*/
```

```
int  sip_handle_accept_lw(char * cip,int nVideoPort, int nAudioPort);
/*
功能：设置sip忙状态
函数名：set_sip_dnd
参数：bool  bopen （true为置忙，false为置闲）
返回值：void
说明：
*/
void set_sip_dnd(bool bopen);
/*
功能：sip服务主程
函数名：HDDP_SIPServer
参数：LPVOID lpParam
返回值：DWORD
说明：
*/
DWORD WINAPI HDDP_SIPServer(LPVOID lpParam );
```

备注：启动sip服务时候，必须在子线程，运行HDDP_SIPServer(LPVOID lpParam)主程。

8. Local System Manage

用户登录：见代码。

用户数据库备份：见代码

系统参数设置：见代码

1.8 SDK demo 介绍

1、Database demo 调用可以在工程 Database 过滤夹中看见，只需要在此处包装后，调用相应接口，

即可操作数据库，完成相应的功能。

2、Phone Module Demo 见：

工程 Video_sipDemo.projx，部分重要代码如下：

```
void CVideo_sipDemoDlg::OnBnClickedOk()//nsMM::emMEET_REQUEST 是sip呼叫请求。
{
    TDataType data;
    data.mValue.flag = nsMM::emMEET_REQUEST;
    memset(data.mValue.device.IP.addr_ss,0,16);
    memcpy(data.mValue.device.IP.addr_ss ,"192.168.191.220",sizeof("192.168.191.220"));
    data.mValue.device.IP.type = mm_comm::TIPAddress::emString;
    data.mValue.device.HouseNo = "2222";
    m_VideoInnet.DataOut(data);
}


void CVideo_sipDemoDlg::OnBnClickedButton1()//nsMM::emMEET_ACCEPT 接受呼叫请求命令
{
    CString sIp;
    m_IpAddress.GetWindowText(sIp);
    TDataType data;
    data.mValue.flag = nsMM::emMEET_ACCEPT;
    memset(data.mValue.device.IP.addr_ss,0,16);
    memcpy(data.mValue.device.IP.addr_ss ,sIp.GetBuffer(), sIp.GetLength());
    data.mValue.device.IP.type = mm_comm::TIPAddress::emString;
    data.mValue.device.HouseNo = "222";
    m_VideoInnet.DataOut(data);
}


void CVideo_sipDemoDlg::OnBnClickedButton2()
{
    TDataType data;
```

```
	CString sIp;
	m_IpAddress.GetWindowText(sIp);
	data.mValue.flag = nsMM::emMEET_STOP;
	memset(data.mValue.device.IP.addr_ss,0,16);
	memcpy(data.mValue.device.IP.addr_ss ,sIp.GetBuffer(), sIp.GetLength());
	data.mValue.device.IP.type = mm_comm::TIPAddress::emString;
	data.mValue.device.HouseNo = "01001010010200";
	m_VideoInnet.DataOut(data);
}


void CVideo_sipDemoDlg::OnBnClickedButton3()
{
	TDataType data;
	CString sIp = "192.168.191.199";
	//m_IpAddress.GetWindowText(sIp);
	data.mValue.flag = nsMM::emWATCH_REQUEST;
	memset(data.mValue.device.IP.addr_ss,0,16);
	memcpy(data.mValue.device.IP.addr_ss ,sIp.GetBuffer(), sIp.GetLength());
	data.mValue.device.IP.type = mm_comm::TIPAddress::emString;
	data.mValue.device.HouseNo = "01001010010200";
	m_VideoInnet.DataOut(data);
}
```