

# 卓岚 WinP2p 和设备 管理开发库

嵌入式设备联网解决方案

版权©2008 上海卓岚信息科技有限公司保留所有权力

ZL DUI 20151003.1.0



版权©2008 上海卓岚信息科技有限公司保留所有权力

**版本信息**

对该文档有如下的修改：

修改记录			
日期	版本	文档编号	修改内容
2015-10-03	Rev. 1	ZL DUI 20151003. 1. 0	发布版本
2017-5-22	Rev. 2	ZL DUI 20151003. 2. 0	增加功能选择和多主机

**所有权信息**

未经版权所有者同意，不得将本文档的全部或者部分以纸面或者电子文档的形式重新发布。

本文档只用于辅助读者使用产品，上海卓岚公司不对使用该文档中的信息而引起的损失或者错误负责。本文档描述的产品和文本正在不断地开发和完善中。上海卓岚信息科技有限公司有权利在未通知用户的情况下修改本文档。

# 目 录

1. 概述 .....	4
1. 函数库描述 .....	4
1.1. 概述 .....	4
1.2. 头文件 .....	4
1.3. 设备管理函数 .....	12
1.4. P2P 函数 .....	16
2. 使用步骤 .....	20
2.1. ZLUseDevManage 功能.....	20
2.2. 加载函数库 .....	22
2.3. 初始化函数库 .....	24
2.4. 退出函数库 .....	24
2.5. 搜索设备 .....	25
2.6. 参数操作 .....	25
3. 其它开发环境 .....	25
1.1. NET 环境下 .....	26
2. 售后服务和技术支持 .....	28

## 1. 概述

“卓岚 WinP2p 和设备管理开发库”简称 ZLDM，是一个 DLL 动态连接库。主要包含有两部分功能：

1. WinP2p: 这个是 P2P 在 windows 上开发的二次开发库，可以实现 id 的添加、删除等功能，从而能够实现 P2P 功能集成到用户程序中。
2. 设备管理：集成对局域网设备的搜索、参数获取、参数修改。后续还将支持手动添加设备、外网设备的搜索、P2p 设备搜索。

ZLDM 库可以被 C++、C#、VB、Delphi 等编译语言调用。为需要集成卓岚 P2p 和卓岚设备搜索的用户减轻了开发工作量。如果需要和卓岚设备进行数据通信请直接使用用户开发环境的 socket 编程实现，这是任何开发平台都带的功能，ZLDM 库将不再提供数据通信功能。

## 1. 函数库描述

### 1.1. 概述

ZLDM 库包含有 ZLDevManage.dll 和 ZLDevManageD11.h 两个主要文件。另外还有 ZLUseDevManage 这个 VC 环境下的使用库的例程。

新的 ZLDM 库（V1.40）版本更加简练，采用返回值获得所要的数据。返回值只有 int 和 string 两种类型。

### 1.2. 头文件

ZLDevManageD11.h 是 C 语言格式，包含宏定义和 dll 库函数定义。

### 其它宏定义

```
#define ZLDM_VER "V1.40" // 库版本号
#define ID_LEN 6 // hex 型的设备 id 的长度
#define ID_STRING_LEN (ID_LEN*2) // 字符串型的设备 id 的长度
#define ZLDM_HANDLER_ARRAY_MIN_SIZE 256 // 定义的设备总数，可以被 256 更大
```

## 参数类型

#define PARAM_SET_TO_DEFAULT	0
#define PARAM_DEV_EXIST_IN_SUBNET	1
#define PARAM_DEV_LOCAL_IP	2
#define PARAM_DEV_LOCAL_PORT	3
#define PARAM_DEV_OUTER_IP	4
#define PARAM_ALL_SUBNET_ID	5
#define PARAM_SIMULATE_PORT	6
#define PARAM_P2P_STATUS_CODE	7
#define PARAM_P2P_STATUS_CHS	8
#define PARAM_P2P_STATUS_EN	9
#define PARAM_WORK_MODE	256
#define PARAM_IP_MODE	257
#define PARAM_FLOW_CONTROL	258
#define PARAM_DEST_MODE	259
#define PARAM_APP_PROTOCOL	260
#define PARAM_STOP_BIT	261
#define PARAM_PARITY	262
#define PARAM_DATA_BITS	263
#define PARAM_BAUDRATE	264
#define PARAM_DNS_SERVER_IP	265
#define PARAM_UDP_GROUP_IP	266
#define PARAM_NET_MASK	268
#define PARAM_GATEWAY	269
#define PARAM_RECONNECT_TIME	270
#define PARAM_KEEP_ALIVE_TIME	271
#define PARAM_WEB_PORT	272
#define PARAM_DEV_VER	273
#define PARAM_DEST_PORT	275
#define PARAM_DEV_NAME	276
#define PARAM_DEST_IP	277
#define PARAM_GAP_TIME	278
#define PARAM_PACKING_LEN	279
#define PARAM_LINK_STATUS	280
#define PARAM_RESOURCE_IP	281
#define PARAM_RESOURCE_PORT	282
#define PARAM_485_TIME_OUT	283
#define PARAM_485_GAP	284
#define PARAM_RECOMMAND_485_TIME_OUT	285
#define PARAM_RECOMMAND_485_GAP	286
#define PARAM_RECOMMAND_GAP_TIME	287
#define PARAM_FUNC_MODBUS	200
#define PARAM_FUNC_P2P	201

```

#define PARAM_ENABLE_P2P                220
#define PARAM_ENABLE_SEND_MAC            221
#define PARAM_ENABLE_TCP_NEED_KEY        222
#define PARAM_ENABLE_UDP_MODIFY_NEED_KEY 223
#define PARAM_WIFI_SSID                  350
#define PARAM_WIFI_CHANNEL                351
#define PARAM_WIFI_KEY                   352
#define PARAM_WIFI_KEY_TYPE              353
#define PARAM_WIFI_STA_AP                354
#define PARAM_WIFI_DHCP_SERVER           355
#define PARAM_WIFI_ETH_BRIDGE            356

```

这些定义都是获得、设置参数时的参数类型定义，列出表格如下，对表格说明如下：

- int：则使用 ZLDM\_SetDevParamInt 和 ZLDM\_GetDevParamInt
- string：则使用 ZLDM\_SetDevParamString 和 ZLDM\_GetDevParamString
- 读取：支持 ZLDM\_GetDevParamInt 或者 ZLDM\_GetDevParamString
- 设置/写入：支持 ZLDM\_SetDevParamInt 或者 ZLDM\_SetDevParamString
- 说明：对参数功能的说明
- 参数 2：ZLDM\_SetDevParamInt 或者 ZLDM\_SetDevParamString 的第二个参数
- 返回值：string 就是需要的字符串，int 的返回值如果为-1 表示错误，其它值的含义见下表。注意设置值是根据在下拉框中的位置设计的，方便和下拉框位置对应。比如停止位 1 返回值为 0，停止位为 2 范围值为 1。这是容易搞混的地方，请引起注意。

表 1. 参数类型

宏定义	类型	读 / 写	参数 2	返回值	说明
PARAM_SET_TO_DEFAULT	int	写	任意	/	将设备所有参数恢复为默认参数
PARAM_DEV_EXIST_IN_SUBNET	int	读	/	-1: 未找到 0: 不是内网 1: 内网 2: 内网但 IP 不是一个网段	设备是否在内网中
PARAM_DEV_LOCAL_IP	string	读	/	ip 字符串	获得设备 ip
PARAM_DEV_LOCAL_PORT	int	读	/	端口	获得设备端口
PARAM_DEV_OUT	string	读	/	ip 字符串	p2p 专用，获得设备

ER_IP	ng				外网 ip
PARAM_ALL_SUBNET_ID	string	读	/	以逗号分隔, 句号结尾。例如 " 284E01020304, 284E12345678. "	p2p 专用, 返回局域网中的所有设备 id
PARAM_SIMULATE_PORT	int	读	/	模拟端口	p2p 专用, 通过 id 获得模拟端口
PARAM_P2P_STATUS_CODE	int	读	/	状态码	p2p 专用, 获得 p2p 状态码。读取状态码可以程序检测是否处于有效连接状态。状态码见 "P2P 状态" 部分。
PARAM_P2P_STATUS_CHS	string	读	/	描述字符串	p2p 专用, 获得 p2p 的中文描述字符串
PARAM_P2P_STATUS_EN	string	读	/	描述字符串	p2p 专用, 获得 p2p 的英文描述字符串
PARAM_WORK_MODE	int	读 / 写	工作模式	0: TCP 服务器 1: TCP 客户端 2: UDP 3: UDP 组播	工作模式
PARAM_IP_MODE	int	读 / 写	方式	0: 静态 1: 动态	IP 自动获取方式, 静态或者 DHCP 模式
PARAM_FLOW_CONTROL	int	读 / 写	流控	0: 无流控 1: CTS/RTS 2: DTR/DSR 3: XON/XOFF	串口流控方式
PARAM_DEST_MODE	int	读 / 写	目的模式	0: 静态 1: 动态	目的模式 (特殊用途, 一般无需设置), 默认为动态
PARAM_APP_PROTOCOL	int	读 / 写	转化协议	0: 透明传输 1: Modbus TCP 转 RTU 2: Realcom 模式	转化协议模式
PARAM_STOP_BIT	int	读 / 写	停止位	0: 停止位 1 1: 停止位 2	串口参数停止位
PARAM_PARITY	int	读 / 写	校验位	0: 无校验 1: 奇校验 2: 偶校验 3: 标记 4: 空格	串口参数校验位
PARAM_DATA_BITS	int	读 / 写	数据位	0: 8 位 1: 7 位	串口参数数据位

				2: 6 位 3: 5 位	
PARAM_BAUNDRA TE	int	读 / 写	波特率	0: 1200 1: 2400 2: 4800 3: 7200 4: 9600 5: 14400 6: 19200 7: 28800 8: 38400 9: 57600 10: 76800 11: 115200 12: 240800 13: 460800	串口参数波特率
PARAM_DNS_SER VER_IP	stri ng	读 / 写	IP	IP 字符串	域名服务器 IP
PARAM_UDP_GRO UP_IP	stri ng	读 / 写	IP	IP 字符串	UDP 组播所在组地址
PARAM_NET_MAS K	stri ng	读 / 写	IP	IP 字符串	子网掩码
PARAM_GATEWAY	stri ng	读 / 写	IP	IP 字符串	网关
PARAM_RECONNE CT_TIME	int	读 / 写	时间	重连时间单 位为秒, 范围 0~255	断线重连时间
PARAM_KEEP_AL IVE_TIME	int	读 / 写	时间	保活定时时 间, 单位 为 秒, 范围 0~ 255	保活定制或者心跳包 时间
PARAM_WEB_POR T	int	读 / 写	端口	网页端口, 范 围 1~65535	内嵌的配置网页的端 口号
PARAM_DEV_VER	stri ng	读	/	版本号	读取设备固件版本号 字符串
PARAM_DEST_PO RT	int	读 / 写	端口	端 口 , 范 围 1~65535	客户端的目的端口
PARAM_DEV_NAM E	stri ng	读 / 写	名称	名称字符串, 长度小于等 于 9 字节	设备名称, 便于记忆 的名称
PARAM_DEST_IP	stri ng	读 / 写	目的	目 的 IP/ 域 名, 小于 30 字节	客户端的目的 IP 或 者域名
PARAM_GAP_TIM E	int	读 / 写	间隔	间隔, 单位是 毫 秒 , 范 围	分包规则中的时间间 隔



				0~255	
PARAM_RECOMM ND_GAP_TIME	int	读	/	1 字节的数据包间隔推荐值, 范围 0~255	获得 PARAM_GAP_TIME 的最小值, 不过设置值应该比推荐值大 2 左右是比较合理的。
PARAM_PACKING _LEN	int	读 / 写	数据包大小	大小, 单位是字节, 范围 1~1400	分包规则中的打包大小
PARAM_LINK_ST ATUS	int	写	状态	0: 未连接 1: 至少存在一个 TCP 连接	读取设备的 TCP 连接状态
PARAM_RESOURC E_IP	stri ng	读	/	IP 字符串	获得设备参数的来路 IP
PARAM_RESOURC E_PORT	int	读	/	2 字节的 UDP 端口号	获得设备参数的来路端口
PARAM_485_TIM E_OUT	int	读 / 写	超时时间	2 字节的超时时间, 范围 0~8191, 注意设置值的分辨率是 32。	在 Modbus TCP 转化协议方式下, 这个参数是必须设置的。它表示当发送的 Modbus RTU 指令没有返回时, 最长等待多长时间。单位为毫秒。使用 PARAM_RECOMMAND_485_TIME_OUT 可以获得推荐值, 设置值应该大于等于推荐值。非 Modbus 方式下应该设置为 0。除非重新设置, 否则使用 PARAM_APP_PROTOCOL 在 modbus 和 none 之间切换时, 该值会自动变为推荐值或 0。
PARAM_RECOMM ND_485_TIME_O UT	int	读	/	2 字节推荐的超时时间	获得 PARAM_485_TIME_OUT 的推荐值, 方便用户设置。它会根据转化协议和波特率计算一个推荐值。
PARAM_485_GAP	int	读 / 写	485 半双工轮询间隔	1 字节, 范围是 0~255。	在 Modbus TCP 转化协议方式下, 这个参数是必须设置的。它表示相邻两次询问之间的空闲间隔。Modbus 方式下, 这个值应该

					大于 20 (ms)，可以用 PARAM_RECOMMAND_485_GAP 获得推荐值，设置值应大于等于推荐值。如果设置太大则轮训的频率会下降。非 Modbus 方式下应该设置为 0。除非重新设置，否则使用 PARAM_APP_PROTOCOL 在 modbus 和 none 之间切换时，该值会自动变为推荐值或 0。
PARAM_RECOMMAND_485_GAP	int	读	/	1 字节推荐的 485 半双工轮询间隔	获得 PARAM_485_GAP 的推荐值，方便用户设置。它会根据转化协议和波特率计算一个推荐值。
PARAM_FUNC_MODBUS	int	读	/	1 表示支持, 0 表示不支持	这个设备是否支持 Modbus 转 RTU 功能。
PARAM_FUNC_P2P	int	读	/	1 表示支持, 0 表示不支持	这个设备是否支持 P2P 功能。
PARAM_ENABLE_P2P	int	读 / 写	启用或者禁用	1 表示启用, 0 表示禁用	启用或者禁用 P2P 功能
PARAM_ENABLE_SEND_MAC	int	读 / 写	启用或者禁用	1 表示启用, 0 表示禁用	启用或者禁用 TCP 连接建立的时候发送设备的 6 字节 MAC 地址给上位机的功能
PARAM_ENABLE_TCP_NEED_KEY	int	读 / 写	启用或者禁用	1 表示启用, 0 表示禁用	启用或者禁用 TCP 连接建立的时候需要密码验证的功能
PARAM_ENABLE_UDP_MODIFY_NEED_KEY	int	读 / 写	启用或者禁用	1 表示启用, 0 表示禁用	启用或者禁用 TCP 连接建立的时候需要密码验证的功能
PARAM_WIFI_SSID	string	读 / 写	字符串 SSID	SSID	读写 wifi 的 SSID，作为 AP 的时候是自己的 SSID，作为 STA 的时候是连接的路由器的 SSID
PARAM_WIFI_KEY	string	读 / 写	字符串 KEY	KEY	读写 wifi 的 KEY
PARAM_WIFI_STA_AP	int	读 / 写	AP 或者 STA	0:AP 1:STA	设置或者读取 wifi 工作模式
PARAM_WIFI_CH	int	读 / 写	信道	0: 1 信道	设置 AP 方式下的

ANNE		写		1: 2 信道 2: 3 信道 3: 4 信道 ..... 10: 11 信道	wifi 信道
PARAM_WIFI_KEY_TYPE	int	读 / 写	加密类型	0: 无加密 1: WEB64 2: WEB128 3: TKIP 4: AES 5: 自动	一般只要选择无加密和自动两种即可，其它 1~4 一般不用选择。
PARAM_WIFI_DHCP_SERVER	int	读 / 写	DHCP 服务器是否启用	0: 禁用 1: 启用	是否起开 DHCP 服务器功能。建议默认关闭，防止影响网络中其它设备获得 IP。
PARAM_WIFI_ETH_BRIDGE	int	读 / 写	网口和 wifi 是否启用互通	0: 禁用 1: 启用	是否将 wifi 收到的数据转到网口，网口数据也转到 wifi，实现互通功能。默认建议关闭。

## P2P 状态

```

#define P2P_GET_STATE_UNKOWN      0      // 未知
#define P2P_GET_STATE_UNCONNECTED 1      // 和服务未连接
#define P2P_GET_STATE_PORT_ERR    2      // 输入的端口不存在
#define P2P_GET_STATE_SUBNET      3      // 内网
#define P2P_GET_STATE_INTERNET    4      // 外网
#define P2P_GET_STATE_PROXY       5      // 代理
#define P2P_GET_STATE_CONNECTING  6      // 连接中或者等待连接
#define P2P_GET_STATE_WAIT_CON    7      // 等待 p2p 连接到来
#define P2P_GET_STATE_SERVER      8      // 是服务器模式，做作为 pc 通过这个
变量可以知道是否调用了 p2popen 函数
#define P2P_GET_STATE_SERVER_AUTH_ERR 9    // pc 或者服务器未验证服务器的合法性
#define P2P_GET_STATE_DEV_OFF_LINE 10     // pc 连接的 dev 处于离线状态
#define P2P_GET_STATE_DEV_AUTH_ERR 11     // pc 连接的 dev 未被服务器验证通过
#define P2P_GET_STATE_PC_AUTH_ERR 12     // pc 未被服务器验证通过
#define P2P_GET_STATE_DEV_BELONG_ERR 13   // pc 希望连接的 dev 不属于这个用户所有
#define P2P_GET_STATE_SERVER_NO_ACK 14    // 发送请求给服务器，但是没有应答
#define P2P_GET_STATE_DEV_NO_ACK      15  // 发送请求后服务器有应答，设备没有
应答
#define P2P_GET_STATE_WAIT_PROXY      16  // 尝试通过代理方式连接

```

注意只有 3 种情况表示 P2P 是联通状态，可以通信的，分别为：

```

#define P2P_GET_STATE_SUBNET      3      // 内网

```

```
#define P2P_GET_STATE_INTERNET    4        // 外网
#define P2P_GET_STATE_PROXY       5        // 代理
```

## 函数指针类型定义

函数库中的函数指针类型定义如下，各个函数的含义将在后面讲解：

```
typedef int (__stdcall * tZLDM_Init)(int ServerPort);
typedef void (__stdcall * tZLDM_Exit)();
typedef char * (__stdcall * tZLDM_GetVer)();

typedef int (__stdcall * tZLDM_StartSearchDev)();
typedef char * (__stdcall * tZLDM_GetDevID)(int Dev_Index);

typedef int (__stdcall * tZLDM_AddManualDev)(const char ip_dns1[], const char ip_dns2[], int port);
typedef void (__stdcall * tZLDM_ClearManualDev)();

typedef char * (__stdcall * tZLDM_GetDevParamString)(const char *id, int ParamType);
typedef int (__stdcall * tZLDM_GetDevParamInt)(const char *id, int ParamType);
typedef int (__stdcall * tZLDM_SetDevParamString)(const char *id, const char *NewParam, int ParamType);
typedef int (__stdcall * tZLDM_SetDevParamInt)(const char *id, int NewParam, int ParamType);
typedef int (__stdcall * tZLDM_SetDevParamExcute)(const char *id);

/* p2p functions */
typedef bool (__stdcall * tZLDM_P2pOpen)(const char* svraddr, int svrport, const char* username, const char* userkey, int retry_time);
typedef bool (__stdcall * tZLDM_P2pClose)();
typedef int (__stdcall * tZLDM_P2pAddPort)(int simulate_port, const char* dev_id, const char *connect_to_ip, unsigned short connect_to_port);
typedef bool (__stdcall * tZLDM_P2pDelPort)(int simulate_port);
typedef void (__stdcall * tZLDM_P2pRestartAll)();
typedef bool (__stdcall * tZLDM_P2pRetartPort)(int simulate_port);
typedef char * (__stdcall * tZLDM_P2pGetID)(int ithP2pDev);
```

### 1.3. 设备管理函数

#### ZLDM\_Init

初始化函数库。无论是否使用 p2p 功能，使用前必须先调用该函数。

```
int __stdcall ZLDM_Init (  
int ServerPort  
);
```

### 参数

#### *ServerPort*

[in] 参数 *ServerPort* 用于在一个 udp 和 tcp 端口监听，这样一旦有设备作为客户端连接上这个端口，就可以被搜索到，实现外网设备的管理。另外一个作用是，在手动添加设备的时候，必须使用 *ServerPort != 0* 的参数来，否则将无法修改手动添加的设备的参数。其它情况可以设置该参数为 0。由于 4196 是 ZLVirCom 的默认监听端口，使用 4196 时请先关闭 ZLVirCom 程序，或者使用其它端口。

### 返回值

TRUE/FALSE，表示初始化是否成功。初始化失败的原因可能是 *ServerPort* 的端口已经被其他程序占用，例如 ZLVirCOM 正在运行时占用 4196 端口。

### ZLDM\_Exit

退出前调用该函数。该函数用于析构函数库内部变量，如果不调用该函数而退出，可能出现错误提示对话框。

```
void __stdcall ZLDM_Exit (  
);
```

### ZLDM\_GetVer

获取 ZLDevManage.dll 函数库的版本号。获取的版本号应该和 ZLDevManage.h 中的 ZLDM\_VER 相同，否则说明头文件和 DLL 库不匹配。即使在调用 ZLDM\_Init() 之前也可以先获取版本号。

```
int __stdcall ZLDM_GetVer(  
);
```

### 返回值

返回一个字符串型版本号。

## ZLDM\_StartSearchDev

搜索网络中所有联网的卓岚联网设备，并保存在内存中，以后随时可以用 ZLDM\_GetDevID() 获得设备列表。注意这个函数内部有等待，由于搜索需要等待设备的返回，内部等待时间约 500ms。

```
int __stdcall ZLDM_StartSearchDev();
```

返回值

返回搜索到的设备的总数。之后可以用 ZLDM\_GetDevID() 逐个获取其中的 ID。

## ZLDM\_GetDevID

调用 ZLDM\_StartSearchDev() 后使用这个函数获取第 i 个需要的设备的 id。这种机制使得用户无需为设备列表构建负责的链表结构进行保存。甚至也无需使用数组保存设备 id 列表。

```
char * __stdcall ZLDM_GetDevID(int Dev_Index);
```

参数

*Devindex*

[in] 获取第几个设备的 id，如果序号超出范围自动返回 NULL 返回值

返回值

第 Dev\_Index 个设备的 id，如果为 NULL 则这个序号的设备不存在。

## ZLDM\_GetDevParamString

获取某个设备的参数，参数的返回值是 string 类型的。

```
char * ZLDM_GetDevParamString(const char *id, int ParamType);
```

参数

*id*

[in] 字符串类型的设备的 id。

*ParamType*

[in] 获取的参数类型，是一个整型的代号，比如设备 IP 等。参考本文“参数类型”部分章节。

返回值

NULL 表示错误，否则是返回的所要的数据。

## ZLDM\_GetDevParamInt

获取某个设备的参数，参数的返回值是 int 类型的。

```
int ZLDM_GetDevParamInt(const char *id, int ParamType);
```

参数

*id*

[in] 字符串类型的设备的 id。

*ParamType*

[in] 获取的参数类型，是一个整型的代号，比如设备端口等。参考本文“参数类型”部分章节。

返回值

-1 表示错误，否则是返回的所要的数据。

## ZLDM\_SetDevParamString

设置某个设备的参数，参数是 string 类型的。所有参数设置完毕后，必须调用 ZLDM\_SetDevParamExcute 执行写入。

```
int ZLDM_GetDevParamString (const char *id, const char *NewParam, int ParamType);
```

参数

*id*

[in] 字符串类型的设备的 id。

*NewParam*

[in] 新的参数内容，是一个字符串。

*ParamType*

[in] 设置的参数类型，是一个整型的代号，比如设备 IP 等。参考本文“参数类型”部分章节。

返回值

0 表示错误，1 表示正确。

## ZLDM\_SetDevParamInt

设置某个设备的参数，参数是 int 类型的。所有参数设置完毕后，必须调用 ZLDM\_SetDevParamExcute 执行写入。

```
int ZLDM_SetDevParamInt(const char *id, int NewParam, int ParamType)
```

参数

*id*

[in] 字符串类型的设备的 id。

*NewParam*

[in] 新的参数内容，是一个 int 类型。

*ParamType*

[in] 设置的参数的类型，是一个整型的代号，比如设备 IP 等。参考本文“参数类型”部分章节。

返回值

0 表示错误，1 表示正确。

## ZLDM\_SetDevParamExcute

ZLDM\_SetDevParamString 和 ZLDM\_SetDevParamInt 调用后并没有将数据真正写入设备，必须调用 ZLDM\_SetDevParamExcute 执行写入。写入后设备会自动重启。

```
int ZLDM_SetDevParamExcute (const char *id)
```

参数

*id*

[in] 字符串类型的设备的 id。

返回值

0 表示错误，1 表示正确。

### 1. 4. P2P 函数

注意非 P2P 类型的函数库是不支持 P2P 函数的。支持 P2P 和不支持 P2P 的区别是：ZLDM\_VER 如果是“V1.40\_P2P”则是支持的，如果为“V1.40”则是不支持的。



### 1.1.1. 初始化

```
bool ZLDM_P2pOpen (const char* svraddr, int svrport, const char* username, const char* userkey, int retry_time);
```

功能：开启 p2p，必须在调用 ZLDM\_Init 之后调用该函数。

svraddr：服务器域名或者 ip

svrport：服务器端口，目前没有什么作用，留待后续扩展（一个服务器运行多个 P2P Server）

username：用户名，匿名登录传入 NULL

userkey：用户密码，匿名登录传入 NULL

retry\_time：p2p 连接重试的次数，取值建议大于 5 次，建议为 20 次。

返回值：ServerName 解析失败或者本地创建一个和服务器通信的 socke 失败，返回 FALSE

### 1.1.2. 释放和关闭

```
bool ZLDM_P2pClose ();
```

功能：关闭 p2p 和释放所有资源

返回值：目前都是 true，留待后续扩展

### 1.1.3. 增加 ID

```
int ZLDM_P2pAddPort (int simulate_port, const char* dev_id, const char *connect_to_ip, unsigned short connect_to_port);
```

功能：增加 id

simulate\_port：本地虚拟的端口，用户应该传进来一个随机的端口

dev\_id：ID 的字符串

connect\_to\_ip：写为 NULL，后续扩展用

connect\_to\_port：写为 0 即可，后续扩展用

返回值：FALSE：添加新的 id 失败。

### 1.1.4. 删除 ID

```
bool ZLDM_P2pDelPort (int simulate_port)
```

删除这个 port 对应的 id。

### 1.1.5. 获取状态

请使用前面讲述的 ZLDM\_GetDevParamString 和 ZLDM\_GetDevParamInt 获得状态。

其中的参数类型和 P2P 相关的有：

```
#define PARAM_DEV_EXIST_IN_SUBNET      1
#define PARAM_DEV_LOCAL_IP             2
#define PARAM_DEV_LOCAL_PORT           3
#define PARAM_DEV_OUTER_IP             4
#define PARAM_ALL_SUBNET_ID            5
#define PARAM_SIMULATE_PORT            6
#define PARAM_P2P_STATUS_CODE          7
#define PARAM_P2P_STATUS_CHS           8
#define PARAM_P2P_STATUS_EN            9
```

具体用法请参考“参数类型”部分的表格。其中这里主要用到 PARAM\_P2P\_STATUS\_CODE、PARAM\_P2P\_STATUS\_CHS。PARAM\_P2P\_STATUS\_CODE 的返回值参考“P2P 状态”部分的定义。

PARAM\_DEV\_EXIST\_IN\_SUBNET 主要作用是：如果一个设备处于局域网内部的，且外网无法联通的情况下，实际是可以直接通信的，但是如果采用 P2P 实际是无法通信的，因为 P2P 服务器无法连接通。为此用户在使用 P2P 之前先使用 GetDevParamInt(“284E00000001”,1) 查询设备是否为内网，如果为内网则可以使用以上函数的 PARAM\_DEV\_LOCAL\_IP 参数直接获得设备的 IP，PARAM\_DEV\_LOCAL\_PORT 参数获得端口，直接建立 TCP 连接。

PARAM\_DEV\_OUTER\_IP 的主要作用是：当 P2P 联通后，用户可以使用 GetDevParamString(“284E00000001”,2) 获得设备的外网 IP，这相当于实现了动态域名的功能。即只要一个网络中存在卓岚 P2P 设备，那么就可以实时获得这个网络的公网 IP。该功能需要首先联通这个设备的 P2P 连接。

PARAM\_ALL\_SUBNET\_ID 的作用是：搜索局域网中的所有的存在的设备，返回 id 列表。返回的 id 如果有多个中间用逗号隔开，末尾用句号结尾。这可以在第一次配置的时候，列出局域网中的设备的 id，节省输入 id 的麻烦。用 ZLDM\_StartSearchDev() 也可以实现类似的功能，但是需要和 ZLDM\_GetDevID 配

合使用。

### 1.1.6. 全部重连

```
void ZLDM_P2pRestartAll()
```

默认在网络恢复后需要 20 秒钟才会探测网络联通新，调用这个函数马上在网络连上后重连，减少等待时间。

### 1.1.7. 单个重连

```
bool ZLDM_P2pRetartPort(int simulate_port)
```

默认装下系统是 20 秒钟重连一下一个设备，如果一个设备上线后不想等待那么久，那么可以调用这个函数，让系统马上重连。

### 1.1.8. 获得设备

```
char const * ZLDM_P2pGetID(int ithP2pDev)
```

这个函数的作用是获得已经添加的第 ithP2pDev 个设备的 ID。使用这个函数可以让用户无需为添加的 p2p 设备保存一个复杂的链表结构，可以使用这个函数遍历查找所要的设备 id。

如果 ithP2pDev 不存在，那么将返回 NULL，否则返回 id 字符串。

### 1.1.9. 使能 P2P

```
(*m_pZLDM_SetDevParamInt)(g_pCurSelID, m_EnableP2p,  
PARAM_ENABLE_P2P);
```

使用参数 PARAM\_ENABLE\_P2P 可以启用一个设备 P2P 功能，具体参考本文 PARAM\_ENABLE\_P2P 相关部分。m\_EnableP2p=1 表示启用，0 表示禁用。

```
m_EnableP2p = (*m_pZLDM_GetDevParamInt)(g_pCurSelID,  
PARAM_ENABLE_P2P);
```

这个函数获得当前是否启用了 P2P 功能。

```
m_FuncSel1P2p = (*m_pZLDM_GetDevParamInt)(g_pCurSelID,  
PARAM_FUNC_P2P);
```

这个函数获得这个设备是否支持 P2P 功能。

## 2. 使用步骤

下面通过在 VC++上实现一个范例程序来讲解如何使用设备管理函数库。

### 2.1. ZLUseDevManage 功能

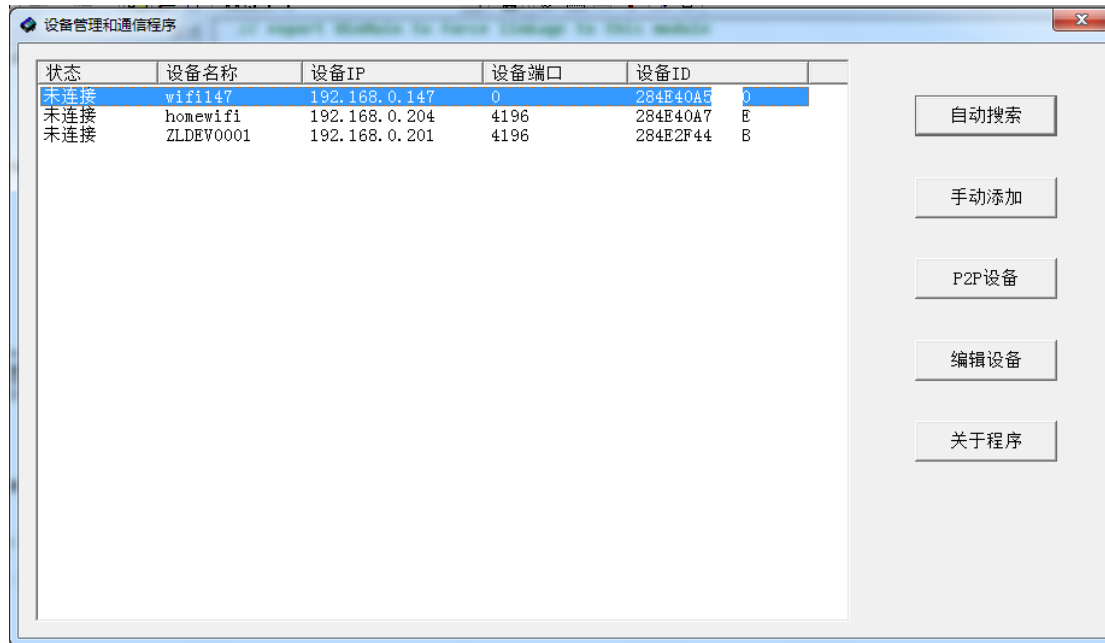


图 1 ZLUseDevManage 点击自动搜索后界面

主界面的自动搜索可以搜索网络中的所有卓岚设备。手动添加功能留待后续扩展使用。P2P 设备演示 P2P 的功能。编辑设备或者双击一行可以弹出设备参数的编辑对话框。



图 2 双击或者点击设备编辑后的界面

参数编辑对话框演示了参数的获取和写入功能。点击修改设置可以修改为所要的参数，点击默认参数可以恢复为出厂设置。

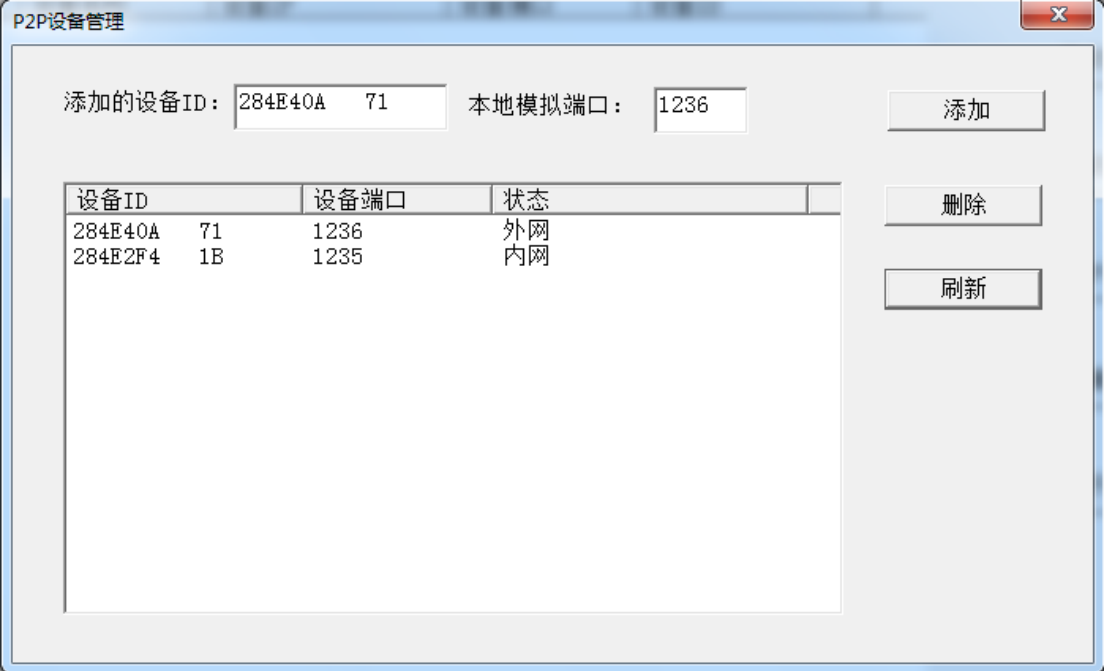


图 3 双击或者点击设备编辑后的界面

点击“P2P 设备”弹出 P2P 管理对话框。输入设备 id 和本地模拟端口后点击“添加”可以添加设备。选中一行之后点击“删除”可以删除 p2p 设备。如

果需要更新状态，则点击“刷新”按钮。

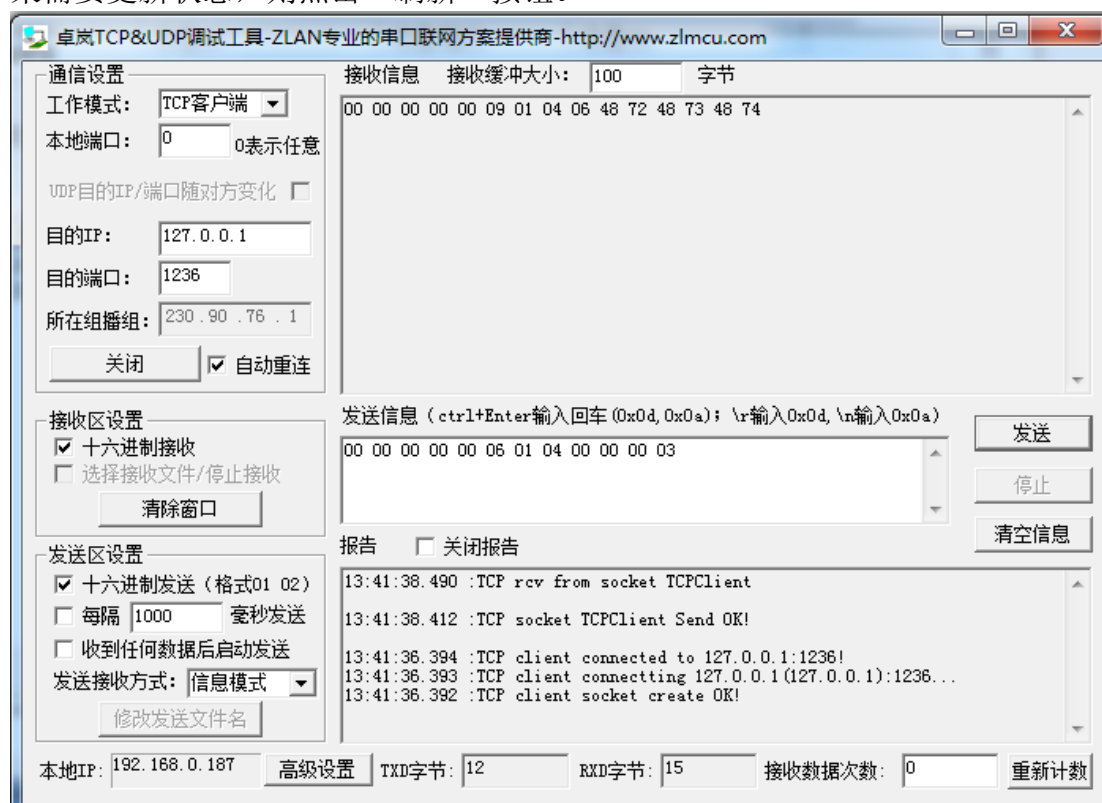


图 4 使用 SocketDlgTest 和 P2P 远程通信

使用 SocketTest 工具可以连接到本地的 1236 端口，这样就和远程的卓岚 P2P 串口建立通道。这里发送一个指令后，从卓岚 P2P 设备的串口输出，用户串口设备应答后被卓岚 P2P 设备串口接收，数据返回给 SocketTest。

以上的所有通信部分，包括远程 P2P 的 TCP 建立和本地 TCP 的建立，都建议使用 SocketTest 或者用户 socket 编程直接实现。

## 2.2. 加载函数库

使用函数库之前需要将 ZLDevManage.dll 加载到内存中。

在 VC++6.0 中，可以在任何初始化的地方加载函数库，这里选择在 OnInitDialog() 函数中加载。

1. 将 ZLDevManage.dll 拷贝到用户工程文件夹中。用户程序发布时，也需要将 ZLDevManage.dll 文件放在用户程序同一文件夹中。
2. 将 ZLDevManageDll.h 拷贝到用户工程文件夹中，使用  

```
#include "ZLDevManageDll.h"
```

包含函数库头文件。

3. 在 OnInitDialog() 函数中加载 DLL。

```
HINSTANCE m_hZLDevManage;  
m_hZLDevManage = LoadLibrary("ZLDevManage.dll");
```

4. 获取函数：为了方便后面程序的调用，初始化时可以先将函数库中的函数全部保存在全局函数指针中。

```
m_hZLDevManage = LoadLibrary("ZLDevManage.dll");  
m_pZLDM_Init      = (tZLDM_Init)GetProcAddress(m_hZLDevManage, "ZLDM_Init");  
m_pZLDM_Exit      = (tZLDM_Exit)GetProcAddress(m_hZLDevManage, "ZLDM_Exit");  
m_pZLDM_StartSearchDev  
(tZLDM_StartSearchDev)GetProcAddress(m_hZLDevManage, "ZLDM_StartSearchDev");  
m_pZLDM_GetDevID  
(tZLDM_GetDevID)GetProcAddress(m_hZLDevManage, "ZLDM_GetDevID");  
  
m_pZLDM_GetDevParamString  
(tZLDM_GetDevParamString)GetProcAddress(m_hZLDevManage,  
"ZLDM_GetDevParamString");  
m_pZLDM_GetDevParamInt  
(tZLDM_GetDevParamInt)GetProcAddress(m_hZLDevManage, "ZLDM_GetDevParamInt");  
m_pZLDM_SetDevParamString  
(tZLDM_SetDevParamString)GetProcAddress(m_hZLDevManage,  
"ZLDM_SetDevParamString");  
m_pZLDM_SetDevParamInt  
(tZLDM_SetDevParamInt)GetProcAddress(m_hZLDevManage, "ZLDM_SetDevParamInt");  
m_pZLDM_SetDevParamExcute  
(tZLDM_SetDevParamExcute)GetProcAddress(m_hZLDevManage,  
"ZLDM_SetDevParamExcute");  
m_pZLDM_GetVer  
(tZLDM_GetVer)GetProcAddress(m_hZLDevManage, "ZLDM_GetVer");  
m_pZLDM_AddManualDev  
(tZLDM_AddManualDev)GetProcAddress(m_hZLDevManage, "ZLDM_AddManualDev");  
m_pZLDM_ClearManualDev  
(tZLDM_ClearManualDev)GetProcAddress(m_hZLDevManage, "ZLDM_ClearManualDev");  
  
m_pZLDM_P2pOpen      = (tZLDM_P2pOpen)GetProcAddress(m_hZLDevManage,  
"ZLDM_P2pOpen");  
m_pZLDM_P2pClose     = (tZLDM_P2pClose)GetProcAddress(m_hZLDevManage,  
"ZLDM_P2pClose");  
m_pZLDM_P2pAddPort   = (tZLDM_P2pAddPort)GetProcAddress(m_hZLDevManage,  
"ZLDM_P2pAddPort");  
m_pZLDM_P2pDelPort   = (tZLDM_P2pDelPort)GetProcAddress(m_hZLDevManage,  
"ZLDM_P2pDelPort");  
m_pZLDM_P2pRestartAll = (tZLDM_P2pRestartAll)GetProcAddress(m_hZLDevManage,  
"ZLDM_P2pRestartAll");
```

```
m_pZLDM_P2pGetID = (tZLDM_P2pGetID)GetProcAddress(m_hZLDevManage,
"ZLDM_P2pGetID");
m_pZLDM_P2pRetartPort = (tZLDM_P2pRetartPort)GetProcAddress(m_hZLDevManage,
"ZLDM_P2pRetartPort");
```

其中各个函数指针的定义为:

```
tZLDM_Init          m_pZLDM_Init;
tZLDM_Exit          m_pZLDM_Exit;
tZLDM_StartSearchDev m_pZLDM_StartSearchDev;
tZLDM_GetDevID      m_pZLDM_GetDevID;
tZLDM_GetDevParamString m_pZLDM_GetDevParamString;
tZLDM_GetDevParamInt m_pZLDM_GetDevParamInt;
tZLDM_SetDevParamString m_pZLDM_SetDevParamString;
tZLDM_SetDevParamInt m_pZLDM_SetDevParamInt;
tZLDM_SetDevParamExcute m_pZLDM_SetDevParamExcute;
tZLDM_GetVer        m_pZLDM_GetVer;
tZLDM_AddManualDev m_pZLDM_AddManualDev;
tZLDM_ClearManualDev m_pZLDM_ClearManualDev;

tZLDM_P2pOpen m_pZLDM_P2pOpen;
tZLDM_P2pClose m_pZLDM_P2pClose;
tZLDM_P2pAddPort m_pZLDM_P2pAddPort;
tZLDM_P2pDelPort m_pZLDM_P2pDelPort;
tZLDM_P2pRestartAll m_pZLDM_P2pRestartAll;
tZLDM_P2pGetID m_pZLDM_P2pGetID;
tZLDM_P2pRetartPort m_pZLDM_P2pRetartPort;
```

之后库函数的调用可以通过函数指针调用, 例如调用版本号函数为:

```
(*m_pZLDM_GetVer)();
```

## 2.3. 初始化函数库

使用函数库前必须初始化函数库, 全局只需要进行一次初始化。

```
(*m_pZLDM_Init)(port);
```

初始化失败的原因可能是 *Port* 端口已经被其他程序占用, 例如 ZLVirCOM 正在运行时占用 4196 端口。建议 *Port*=0 就能够满足一般要求, 也不会产生冲突问题。

## 2.4. 退出函数库

在用户程序退出前, 一般需要先退出函数库。

```
(*m_pZLDM_Exit)();
```



## 2. 5. 搜索设备

使用 ZLDM\_StartSearchDev 可以搜索设备。

```
m_DevCnt = (*m_pZLDM_StartSearchDev)();

// 防止越界
if(m_DevCnt > ZLDM_HANDLER_ARRAY_MIN_SIZE)
    return;

// 逐个获得 id
for(i = 0; i < m_DevCnt; i++)
{
    m_h[i] = (*m_pZLDM_GetDevID)(i);
}
```

## 2. 6. 参数操作

可以如下方法读取某个设备的参数:

```
value = (*m_pZLDM_GetDevParamInt)(g_pCurSelID, PARAM_DEST_MODE);
m_DestIP = (*m_pZLDM_GetDevParamString)(g_pCurSelID, PARAM_DEST_IP);
```

如下方法进行参数的修改:

```
(*m_pZLDM_SetDevParamInt)(g_pCurSelID, m_DevWorkMode.GetCurSel(),
PARAM_WORK_MODE);
(*m_pZLDM_SetDevParamString)(g_pCurSelID, m_DnsServerIP,
PARAM_DNS_SERVER_IP);
(*m_pZLDM_SetDevParamInt)(g_pCurSelID, m_ReconnectTime,
PARAM_RECONNECT_TIME);
// 执行以上的写入的结果
(*m_pZLDM_SetDevParamExcute)(g_pCurSelID);
```

修改成功后设备将自动重新启动。

## 2. 7. P2P 使用步骤

使用 P2P 功能时请注意一下问题:

1. 必须首先在程序运行的时候调用库的初始化 ZLDM\_Init()。
2. 再使能 P2P 调用 (\*m\_pZLDM\_SetDevParamInt)(g\_pCurSelID, m\_EnableP2p, PARAM\_ENABLE\_P2P) 之前应该先调用设备搜索 ZLDM\_StartSearchDev()。只有确保这个 ID 所在设备能够搜索到才能进行配置。
3. 使用 P2P 的添加 ID 功能前, 必须先调用 ZLDM\_P2pOpen (const char\* svraddr,int svrport, const char\* username ,const char\* userkey, int retry\_time);函

数，其中的 svraddr 一般为：[www.i-zlan.com](http://www.i-zlan.com)，端口为：8000。这个函数运行后也不能马上添加 ID，因为需要在服务器上注册和验证信息。

4. 调用 ZLDM\_GetDevParamInt(id, PARAM\_P2P\_STATUS\_CODE) 获得当前的状态，如果是 P2P\_GET\_STATE\_SERVER 表示还没有成功调用 ZLDM\_P2pOpen；如果是 P2P\_GET\_STATE\_UNCONNECTED 表示和服务器没有连接，可能是本计算机处于断网状态；只有 P2P\_GET\_STATE\_SUBNET、P2P\_GET\_STATE\_INTERNET、P2P\_GET\_STATE\_PROXY 才算是处于 id 联通状态。

### 3. 其它开发环境

各种开发环境下如何调用设备管理函数库。

#### 1. 1. NET 环境下

##### 1. 1. 1. C#

1. 启动 VS.NET，新建一个项目，项目名称为“ZLDLLCall”，模板为“Windows 应用程序”。
2. 在“工具箱”的“Windows 窗体”项中双击“Button”项，向“Form1”窗体中添加一个按钮。
3. 改变按钮的属性：Name 为“B1”，Text 为“用 DllImport 调用设备管理函数库”，并将按钮 B1 调整到适当大小，移到适当位置。
4. 在类视图中双击“Form1”，打开“Form1.cs”代码视图，在“namespace ZLDLLCall”上面输入“using System.Runtime.InteropServices;”，以导入该命名空间。
5. 在“Form1.cs [设计]”视图中双击按钮 B1，在“B1\_Click”方法上面使用关键字 static 和 extern 声明方法“GetVer”，将 DllImport 属性附加到该方法，这里我们要使用的是“ZLDevManage.dll”中的“ZLDM\_GetVer”函数，具体代码如下：

```
[DllImport("ZLDevManage.dll", EntryPoint="
ZLDM_GetVer")]
static extern int GetVer ();
```

然后在“B1\_Click”方法体内添加如下代码，以调用方法“GetVer”：  
GetVer();

#### 注意：

返回字符串指针请转化为字符串用 Marshal.PtrToStringAnsi(GetDevID());如果 ptr 为 null, 则该方法返回空字符串。

例子：

加载：

```
[DllImport("ZLDevManager.dll", EntryPoint = "ZLDM_P2pOpen")]  
public static extern bool P2pOpen(string ip, int svrport, byte[] username
```

声明：

```
public static extern bool P2pOpen(string ip, int svrport, byte[] username, byte[] userkey, int r  
....
```

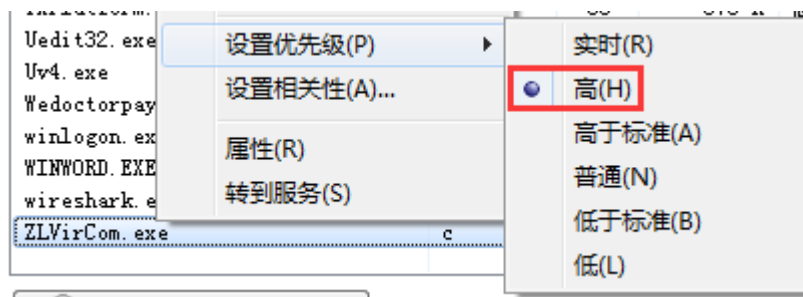
### 1.1.2. Delphi

调用的例子：

```
RetFunc:=ZLDM_P2pOpen(PChar('www.i-zlan.com'), 8000, PChar('zlp2psdk'), PChar('zlan'), 10);
```

请使用 Byte 类型的字符串，而不是 Pchar 类型的。

为了增加实时性，可以将 delphi 的实时性提高一下，类似 zlvircom。



另外请使用 dll 版本为 1.145 及以上版本的库。

### 1.1.3. C++

Visual C++ .net 2005 的使用方法如下：

1. 新建-----vc++-----Win32-----Win32 控制台应用程序（命名为：Win32\_MAIN）-----应用程序设置（应用程序类型选择控制台应用程序，同时选择预编译头）
2. 把 ZLDevManage.lib 与 ZLDevManage.dll 两个文件拷到 Win32\_MAIN 工程下（不是 Win32\_MAIN 解决方案下）
3. 在 Win32\_MAIN.cpp（控制台入口点：自动生成文件）中添加以下内容：

```
#include "stdafx.h"
#include <iostream>
using namespace std;
#pragma comment(lib, "ZLDevManage.lib")//添加
int ZLDM_GetVer ();//函数声明

int _tmain(int argc, _TCHAR* argv[])
{
    cout<< ZLDM_GetVer ()<<endl;//函数调用
    return 0;
}
```

## 2. 售后服务和技术支持

上海卓岚信息技术有限公司

地址：上海市徐汇区漕宝路 80 号光大会展 D 幢 12 层

电话：021-64325189

传真：021-64325200

网址：<http://www.zlmcu.com>

邮箱：[support@zlmcu.com](mailto:support@zlmcu.com)