VM301 软件开发手册

目录

1.简介

基于 VM301SDK 为用户提供一个简单、快速、高效开发 APK/IOS 等上层应用的平台。

本文介绍如何使用 SDK 库，以及 API 相关接口进行二次开发，主要的阅读对象为需要使用 VM301 进行上层应用开发的软件人员。

2.SDK 库 API 接口

2.1 初始化库

函数：	int ulink_init(const char *host, const char *appid);
参数：	host： 服务器域名或者 IP 地址（固定” p2p.vlinks.cn”） appid :代表本机唯一的 ID 码(MAC 地址) 16 字节 HEX 字符串
返回：	-1：表示初始化失败 0 ：表示初始化成功
描述：	在使用其他 API 接口函数之前运行此初始化库函数，需要注意的是，本函数只能在整个程序生命周期中，只运行一次。

用例：	**VC:**
	char ucMACString[18];
	PIP_ADAPTER_INFO pAdapterInfo;
	ULONG ulOutbufLen = sizeof(IP_ADAPTER_INFO);
	pAdapterInfo = (IP_ADAPTER_INFO *)malloc(sizeof(IP_ADAPTER_INFO));
	
	if (ERROR_BUFFER_OVERFLOW == GetAdaptersInfo(pAdapterInfo, &ulOutbufLen))
	{
	free(pAdapterInfo);
	pAdapterInfo = (IP_ADAPTER_INFO *) malloc (ulOutbufLen);
	}
	if (ERROR_SUCCESS == GetAdaptersInfo(pAdapterInfo, &ulOutbufLen))
	{
	sprintf((char*)&ucMACString[0],"0000%02x%02x%02x%02x%02x%02x",
	pAdapterInfo->Address[0],
	pAdapterInfo->Address[1],
	pAdapterInfo->Address[2],
	pAdapterInfo->Address[3],
	pAdapterInfo->Address[4],
	pAdapterInfo->Address[5]);
	
	}
	ulink_init("p2p.vlinks.cn", ucMACString);
	free(pAdapterInfo);

	Andorid:
	@Override
	protected void onCreate(Bundle savedInstanceState)
	{
	// 初始化库
	String up2pURL = "p2p.vlinks.cn";
	WifiManager mWifiManager = (WifiManager) getSystemService (Context.WIFI_SERVICE);
	WifiInfo info= mWifiManager.getConnectionInfo();
	String mac=info.getMacAddress();
	String appId ="0000"+mac.replace(":", "");
	UlinkNative.**ulinkInit**(up2pURL, appId);
	}
	
	注意：增加如下网络权限，否则获取不到 mac 地址信息
	<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
	<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
	<uses-permission android:name="android.permission.CHANGE_NETWORK_STATE"/>
	<uses-permission android:name="android.permission.INTERNET"/>
	
	IOS:

2.2 卸载库

函数：	int ulink_deinit();
参数：	
返回：	0 ：表示卸载库成功
描述：	在退出应用程序之前运行此函数

用例：	**VC:**
	ulink_deinit();
	
	Andorid:
	@Override
	protected void onDestroy()
	{
	super.onDestroy(); // 此处,特别注意,需要先销毁其他Activity,
	UlinkNative.**ulinkDeinit**(); // 最后才调用ulinkDeinit释放库资源
	}
	
	IOS:

2.3 配置 WIFI 信息

函数：	int smlink_start(const char *ssid, const char *password)
参数：	ssid: 需要配置 WIFI 的 SSID 名称 password： 需要配置 WIFI 的 PSW 密码

返回：	0 ：表示成功
描述：	本函数根据输入的 SSID 和 PSW 生成一系列规则包，并创建一个内部线程循环发送规则包(局域网发送)，每包发送间隔为 10ms，每发送完成一轮后延时 100ms 后继续发送； 注意：由于此函数发送数据量较大，请勿长期开启（调用 smlink_stop 停止发送）

用例：	**VC:**
	smlink_start(“vlinks_test”,”vlinks.cn”);
	
	Andorid:
	String sSSID="";
	sSSID=eSSID.getText().toString();
	String sPSW="";
	sPSW=ePSW.getText().toString();
	if (sSSID.length() != 0)// 当ssid不为空时，发送wfi配置信息
	{
	UlinkNative.**smlinkStart**(sSSID, sPSW);
	}
	
	IOS:

2.4 停止发送配置 WIFI 信息

函数：	int smlink_stop()
参数：	
返回：	0 ：表示成功
描述：	由于 smlink_start 函数发送数据量较大，请勿长期开启，调用 smlink_stop 停止发送。

用例：	**VC:**
	smlink_stop();
	
	Andorid:
	UlinkNative.**smlinkStop**(); // 停止 配置wifi信息
	
	IOS:

2.5 初始化扫描设备列表

函数：	int pid_list_init(void)
参数：	
返回：	0 ：表示成功
描述：	本函数执行初始化（清空）内部设备列表，为下一步执行 pid_list_read 做准备。 常规做法为：当需要扫描设备时，新建一个线程，线程初始化时执行 pid_list_init 初始化函数，线程内部循环（延时 1~3 秒）调用 pid_scan 和 pid_list_read 读取设备列表信息。

用例：

VC:

```
// 创建线程
CreateThread(NULL, 0, (LPTHREAD_START_ROUTINE) Thread_check_dev_list, this, 0, 0);

// 线程函数
void Thread_check_dev_list(void* param)
{
	int devlen,i;
	CVM301_MFCDlg* pThis;
	pThis = (CVM301_MFCDlg*)param;

	smlink_start(pThis->ssid,pThis->psw);

	pid_list_init();
	while (pThis->check_dev_stop_flag)
	{
		pid_scan();
		devlen = pid_list_read(pThis->devlist,200);
		// 显示设备列表
		for (i=0; i<devlen; i++)
		{
			//add dev
		}
		Sleep(2000);
	}
	smlink_stop();
}
```

Andorid:

```
//  详细参考openActivity.java
// 清空list
listItem.clear();
adapter.notifyDataSetChanged();
UlinkNative.pidListInit();

//  启动扫描设备定时任务
ScandBcakRun = true;
ScandBcakhandler.postDelayed(ScandBacktask, 1);

// 扫描设备定时任务,由于扫描函数非祖塞,所以能直接在UI线程中执行
private final Runnable ScandBacktask = new Runnable()
{
String sTmp1,sTmp2,sTmp3;
Integer iTmp;
@Override
public void run()
{
  // TODO Auto-generated method stub
  if ( ScandBcakRun )
  {
     listItem.clear();                  // 清空列表
     UlinkNative.pidScan();             // 发送扫描指令
     ArrayList<UlinkPidDev> mUlinkPidDevList = UlinkNative.readPidList(64);
     if ( mUlinkPidDevList != null )
     {
        for (int i = 0; i < mUlinkPidDevList.size(); i++)
        {
          map   = new HashMap<String, String>();
          sTmp1 = String.format("%08X%08X",mUlinkPidDevList.get(i).getDev0(),
mUlinkPidDevList.get(i).getDev1());
          sTmp2 = String.format("%08X%08X", mUlinkPidDevList.get(i).getPid0(),
mUlinkPidDevList.get(i).getPid1());
          iTmp  = mUlinkPidDevList.get(i).getKeytype();
          if ( iTmp == 0 )           // 未加密
          {
```

	sTmp3 = "未加密";
	}
	else if (iTmp == 1) // 已加密
	{
	sTmp3 = "已加密";
	}
	else // 未知加密类型
	{
	sTmp3 = "未知加密类型";
	}
	map.put("Mac", sTmp1);
	map.put("Pid", sTmp2);
	map.put("KeyType", sTmp3);
	listItem.add(map);
	}
	}
	adapter.notifyDataSetChanged(); // 刷新ui
	ScandBcakhandler.postDelayed(this, 1000); // 1秒扫描一次
	}
	}
	};
	IOS:

2.6 发送设备扫描信息

函数：	int pid_scan(void)
参数：	
返回：	0 ：表示成功
描述：	本函数执行(局域网)广播发送设备扫描命令，当设备接收此命令后，回复相关信息（包括设备识别码 PID、设备地址 DEV、加密情况），接收线程接收后保存在设备列表中，待调用 pid_list_read 返回设备列表信息。
用例：	**VC:** 参照上表 **Andorid:** 参照上表 **IOS:**

2.7 读取设备列表信息

函数：	int pid_list_read(UP2P_PID_DEV *pList,int maxlen)
参数：	pList： 待返回设备列表指针

	maxlen: 待返回设备列表最大数组，不能超过 255
返回：	0~255: 表示实际(局域网)在线设备数量
描述：	本函数返回局域网内所有在线设备信息，本函数与上表各个函数配合使用，实现对设备【一键配置】功能。
用例：	**VC:** 参照上表 **Andorid:** 参照上表 **IOS:**

2.8 设置通信密钥

函数：	int ulink_only_config(const char *devid, char *outkey)
参数：	devid: 16 字节的设备 ID 字符串 outkey: 返回密钥字符串（16 字节）
返回：	返回值含义见下表：
描述：	当设备通过【一键配置】连接上网络后，默认的通信密码是 0，因此需要通过本函数设置 16 位随机密码，后续所有通信都需要使用此密码才能正常通信。 注意：设置通信密钥函数只需要执行一次，保存返回的密码，下次直接使用 ulink_open 函数即可。

用例：	**VC:**
	int ret;
	ULINK * ulink;
	char devid[18],outkey[18];
	UP2P_PID_DEV *pUP2P_PID_DEV = &devlist[nSel];
	
	sprintf(devid,"%08x%08x",pUP2P_PID_DEV->dev0,pUP2P_PID_DEV->dev1);
	ret = **ulink_only_config**(devid,outkey);
	if (ret == ULINK_ERR_NONE)
	{
	ulink = ulink_open(devid,outkey);
	}
	
	Andorid:
	// 详细参考 openActivity.java
	// 判断数据合法性
	String sMAC=eMAC.getText().toString();
	if (sMAC.length()!=16)
	{
	ShowMessage("Mac长度必须为16字节");//设置显示的内容
	return;
	}
	// 由于 smnt发送数据量较大，所以在连接配置时，应该关闭smnt
	OnClick_SmntStop(null);
	OnClick_Close(null);

	// config，加密配置
	byte bKey[] = new byte[16]; // 此处只能用16字节，多了会转码错误
	int ret = vm101Data.ulinkNative.**ulinkOnlyConfig**(sMAC,bKey);
	if (ret != 0)// 配置失败
	{
	ShowErrorMessage(ret);
	return;
	}
	ShowMessage("配置成功");
	
	IOS:

2.9 一键配置

函数：	int ulink_onekey_config(const char *pid, const char *ssid, const char *key, int type, char *outdevid, char *outkey)
参数：	pid ：16 字节的产品 ID 字符串（由联讯公司分配） ssid ：无线路由器 SSID key ：无线路由器密码 type ：加密类型（默认 0）
返回：	ULINK_ERR: 当且仅当返回 ULINK_ERR_NONE 时表示成功 当返回成功时，参数 outdevid，outkey 有效： outdevid ：返回的设备 ID 字符串 outkey： 返回密钥字符串
描述：	此函数为简化版配置设备功能，内部执行动作顺序依次为： 1. 配置 wifi 信息（smlink_start） 2. 初始化扫描设备列表（pid_list_init） 3. 发送设备扫描信息（pid_scan） 4. 读取设备列表信息（pid_list_read） 5. 当发现设备 pid 相同且未加密时，设置通信密钥（ulink_only_config） 6. 返回设备 mac 以及加密密钥 以上动作即本章中 2.3~2.8 小节所示内容。

用例：	**VC:**
	
	Andorid:
	public int OneKeyConfig()
	{
	int ret = -1,i;
	String sPid = ePID.getText().toString(); // pid
	String sSsid = eSSID.getText().toString(); // ssid
	String sPsw = ePSW.getText().toString(); // psw
	try
	{
	byte[]bSsid = sSsid.getBytes("GBK");
	byte[]bPsw = sPsw.getBytes("GBK");
	byte[]bPid = sPid.getBytes("GBK");
	if (bSsid.length > 32)
	{

```
                        ShowMessage("ssid 长度必须小于 32 字节");
                        return ret;
                }
                if ( bPsw.length > 64 )
                {
                        ShowMessage("psw 长度必须小于 64 字节");
                        return ret;
                }
                if ( bPid.length != 16 )
                {
                        ShowMessage("Pid 长度必须为 16 字节");
                        return ret;
                }
                for (i=0; i<bSsid.length; i++)
                {
                        ulinkConfig.uSsid[i]  = bSsid[i];
                        ulinkConfig.uSsid[i+1]= 0;
                }
                for (i=0; i<bPsw.length; i++)
                {
                        ulinkConfig.uPsw[i]  = bPsw[i];
                        ulinkConfig.uPsw[i+1]= 0;
                }
                for (i=0; i<bPid.length; i++)
                {
                        ulinkConfig.uPid[i]  = bPid[i];
                        ulinkConfig.uPid[i+1]= 0;
                }

        }
        catch (UnsupportedEncodingException e)
        {
                // TODO Auto-generated catch block
                e.printStackTrace();
        }

        ret = vm101Data.ulinkNative.ulinkOnekeyConfig(ulinkConfig);
        if (ret != 0 )
        {
                ShowErrorMessage(ret);
                return ret;
        }
        return ret;
}
private Handler mHandler = new Handler()
{
        @Override
        public void handleMessage(Message msg)
        {
                super.handleMessage(msg);
                switch(msg.what)
                {
                case 0:  // 一键配置成功
                String sDev="";// 将结果转码
                String sKey="";
                try
                {
                        sDev = new String(ulinkConfig.uOutDev,"GBK");
                        sKey = new String(ulinkConfig.uOutKey,"GBK");
                }
                catch (UnsupportedEncodingException e)
                {
                   e.printStackTrace();
        }
                eMAC.setText(sDev);// 保存 mac
```

	eKEY.setText(sKey);// 保存 key
	OnClick_Open(null);// 执行 open 动作
	break;
	}
	bOneKeyConfig.setEnabled(true);
	
	}
	};
	IOS:

2.10 打开设备

函数：	ULINK *ulink_open(const char *devid, const char *key);
参数：	devid: 16 字节的设备 ID 字符串 key: 16 字节密钥字符串
返回：	NULL： 表示打开设备失败 非 NULL：表示打开设备成功 typedef struct ULINK ULINK;
描述：	在执行通信操作之前，需要使用此函数，只有当打开设备成功后，才能进行其他通信操作，与 ulink_close 配对使用。

用例：	**VC:**
	int ret;
	ULINK * ulink;
	char devid[18],outkey[18];
	UP2P_PID_DEV *pUP2P_PID_DEV = &devlist[nSel];
	
	sprintf(devid,"%08x%08x",pUP2P_PID_DEV->dev0,pUP2P_PID_DEV->dev1);
	ret = ulink_only_config(devid,outkey);
	if (ret == ULINK_ERR_NONE)
	{
	ulink = **ulink_open**(devid,outkey);
	}
	
	Andorid:
	// 判断数据合法性
	String sMAC=eMAC.getText().toString();
	String sKEY=eKEY.getText().toString();
	if (sMAC.length()!=16)
	{
	ShowMessage("Mac长度必须为16字节");//设置显示的内容
	return;
	}
	if (sKEY.length() !=16)
	{
	ShowMessage("Key长度必须为16字节");//设置显示的内容
	return;
	}
	
	// 由于 smnt发送数据量较大，所以在连接配置时，应该关闭smnt
	OnClick_SmntStop(null);
	
	// open,内部保存ulink指针
	int ret = vm101Data.ulinkNative.**UlinkNative_Open**(sMAC,sKEY);
	if (ret > 0)

	{
	bOpen.setEnabled(false);
	bClose.setEnabled(true);
	}
	IOS:

2.11 关闭设备

函数：	int ulink_close(ULINK *ulink);
参数：	ulink： 执行 ulink_open 成功后返回的变量
返回：	0 ：表示成功
描述：	关闭设备时，需要使用此函数，与 ulink_open 配对使用。
用例：	**VC:** ulink_close (ulink); **Andorid:** // 关闭ulink成功后，使ulink变回0初始化，否则容易出错 vm101Data.ulinkNative.**UlinkNative_Close**(); **IOS:**

2.12 发送通信指令

函数：	int ulink_cmd_send(ULINK *ulink, u32 cmd, void *param, int len);
参数：	ulink： 执行 ulink_open 成功后返回的变量 cmd: 待发送的通信指令 param: 待发送的通信数据 len： 待发送数据的长度
返回：	大于 0 ：表示发送数据长度 -1： 表示发送失败
描述：	所有通信指令均通过此函数发送，本函数与 ulink_cmd_wait 配对使用
用例：	**VC:** 下面通信指令详细介绍 **Andorid:** **IOS:**

2.13 接收通信指令

函数：	int ulink_cmd_wait(ULINK *ulink, u32 cmd, void *param, int maxlen);
参数：	ulink： 执行 ulink_open 成功后返回的变量

	cmd: 待接收的通信指令 param: 待接收的通信数据 len： 待接收最大数据的长度
返回：	大于 0 ：表示接收数据长度 -1： 表示接收失败
描述：	所有带接收的通信指令均= 发送指令+1，当调用 ulink_cmd_send 函数后，需要确保发送的内容 VM301 是否有接收到，调用本函数即实现此功能。 注意：本函数为阻塞操作，请勿在主线程中使用。
用例：	**VC:** 下面通信指令详细介绍 **Andorid:** **IOS:**

3.通信指令

3.1 串口发送

名称：	CMD_SEND_SERIAL
数值：	0x3000

用例：	**VC:**
	char sendbuff[256];
	int ret;
	
	strcpy(sendbuff,"vlinks");
	ulink_cmd_send(pMFCDlg->ulink, CMD_SEND_SERIAL,&sendbuff[0],strlen(sendbuff));
	ret = ulink_cmd_wait(pMFCDlg->ulink, CMD_SEND_SERIAL_ACK,NULL,0);
	
	Andorid:
	// 详细参考 uartActivity.java
	// 由于接收串口数据线程,为祖塞函数,所以需要使用message来处理,返回的结果
	Handler hRecvUart = new Handler()
	{
	@Override
	public void handleMessage(Message msg)
	{
	switch (msg.what)
	{
	case UlinkNative.CMD_READ_SERIAL: // 显示串口数据消息

```
    case UlinkNative.CMD_TXRX_SERIAL: // 以收发一体的形式发送串口数据
      // 将结果转码
      String str=null;
      try
      {
         str = new String((byte[])msg.obj,"GBK");
      }
      catch (UnsupportedEncodingException e)
      {
         e.printStackTrace();
      }
      String stmp = tRecvUart.getText()+str;
      tRecvUart.setText(stmp);
      break;
    case UlinkNative.CMD_SEND_SERIAL:// 发送串口数据
      Toast.makeText(uartActivity.this, "发送成功", Toast.LENGTH_SHORT).show();
      break;
    default:
      super.handleMessage(msg);
      break;
    }

}
};
// 发送串口数据
public void OnClick_SendUart(View view)
{
    // 获取串口发送数据
    try
    {
       int ret = -1;
       byte recvbuf[]= new byte[256];
       String stmp = eUart_Send_Buff.getText().toString();
       byte[]btmp  = stmp.getBytes("GBK");
       if (bSendRecv==0)
       {
          vm101Data.ulinkNative.UlinkNative_SendCmd(hRecvUart, UlinkNative.CMD_SEND_SERIAL, btmp);
       }
       else
       {
            vm101Data.ulinkNative.UlinkNative_SendCmd(hRecvUart, UlinkNative.CMD_TXRX_SERIAL, btmp);

       }
    }
    catch (UnsupportedEncodingException e)
    {
       e.printStackTrace();
    }
}
```

IOS:

3.2 串口接收

名称：	CMD_READ_SERIAL
数值：	0x3010

用例：	**VC:**
	char recbuff[256];
	ulink_cmd_send(pThis->ulink,CMD_READ_SERIAL,NULL,0);
	ret = ulink_cmd_wait(pThis->ulink,CMD_READ_SERIAL_ACK,&recbuff[0],sizeof(recbuff));
	if (ret > 0)
	{
	printf("%s\n",recbuff);

	}
	Andorid:
	// 接收串口数据线程,祖塞函数
	Thread thRecvUart = new Thread()
	{
	@Override
	public void run()
	{
	while(bOnPause != 2)// 接收线程标志位，0=运行，1=暂停，2=退出
	{
	if (bOnPause == 0 && bSendRecv==0)// bSendRecv==0,非收发一体
	{
	// 当没有发送数据时，读取串口内容
	if (vm101Data.ulinkNative.busy == 0)
	{
	vm101Data.ulinkNative.UlinkNative_SendCmd(hRecvUart, UlinkNative.**CMD_READ_SERIAL**, (Object)null);
	}
	}
	try // 每300ms运行一次接收函数
	{
	Thread.sleep(300);
	}
	catch (Exception ex)
	{
	}
	}
	}
	};
	IOS:

3.3 串口发送接收一体

名称：	CMD_TXRX_SERIAL
数值：	0x3020

用例：	**VC:**
	char sendbuff[256];
	char recbuff[256];
	ulink_cmd_send(pMFCDlg->ulink,CMD_TXRX_SERIAL,&sendbuff[0],strlen(sendbuff));
	ret = ulink_cmd_wait(pMFCDlg->ulink,CMD_TXRX_SERIAL_ACK,&recbuff[0],sizeof(recbuff));
	
	if (ret > 0)
	{
	printf("%s\n",recbuff);
	}
	Andorid:
	参考 3.1 串口发送
	
	IOS:

3.4 读 GPIO 口

名称：	CMD_GPIO_READ

数值：	0x3070

用例：	**VC:**
	UP2P_GPIO sUP2P_GPIO;
	sUP2P_GPIO.pin = ePin0;
	sUP2P_GPIO.mode = eGPIO_Input;
	ulink_cmd_send(ulink, CMD_GPIO_READ, &sUP2P_GPIO, sizeof(UP2P_GPIO));
	ret = ulink_cmd_wait(ulink, CMD_GPIO_READ_ACK, &sUP2P_GPIO, sizeof(UP2P_GPIO));
	if (ret > 0)
	{
	printf("pin%d = %d\n",sUP2P_GPIO.pin,sUP2P_GPIO.value);
	}
	Andorid:
	// 单个gpio，读取按键
	public void OnClick_OneGpioRead(View view)
	{
	UlinkGPIO ulinkGPIO = new UlinkGPIO();
	ulinkGPIO.pin = (int)ddOneGpioNumber.getSelectedItemId();
	ulinkGPIO.mode = UlinkNative.eGPIO_Input;
	vm101Data.ulinkNative.UlinkNative_SendCmd(hRecvUart, UlinkNative.**CMD_GPIO_READ**, ulinkGPIO);
	}
	IOS:

3.5 写 GPIO 口

名称：	CMD_GPIO_WRITE
数值：	0x3080

用例：	**VC:**
	UP2P_GPIO sUP2P_GPIO;
	sUP2P_GPIO.pin = ePin0;
	sUP2P_GPIO.mode = eGPIO_Output;
	sUP2P_GPIO.value = 1;// 1=高电平，0=低电平
	ulink_cmd_send(ulink, CMD_GPIO_WRITE, &sUP2P_GPIO, sizeof(UP2P_GPIO));
	ret = ulink_cmd_wait(ulink, CMD_GPIO_WRITE_ACK, NULL,0);
	if (ret >= 0)
	{
	printf("ok");
	}
	Andorid:
	// 单个gpio设置按键
	public void OnClick_OneGpio(View view)
	{
	UlinkGPIO ulinkGPIO = new UlinkGPIO();
	ulinkGPIO.pin = (int)ddOneGpioNumber.getSelectedItemId();
	ulinkGPIO.mode = UlinkNative.eGPIO_Input;
	ulinkGPIO.value= (int)ddOneGpioLevel.getSelectedItemId();//0=低电平，1=高电平
	vm101Data.ulinkNative.UlinkNative_SendCmd(hRecvUart, UlinkNative.**CMD_GPIO_WRITE**, ulinkGPIO);
	}
	IOS:

3.6 读模块版本号

名称：	CMD_WIFISDKVER_QURY
数值：	0x3030

用例：	**VC:**
	char recbuff[256];
	ulink_cmd_send(ulink,CMD_WIFISDKVER_QURY, NULL,0);
	ret0 = ulink_cmd_wait(ulink, CMD_WIFISDKVER_QURY_ACK, recbuff, sizeof(recbuff));
	if (ret > 0)
	{
	printf("%s\n",recbuff);
	}
	Andorid:
	IOS:

3.7 设置普通定时

名称：	CMD_CLOCK_SET
数值：	0x3110

用例：	**VC:**
	u8 buf[255];
	char tempbuff[100];
	settimepacke_st *apptm= (settimepacke_st *)buf;
	CString m_updatatimer_string;
	int ret;
	time_t tm;
	u8 ucGPIOCmd;
	int m_Week;
	if (sRadioNumber_Time==0 || sRadioNumber_Cmd==0)
	{
	MessageBox(TEXT("请选择定时选项"));
	return;
	}
	m_Week = GetWeek();// 返回周星期的选择情况，0=全部没有选择，bit0=星期 1，bit1=星期二。。。。
	if (sRadioNumber_Time==1 && m_Week==0)
	{
	MessageBox(TEXT("请选择星期"));
	return;
	}
	
	memset(buf,0x00,sizeof(buf));
	m_button_write.EnableWindow(FALSE);
	UpdateData(TRUE);
	if (pMFCDlg->check_ulink_busy(TRUE) == 0)
	{
	time(&tm);
	apptm->currentm = tm;// 当前时间
	apptm->mon_unixtm =pMFCDlg->get_unixtime("1990-01-01-00-00-00");//星期一时间
	
	if (sRadioNumber_Time==1)//周定时
	{
	// 开始时间
	m_week_date.GetTime(m_week_time);
	m_updatatimer_string = m_week_time.Format("1990-01-01-%H-%M-%S");
	pMFCDlg->StringToChar(m_updatatimer_string,tempbuff);
	apptm->settm.clkstar = pMFCDlg->get_unixtime(tempbuff);
	// 结束时间

```
                m_updatatimer_string = m_end_data.Format("3000-01-01-00-00-00");
                pMFCDlg->StringToChar(m_updatatimer_string,tempbuff);
                apptm->settm.clkend = pMFCDlg->get_unixtime(tempbuff);
                // 间隔时间
                apptm->settm.clkinv = 0;
                // 间隔定时
                apptm->settm.weeks = m_Week;

        }
        else//间隔定时
        {
                // 开始时间
                m_datetime_star.GetTime(m_star_data);
                m_updatatimer_string = m_star_data.Format("%Y-%m-%d-%H-%M-%S");
                pMFCDlg->StringToChar(m_updatatimer_string,tempbuff);
                apptm->settm.clkstar = pMFCDlg->get_unixtime(tempbuff);
                // 结束时间
                m_datetime_end.GetTime(m_end_data);
                m_updatatimer_string = m_end_data.Format("%Y-%m-%d-%H-%M-%S");
                pMFCDlg->StringToChar(m_updatatimer_string,tempbuff);
                apptm->settm.clkend = pMFCDlg->get_unixtime(tempbuff);
                // 间隔时间
                apptm->settm.clkinv = m_inv_value;
                // 间隔定时
                apptm->settm.weeks = 0x80;//当设定间隔定时时，必须设置为 0x80
        }

        if ( sRadioNumber_Cmd == 3 )// 串口命令
        {
                apptm->settm.cmd = TM_UartSend;
                // 串口数据
                apptm->settm.ulen = m_edit_uartstring.GetLength()+1;
                pMFCDlg->StringToChar(m_edit_uartstring,(char*)&apptm->settm.udata[0]);
        }
        else// GPIO 口命令
        {
                // GPIO 命令
                ucGPIOCmd = m_pin_number+1;
                ucGPIOCmd <<= 4;
                ucGPIOCmd |= m_pin_cmd;
                apptm->settm.cmd = ucGPIOCmd;
                // 串口数据
                apptm->settm.ulen = 0;
        }

        // 存储位置
        apptm->settm.uart_idx = m_timer_idx;
        apptm->settm.clk_idx = m_timer_idx;

        ulink_cmd_send(pMFCDlg->ulink, CMD_CLOCK_SET, apptm, sizeof(settimepacke_st)+apptm->settm.ulen);
        ret = ulink_cmd_wait(pMFCDlg->ulink, CMD_CLOCK_SET_ACK,NULL, 0);
        if (ret == 0)
        {
                MessageBox(TEXT("设置定时任务成功"));
        }
        pMFCDlg->busy_ulink_release();
}
m_button_write.EnableWindow(TRUE);
```

Andorid:

```
// 写定时任务
public void OnClick_bWriteTime(View view)
{
    UlinkTimer ulinkTimer = GetUlinkTimer();
    if (ulinkTimer == null)
    {
        return;
    }
     vm101Data.ulinkNative.UlinkNative_SendCmd(hRecvTimer, UlinkNative.CMD_CLOCK_SET, ulinkTimer);
}
```

```
private UlinkTimer GetUlinkTimer()
{
	UlinkTimer uTimer = new UlinkTimer();
	// 周定时 还是 间隔定时
	if ( radioZHOU.isChecked() == true )    // 周定时
	{
		uTimer.clkstar = DateToUnixtm(1990,0,1,iZHOU_Time_Hour,iZHOU_Time_Minute,iZHOU_Time_Second);
		uTimer.clkend  = DateToUnixtm(2115,0,1,iZHOU_Time_Hour,iZHOU_Time_Minute,iZHOU_Time_Second);
		uTimer.clkinv  = 0;
		uTimer.weeks   = GetWeeks();
		if ( uTimer.weeks == 0 )
		{
		ShowMessage("请选择星期");
		return null;
		}
	}
	else if ( radioDUAN.isChecked() == true )		// 间隔定时
	{
		uTimer.clkstar =
DateToUnixtm(iDUAN_Star_Data_Year,iDUAN_Star_Data_Month,iDUAN_Star_Data_Day,iDUAN_Star_Time_Hour,iDUAN_Star_Time_
Minute,iDUAN_Star_Time_Second);
		uTimer.clkend  =
DateToUnixtm(iDUAN_End_Data_Year,iDUAN_End_Data_Month,iDUAN_End_Data_Day,iDUAN_End_Time_Hour,iDUAN_End_Time_M
inute,iDUAN_End_Time_Second);
		uTimer.clkinv  = 0;
		String stmp = eDUAN_Times.getText().toString();
		if ( stmp.length() > 0 )
		{
		   uTimer.clkinv = Integer.parseInt(stmp);
		}
		uTimer.weeks   = 0x80;
		if ( uTimer.clkinv == 0 )
		{
		   ShowMessage("请输入正确的定时时间");
		   return null;
		}
	}
	else
	{
		ShowMessage("请选择定时类型");
		return null;
	}


	// 串口命令 还是 GPIO 命令
	if ( radioUART.isChecked() == true )    //串口命令
	{
		uTimer.cmd = UlinkNative.CMD_TM_UARTSEND;
		// 串口数据
		String stmp = eUART_Buff.getText().toString();
		try {
		   byte[]btmp   = stmp.getBytes("GBK");
		   uTimer.ulen  = btmp.length;
		   uTimer.udata = btmp.clone();
		} catch (UnsupportedEncodingException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}

	}
	else if ( radioGPIO.isChecked() == true ) //GPIO 命令
	{
		int iNumber = (int)ddPinNumber.getSelectedItemId()+1;
		int iLevel  = (int)ddPinLevel.getSelectedItemId();
		uTimer.cmd  = (int)((iNumber<<4)|iLevel);
		uTimer.ulen = 0;
	}
	else
	{
		ShowMessage("请选择定时动作");
		return null;
```

```
        }

        // 存储位置
        uTimer.uart_idx = (int)ddSaveNumber.getSelectedItemId();
        uTimer.clk_idx  = uTimer.uart_idx;

        uTimer.currentm   =
DateToUnixtm(c.get(Calendar.YEAR),c.get(Calendar.MONTH),c.get(Calendar.DAY_OF_MONTH),c.get(Calendar.HOUR_OF_DAY),c.ge
t(Calendar.MINUTE),c.get(Calendar.SECOND));
        uTimer.mon_unixtm = 631123200;// 中国时区，1990-1-1,0:0:0（周一时间值）

        return uTimer;
}
```

IOS:

3.8 读取普通定时

名称：	CMD_CLOCK_GET
数值：	0x3120

用例：	**VC:**
	// TODO: 在此添加控件通知处理程序代码
	u8 buf[255];
	char tempbuff[100];
	settimepacke_st *apptm= (settimepacke_st *)buf;
	CString m_updatatimer_string;
	int ret;
	time_t tm_t;
	struct tm *tblock;
	
	memset(buf,0x00,sizeof(buf));
	m_button_read.EnableWindow(FALSE);
	UpdateData(TRUE);
	if (pMFCDlg->check_ulink_busy(TRUE) == 0)
	{
	time(&tm_t);
	apptm->currentm = 0;
	apptm->mon_unixtm = 0;
	
	apptm->settm.clk_idx = m_timer_idx;// 选择第几个定时任务
	ulink_cmd_send(pMFCDlg->ulink, CMD_CLOCK_GET, apptm, sizeof(settimepacke_st));
	ret = ulink_cmd_wait(pMFCDlg->ulink, CMD_CLOCK_GET_ACK,&apptm->settm, 100);
	if (ret > 0)
	{
	if (apptm->settm.cmd != 0)// 定时任务不空
	{
	// 周定时 还是 间隔定时
	if (apptm->settm.weeks == 0x80)// 间隔定时
	{
	SetRadio(2);
	tm_t = apptm->settm.clkstar;// 开始时间
	m_star_data = tm_t;
	m_datetime_star.SetTime(&m_star_data);
	
	tm_t = apptm->settm.clkend;// 结束时间
	m_end_data = tm_t;
	m_datetime_end.SetTime(&m_end_data);
	// 间隔时间
	m_inv_value = apptm->settm.clkinv;
	}
	else // 周定时
	{
	SetRadio(1);

```
                                        tm_t = apptm->settm.clkstar;// 开始时间
                                        m_week_time = tm_t;
                                        m_week_date.SetTime(&m_week_time);
                                        SetWeek(apptm->settm.weeks);
                                }

                                // 串口命令 还是 GPIO 命令
                                if ( apptm->settm.cmd == TM_UartSend )
                                {
                                        SetRadio(3);
                                        m_edit_uartstring = apptm->settm.udata;
                                }
                                else
                                {
                                        SetRadio(4);

                                        m_pin_cmd = apptm->settm.cmd&0x01;// ON=1 还是 OFF=0
                                        m_pin_number = ((apptm->settm.cmd>>4)&0x0F)-1;

                                }


                        }
                        else        // 命令为空，可以重新设置
                        {
                                SetRadio(0);
                                MessageBox(TEXT("定时任务空"));
                        }
                }
                else
                {
                        MessageBox(TEXT("读回定时任务失败！"));
                }
                UpdateData(FALSE);
                pMFCDlg->busy_ulink_release();
}
m_button_read.EnableWindow(TRUE);
```

Andorid:

```
// 读定时任务
public void OnClick_bReadTime(View view)
{
    UlinkTimer ulinkTimer = new UlinkTimer();
    // 当前时间必须设置为 0，或者当前时间，不能不赋初始化
    ulinkTimer.currentm     = 0l;     /* 当前时间 */
    ulinkTimer.mon_unixtm   = 0l;     /* 星期一 00:00 的 unix 时间帧*/
    ulinkTimer.ulen     = 0;          /* 长度最好设置为 0*/
     ulinkTimer.clk_idx  = (int)ddSaveNumber.getSelectedItemId();          /* 序号*/
    vm101Data.ulinkNative.UlinkNative_SendCmd(hRecvTimer, UlinkNative.CMD_CLOCK_GET, ulinkTimer);
}

// 由于接收串口数据线程,为祖塞函数,所以需要使用 message 来处理,返回的结果
Handler hRecvTimer = new Handler()
{
        @Override
        public void handleMessage(Message msg)
        {
                switch (msg.what)
                {
                case UlinkNative.CMD_CLOCK_SET:                 // 设置定时
                        Toast.makeText(timeActivity.this, "设置定时成功", Toast.LENGTH_SHORT).show();
                        break;
                case UlinkNative.CMD_CLOCK_GET:                 //读取模块模块已经设置的定时
                        Toast.makeText(timeActivity.this, "读定时成功", Toast.LENGTH_SHORT).show();
                        ShowUlinkTimer((UlinkTimer)msg.obj);
                        break;
                case UlinkNative.CMD_CLOCK_DEL:                 //删除定时
                        Toast.makeText(timeActivity.this, "删除定时成功", Toast.LENGTH_SHORT).show();
                        break;
```

```
                default:
                        super.handleMessage(msg);
                break;
                }

        }
};
private void ShowUlinkTimer(UlinkTimer uTimer)
{
        if ( uTimer.cmd == 0 || uTimer.clkend<600000000)          // 空定时任务
        {
                SetRadioGroup(0xff);
                ShowMessage("无定时任务");
                return;
        }

        // 周定时 还是 间隔定时
        if ( uTimer.weeks == 0x80 )  // 间隔定时
        {
              SetRadioGroup(1);
              // 显示开始时间
              Date dtmp1 = UnixtmToDate(uTimer.clkstar);
              Show_eDUAN_Star_Data((int)dtmp1.getYear()+1990,(int)dtmp1.getMonth(),(int)dtmp1.getDay());
              Show_eDUAN_Star_Time((int)dtmp1.getHours(),(int)dtmp1.getMinutes(),(int)dtmp1.getSeconds());
              // 显示结束时间
               Date dtmp2 = UnixtmToDate(uTimer.clkend);
               Show_eDUAN_End_Data((int)dtmp2.getYear()+1990,(int)dtmp2.getMonth(),(int)dtmp2.getDay());
               Show_eDUAN_End_Time((int)dtmp2.getHours(),(int)dtmp2.getMinutes(),(int)dtmp2.getSeconds());
                // 显示定时时间
                String stmp = ""+uTimer.clkinv;
                eDUAN_Times.setText(stmp);
        }
        else                                                      // 周定时
        {
                SetRadioGroup(0);
                // 显示开始时间
                Date dtmp = UnixtmToDate(uTimer.clkstar);
                Show_eZHOU_Time((int)dtmp.getHours(),(int)dtmp.getMinutes(),(int)dtmp.getSeconds());
                // 显示周设定
                ShowWeeks(uTimer.weeks);
        }

        // 串口命令 还是 GPIO 命令
        if ( uTimer.cmd == UlinkNative.CMD_TM_UARTSEND )
        {
                SetRadioGroup(2);
                // 将结果转码
                String str="";
                try
                {
                   str = new String(uTimer.udata,0,uTimer.ulen,"GBK");
                }
                catch (UnsupportedEncodingException e)
                {
                  e.printStackTrace();
                }
                eUART_Buff.setText(str);
        }
        else
        {
                SetRadioGroup(3);
                int m_pin_cmd    = uTimer.cmd&0x01;                       // ON=1 还是 OFF=0
                int m_pin_number = ((uTimer.cmd>>4)&0x0F)-1;
                ddPinNumber.setSelection(m_pin_number);
                ddPinLevel.setSelection(m_pin_cmd);
        }

}
```

IOS:

3.9 删除普通定时

名称：	CMD_CLOCK_DEL
数值：	0x3130

用例：	**VC:**
	// TODO: 在此添加控件通知处理程序代码
	u8 buf[255];
	char tempbuff[100];
	settimepacke_st *apptm= (settimepacke_st *)buf;
	CString m_updatatimer_string;
	int ret;
	time_t tm;
	u8 ucGPIOCmd;
	int m_Week;
	
	memset(buf,0x00,sizeof(buf));
	m_button_del.EnableWindow(FALSE);
	UpdateData(TRUE);
	if (pMFCDlg->check_ulink_busy(TRUE) == 0)
	{
	time(&tm);
	apptm->currentm = tm;
	apptm->mon_unixtm = pMFCDlg->get_unixtime(MOND_UNIXTM)-UTC2BJ;
	
	apptm->settm.cmd = 0;//命令删除
	// 存储位置
	apptm->settm.uart_idx = m_timer_idx;
	apptm->settm.clk_idx = m_timer_idx;
	
	ulink_cmd_send(pMFCDlg->ulink, CMD_CLOCK_SET, apptm, sizeof(settimepacke_st)+apptm->settm.ulen);
	ret = ulink_cmd_wait(pMFCDlg->ulink, CMD_CLOCK_SET_ACK,NULL, 0);
	if (ret == 0)
	{
	MessageBox(TEXT("删除定时任务成功"));
	SetRadio(0);
	}
	pMFCDlg->busy_ulink_release();
	}
	m_button_del.EnableWindow(TRUE);
	Andorid:
	// 删除定时任务
	public void OnClick_bDelTime(View view)
	{
	UlinkTimer ulinkTimer = new UlinkTimer();
	ulinkTimer.currentm = 0l; /* 当前时间 */
	ulinkTimer.mon_unixtm = 0l; /* 星期一 00:00 的 unix 时间帧*/
	ulinkTimer.ulen = 0; /* 长度最好设置为 0*/
	ulinkTimer.clk_idx = (int)ddSaveNumber.getSelectedItemId(); /* 序号*/
	ulinkTimer.uart_idx = ulinkTimer.clk_idx; /* c 串口序号*/
	ulinkTimer.clkstar = 0l; /* 执行时间 ：周期定时则为最近定时 unix time */
	ulinkTimer.clkend = 0l; /* 定时结束时间 */
	ulinkTimer.clkinv = 0; /* 定时间隔 只有 weeks == 0x80 时才生效*/
	ulinkTimer.weeks = 0; /* 定时执行的星期 0x01 mon；0x02 tus ..*/
	ulinkTimer.cmd = 0; /* 动作*/
	ulinkTimer.udata[0] = 0; /* 串口数据 */
	vm101Data.ulinkNative.UlinkNative_SendCmd(hRecvTimer, UlinkNative.**CMD_CLOCK_DEL**, ulinkTimer);
	}
	
	IOS:

4.PID 含义解析

PID: 表示设备类型识别码,请查看下面详细定义

PID0				PID1				UUID	PRODUCT
Byte4	Byte3	Byte2	Byte1	Byte4	Byte3	Byte2	Byte1		
保留		厂家		类型		型号			

5.结构体定义

5.1 设备列表

```
typedef struct{
        u32 pid0;       // 设备识别码
        u32 pid1;
        u32 dev0;       // 设备名，通常为 mac 地址
        u32 dev1;
        u8 keytype;     // 加密情况,0=未加密,1=已加密
} UP2P_PID_DEV;
```

5.2 设置通信密钥返回值

```
typedef enum
{
ULINK_ERR_NONE,                 // 成功
ULINK_ERR_DEVID_INVILD,         // 无效的 devid
ULINK_ERR_DEV_OFFLINE,          // 设备不在线
ULINK_ERR_SERVER_OFFLINE,       // 服务器不在线, 或者广域网络不通
ULINK_ERR_INIT_TOKEN,     // 初始化令牌错误, 可能是模块已经被配置过
```

```
ULINK_ERR_CONFIG_HOST,  // 配置服务器错误, 可能是模块网络不好或者 flash 写失败
ULINK_ERR_INIT_KEY,     // 初始化密钥错误, 可能是模块网络不好
ULINK_ERR_DEV_NOFOUND,// 没有找到设备, 可能是附近没有等待一键配置的设备
ULINK_ERR_PID_INVILD,   // 错误的 PID
} ULINK_ERR;
```

5.3 GPIO 结构体

```
typedef struct{
        u8 pin;         // gpio pin 脚号
        u8 mode;        // gpio 工作模式（输入/输出）
        u8 value;       // gpio pin 脚当前电平值
}UP2P_GPIO;
enum
{
        eGPIO_Output = 0,// 输入模式
        eGPIO_Input,    // 输出模式
};
enum
{
        ePin0 = 0,      // pin0 Input/Output     I2C_CLK/GPIO/PWM
        ePin1,          // pin1 Input/Output     I2C_SDA/GPIO/PWM
        ePin2,          // pin2 Input
        ePin3,          // pin3 Output
        ePin4,          // pin4 Input/Output     GPIO/PWM
        epinMX,
};
```

5.4 普通定时结构体

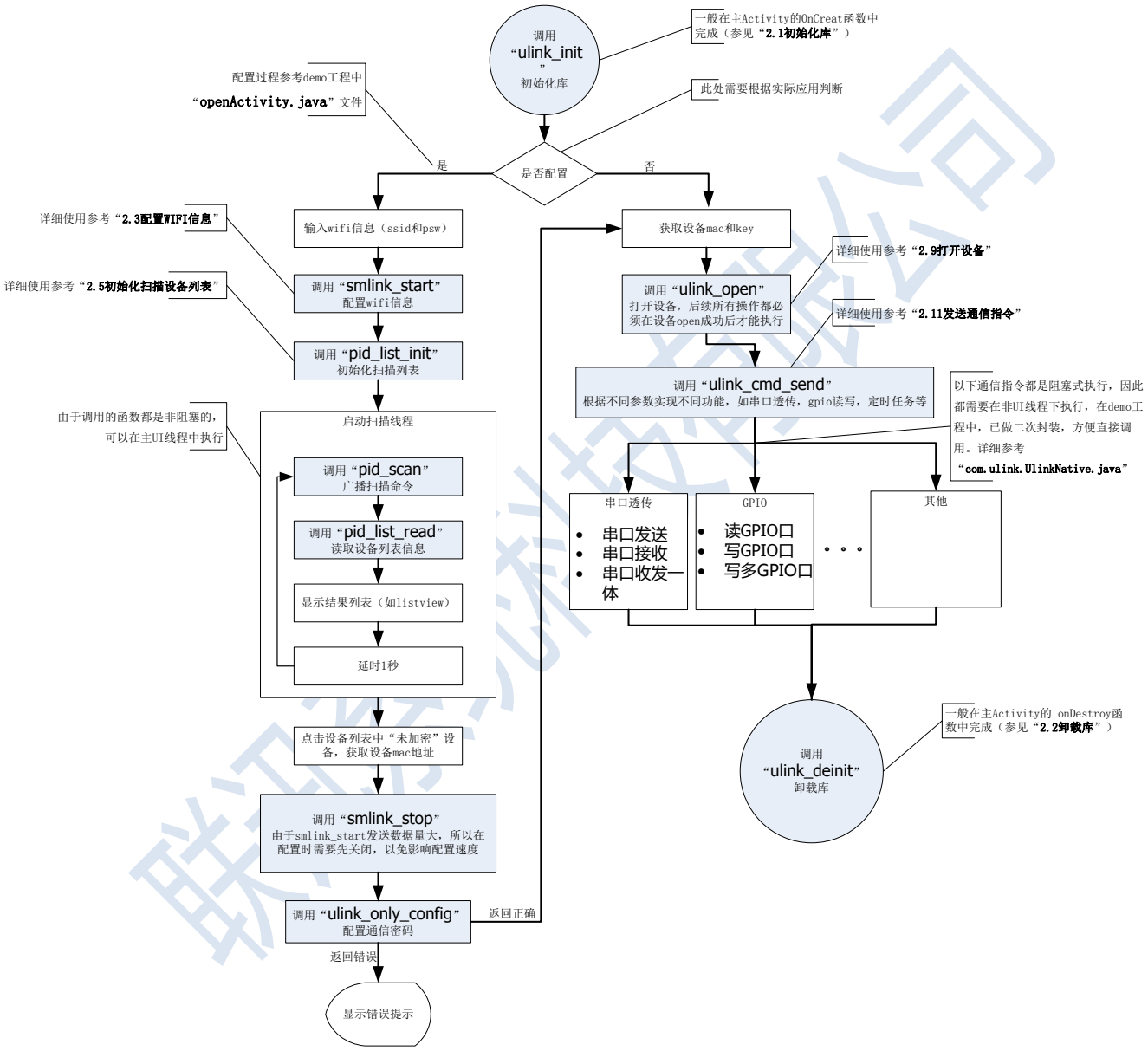
```
enum
{
        TM_Fuc_NoEv =0x00,
        TM_UartSend =0x01,
        TM_Pin0_ON  =0x11,
        TM_Pin0_OFF =0x10,
        TM_Pin1_ON  =0x21,
        TM_Pin1_OFF =0x20,
        TM_Pin2_ON  =0x31,
        TM_Pin2_OFF =0x30,
        TM_Pin3_ON  =0x41,
        TM_Pin3_OFF =0x40,
        TM_Pin4_ON  =0x51,
        TM_Pin4_OFF =0x50,
        TM_Pin5_ON  =0x61,
        TM_Pin5_OFF =0x60,
        TM_Pin6_ON  =0x71,
        TM_Pin6_OFF =0x70,

        TM_Max
};
```

```
/*
* 时间帧格式
*/
typedef struct{
        u32 clkstar;      /* 执行时间 ：周期定时则为最近定时 unix time */
        u32 clkend;       /* 定时结束时间 */
        u32 clkinv;       /* 定时间隔 只有 weeks == 0x80 时才生效*/
        u8 weeks;         /* 定时执行的星期 0x01 mon；0x02 tus ..*/
        u8 cmd;           /* 动作*/
        u8 clk_idx;       /*序号*/
        u8 uart_idx;      /*串口序号*/
        u8 ulen;          /* 串口数据长度*/
        u8 udata[0];      /* 串口数据 */
} timestamp_st;           /*4*5=20Byte*/
/*
* app 发送的时间帧
*/
typedef struct{
        u32 currentm;             /* 当前时间 */
        u32 mon_unixtm;           /* 星期一 00:00 的 unix 时间帧*/
        timestamp_st settm;
} settimepacke_st;                /*7*4 = 28 byte*/
```

6.附录

6.1 参考开发流程



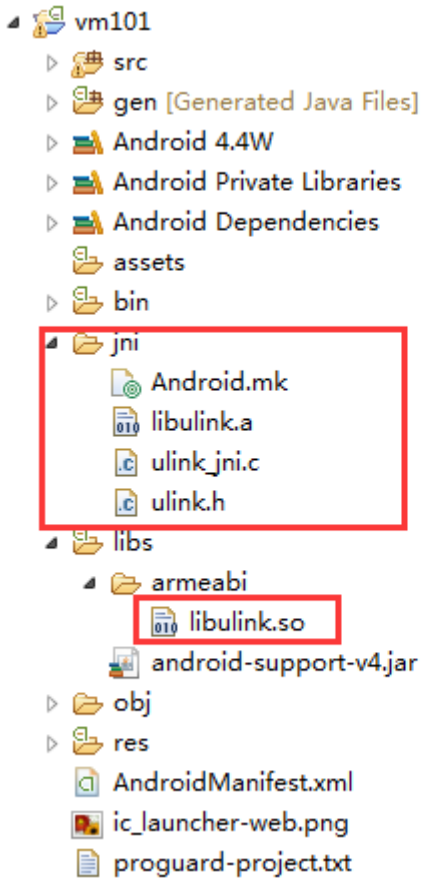
6.2 搭建 JNI 编译环境简易步骤

http://www.th7.cn/Program/Android/201411/311015.shtml

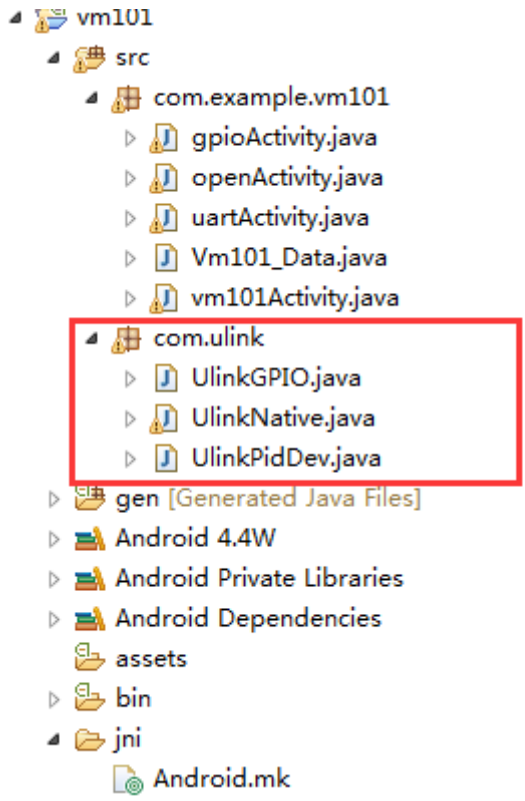
注意：包名不要用”_”(下划线)命名，因为 JNI 函数定义用”_”区分不同的包路径

6.3 移植 demo 动态库以及接口

- 新建 jni 文件夹以及 libs 文件夹，如下：
- 拷贝 demo 工程下 jni 所有文件
- 拷贝 libs\armeabi 下 libulink.so 文件



- 拷贝 demo 工程 com.ulink 包


vm101
src
com.example.vm101
gpioActivity.java
openActivity.java
uartActivity.java
Vm101_Data.java
vm101Activity.java
com.ulink
UlinkGPIO.java
UlinkNative.java
UlinkPidDev.java
gen [Generated Java Files]
Android 4.4W
Android Private Libraries
Android Dependencies
assets
bin
jni
Android.mk