# 鑫源温控器Android开发包使用说明

[鑫源温控器Android开发包使用说明 1](#_Toc19357)

[概述 2](#_Toc22750)

[主要功能 2](#_Toc5868)

[使用说明 2](#_Toc16127)

[导入库文件 2](#_Toc9474)

[初始化和释放 2](#_Toc2424)

[使用概要说明 3](#_Toc17670)

[接口列表 3](#_Toc4671)

[通用接口 3](#_Toc15064)

[局域网扫描 4](#_Toc8299)

[一键配置 4](#_Toc24842)

[鑫源温控器设备控制 5](#_Toc24842)

[数据结构](#_Toc3573) 7

## 概述

Android开发包对已有接口和部分通用流程进行封装，提供简单易用开发接口，以供友商自主开发移动客户端软件。本开发包适用于x86、mips、armeabi、armeabi-v7a处理器的Android移动平台。

## 主要功能

1. 提供通用的一键配置功能
2. 提供局域网扫描能力
3. 提供事件响应回调接口
4. 提供鑫源温控器控制接口
5. 提供鑫源温控器矫正参数设置接口
6. 提供鑫源温控器智能参数设置接口
7. 提供鑫源温控器状态查询

## 使用说明

### 导入库文件

将银河风云提供的.so库和jar文件拷贝至Android工程的libs文件目录下即可

### 初始化和释放

初始化调用XinYuanKit.getInstance().init(appContext);即可，其中，appContext应为app的上下文参数。

释放可直接调用XinYuanKit.getInstance().release();

### 使用概要说明

Android APP客户端与开发包之间，通过回调接口WukitEventHandler传递消息，该接口申明了唯一的回调方法: **public void callback(int wukitEvent, int obj\_handle, int err\_no);**

回调事件都是由类**BaseEventMappe**r定义的整型常量，所有方法的返回值和错误码都为KitRs定以的int常量，APP可通过对事件范围和handle进行注册，来监听对应的回调事件，

也可注销监听。

### 接口列表

### 通用接口

|  |  |  |  |
| --- | --- | --- | --- |
| **方法名** | **void callback (int wukitEvent, int obj\_handle, int err\_no)** | | |
| **功能说明** | Wukit与APP的消息传递和处理回调函数 | | |
| **参数** | wukitEvent | 回调事件 | |
| obj\_handle | 设备句柄 | |
| err\_no | 错误号 | |
| **返回值** | 无 | |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **方法名** | ***void registerEvent(int eventMin, int eventMax, int handle, WukitEventHandler eh)*** | | |
| **功能说明** | 注册监听的事件范围和设备句柄，事件范围为[min, max) | | |
| **参数** | eventMin | 最小事件 | |
| eventMax | 最大事件 | |
| Handle | 监听的设备句柄，若为WukitEventHandler.HANDLE\_ALL,则监听所有设备 | |
| eh | 监听对象，是接口WukitEventHandler的实现 | |
| **返回值** | 无 | |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **方法名** | ***void unRegisterEvent(int eventMin, int eventMax, int handle, WukitEventHandler eh)*** | | |
| **功能说明** | 注销监听的事件范围和设备句柄，事件范围为[min, max) | | |
| **参数** | eventMin | 最小事件 | |
| eventMax | 最大事件 | |
| Handle | 设备句柄，若为WukitEventHandler.HANDLE\_ALL,则注销监听所有设备 | |
| eh | 监听对象，是接口WukitEventHandler的实现 | |
| **返回值** | 无 | |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **方法名** | ***void unRegisterEvent(WukitEventHandler eh)*** | | |
| **功能说明** | 注册监听的所有事件 | | |
| **参数** | eh | 监听对象，是接口WukitEventHandler的实现 | |
| **返回值** | 无 | |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **方法名** | ***ClibLimit getLimilation()*** | | |
| **功能说明** | 获取系统限制 | | |
| **参数** | 无 |  | |
| **返回值** | ClibLimit对象 | | 数据结构见后面说明 |

|  |  |  |  |
| --- | --- | --- | --- |
| **方法名** | ***void setDebugEnable(boolean enable)*** | | |
| **功能说明** | 设置是否输出打印信息 | | |
| **参数** | enable | true:打印 false:不打印 | |
| **返回值** | 无 | |  |

### 局域网扫描

|  |  |  |  |
| --- | --- | --- | --- |
| **方法名** | ***void startProbe()*** | | |
| **功能说明** | 开始局域网扫描 | | |
| **参数** | 无 |  | |
| **返回值** | 无 | |  |
| **诱发事件** | PE\_DEV\_CHANGED | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **方法名** | ***void stopProbe()*** | | |
| **功能说明** | 结束局域网扫描 | | |
| **参数** | 无 |  | |
| **返回值** | 无 | |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **方法名** | ***ArrayList<KitLanDev> getLanDevInfo()*** | | |
| **功能说明** | 获取扫描的局域网设备 | | |
| **参数** | 无 |  | |
| **返回值** | 扫描到的设备容易，KitLanDev见后续说明 | |  |

### 一键配置

|  |  |  |  |
| --- | --- | --- | --- |
| **方法名** | ***void startSmartConfig (String ssid, String pwd, int confMode, int overtime)*** | | |
| **功能说明** | 开始一键配置 | | |
| **参数** | ssid | Wifi的ssid | |
| pwd | wifi密码 | |
| confMode | 配置模式，SdkConfigApi.CONF\_MODE\_MUT 组播模式  SdkConfigApi.CONF\_MODE\_AP 手机AP模式 | |
| overtime | 超时时间，单位秒 | |
| **返回值** | 无 | |  |
| **诱发事件** | BaseEventMapper.SC\_CONFIG\_OK 配置成功  BaseEventMapper.SC\_CONFIG\_FAIL 配置失败  回调事件中的obj\_handle为扫描设备句柄，可通过getLanDevInfo获取  以后的事件描述中，不再再回调事件前，加上BaseEventMapper | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **方法名** | ***void stopSmartConfig()*** | | |
| **功能说明** | 停止一键配置 | | |
| **参数** | 无 |  | |
| **返回值** | 无 | |  |

### 鑫源温控器设备控制

|  |  |  |
| --- | --- | --- |
| **方法名** | ***Int xyCtrlPower（int handle,byte onoff)*** | |
| **功能说明** | 鑫源温控器控制开关接口 | |
| **参数** | handle | 设备句柄 |
|  | onoff | 0代表关闭 1代表开启 |
| **返回值** | Int 请求成功与否 | |
| **诱发事件** | CE\_CTRL\_OK, CE\_CTRL\_FAIL, UE\_MODIFY | |

|  |  |  |
| --- | --- | --- |
| **方法名** | ClibXYWkq XinYuanGetInfo(int handle) | |
| **功能说明** | 获取设备的基本信息 | |
| **参数** | handle | 设备句柄 |
| **返回值** | 鑫源温控器设备信息，ClibXYWkq见后面的说明 | |
| **诱发事件** | 无 | |

|  |  |  |
| --- | --- | --- |
| **方法名** | int xyCtrlTemp(int handle, byte temp) | |
| **功能说明** | 恒温模式下的温度设置 | |
| **参数** | handle | 设备句柄 |
|  | temp | 温度值 |
| **返回值** | Int 请求成功与否 | |
| **诱发事件** | CE\_CTRL\_OK, CE\_CTRL\_FAIL, UE\_MODIFY | |

|  |  |  |
| --- | --- | --- |
| **方法名** | int xyCtrlTimer(int handle, short time) | |
| **功能说明** | 休假模式时间设置 | |
| **参数** | handle | 设备句柄 |
| time | 时间 单位小时 |
| **返回值** | 请求成功与否 | |
| **诱发事件** | CE\_CTRL\_OK, CE\_CTRL\_FAIL, UE\_MODIFY | |

|  |  |  |
| --- | --- | --- |
| **方法名** | int xyCtrlLockOnoff(int handle, byte onoff) | |
| **功能说明** | 按键锁定控制 | |
| **参数** | handle | 设备句柄 |
| **参数** | onoff | 锁定开关 0关 1开 |
| **返回值** | 请求成功与否 | |
| **诱发事件** | CE\_CTRL\_OK, CE\_CTRL\_FAIL, UE\_MODIFY | |

|  |  |  |
| --- | --- | --- |
| **方法名** | int xyCtrlMode(int handle, byte mode) | |
| **功能说明** | 模式设置 | |
| **参数** | handle | 设备句柄 |
| mode | 1-恒温 2-智能 3-休假 |
| **返回值** | 请求成功与否 | |
| **诱发事件** | CE\_CTRL\_OK, CE\_CTRL\_FAIL, UE\_MODIFY | |

|  |  |  |
| --- | --- | --- |
| **方法名** | int xyConfigSmartMode(int dev\_handle, ClibXYWkqSmartMode pst\_info) | |
| **功能说明** | 智能模式参数配置 | |
| **参数** | handle | 设备句柄 |
| Pst\_info | 智能模式参数配置 |
| **返回值** | 命令是否发送成功 | |
| **诱发事件** | CE\_CTRL\_OK, CE\_CTRL\_FAIL, UE\_MODIFY | |

|  |  |  |
| --- | --- | --- |
| **方法名** | int xyCtrlAdjust(int dev\_handle, ClibXYWkqAdjust padjust) | |
| **功能说明** | 后台校正参数 | |
| **参数** | handle | 设备句柄 |
| padjust | 后台矫正参数配置 |
| **返回值** | 命令是否发送成功 | |
| **诱发事件** | CE\_CTRL\_OK, CE\_CTRL\_FAIL, UE\_MODIFY | |

数据结构

|  |  |
| --- | --- |
| 类型 | 定义 |
| com.galaxywind.wukit.kits.clibevent.BaseEventMapper | public abstract class BaseEventMapper {  //用户事件  public static final int *UE\_BEGIN* = 0;  public static final int *UE\_LOGOUT* = *UE\_BEGIN* + 1;//注销  public static final int *UE\_OFFLINE* = *UE\_BEGIN* + 2;//登陆成功，设备离线  public static final int *UE\_ONLINE* = *UE\_BEGIN* + 3;//登陆成功，设备在线  public static final int *UE\_MODIFY* = *UE\_BEGIN* + 4;//数据更新  public static final int *UE\_PASSWD\_OK* = *UE\_BEGIN* + 5;//设置密码成功  public static final int *UE\_PASSWD\_FAIL* = *UE\_BEGIN* + 6;//设置密码失败  public static final int *UE\_NICKNAME\_OK* = *UE\_BEGIN* + 7;//设置昵称成功  public static final int *UE\_NICKNAME\_FAIL* = *UE\_BEGIN* + 8;//是指昵称失败  public static final int *UE\_LOGIN\_ERROR* = *UE\_BEGIN* + 9;//登陆失败  public static final int *UE\_DEV\_UPGRADE\_READY* = *UE\_BEGIN* + 10;//准备好升级  public static final int *UE\_DEV\_UPGRADE\_PROGRESS* = *UE\_BEGIN* + 11;//设备正在升级  public static final int *UE\_REBOOT\_FAIL* = *UE\_BEGIN* + 12;//设备重启失败  public static final int UE\_HISTORY\_DIGEST = UE\_BEGIN + 13;// 设备历史记录摘要更新  public static final int UE\_HISTORY\_ITEM = UE\_BEGIN + 14;// 设备历史记录摘要更新  public static final int UE\_ALARM = UE\_BEGIN + 15;//通用报警信息  public static final int *UE\_END* = *UE\_BEGIN* + 99;    //扫描事件  public static final int *PE\_BEGIN* = 100;  public static final int *PE\_DEV\_CHANGED* = *PE\_BEGIN* + 1;  public static final int *PE\_END* = *PE\_BEGIN* + 99;    //手机事件  public static final int *PHOE\_BEGIN* = 300;  public static final int *PHOE\_PHONE\_USER\_REGISTER\_OK* = *PHOE\_BEGIN* + 1;//绑定手机成功  public static final int *PHOE\_PHONE\_USER\_REGISTER\_FAILED* = *PHOE\_BEGIN* + 2;//绑定手机失败  public static final int *PHOE\_PHONE\_USER\_REGISTERED* = *PHOE\_BEGIN* + 3;//手机账号已被注册  public static final int *PHOE\_PHONE\_USER\_RESET\_OK* = *PHOE\_BEGIN* + 4;//重设手机密码成功  public static final int *PHOE\_PHONE\_USER\_RESET\_FAILED* = *PHOE\_BEGIN* + 5;//重设手机密码  public static final int *PHOE\_PHONE\_USER\_GOOD\_VCODE* = *PHOE\_BEGIN* + 6;//输入手机验证码正确  public static final int *PHOE\_PHONE\_USER\_BAD\_VCODE* = *PHOE\_BEGIN* + 7;//输入手机验证码失败  public static final int *PHOE\_PHONE\_ADD\_DEV\_FINISH* = *PHOE\_BEGIN* + 8;//手机添加设备完成  public static final int *PHOE\_PHONE\_DEL\_DEV\_FINISH* =*PHOE\_BEGIN* + 9;//手机删除设备完成  public static final int *PHOE\_END* = *PHOE\_BEGIN* + 99;    //空调事件  public static final int *SAE\_BEGIN* = 300;  public static final int *SAE\_CODE\_MATCH\_READY*= *SAE\_BEGIN* + 1;//设备准备好匹配 1.云匹配是准备好接收遥控板编码 2.全匹配是已经开始匹配  public static final int *SAE\_CODE\_MATCH\_PROCESS* = *SAE\_BEGIN* + 2;//匹配编码过程，通过errno可以获取匹配百分比  public static final int *SAE\_CODE\_MATCH\_OK* = *SAE\_BEGIN* + 3;//匹配编码成功  public static final int *SAE\_CODE\_MATCH\_FAILED* = *SAE\_BEGIN* + 5;//匹配编码失败  public static final int *SAE\_POWER\_NOTIFY* = *SAE\_BEGIN* + 6;//设备功率发生变化  public static final int *SAE\_END* = *SAE\_BEGIN* + 99;    //控制事件  public static final int *CE\_BEGIN* = 400;  public static final int *CE\_CTRL\_OK* = *CE\_BEGIN* + 1;//控制成功  public static final int *CE\_CTRL\_FAIL* = *CE\_BEGIN* + 2;//控制失败  public static final int *CE\_END* = *CE\_BEGIN* + 99;    //智能事件  public static final int *SME\_BEGIN* = 500;  public static final int *SME\_HOME\_ON* = *SME\_BEGIN* + 1;  public static final int *SME\_HOME\_CANCEL* = *SME\_BEGIN* + 2;  public static final int *SME\_END* = *SME\_BEGIN* + 99;  /\*  \* 下面的事件是SDK自己产生的事件，而不是由CLIB映射过来的  \*/  public static final int *WUKIT\_EVENT\_BEGIN* = 1000000;  public static final int *WUKIT\_EVENT\_END* = 2000000;  //一键配置事件  public static final int *SC\_BEGIN* = *WUKIT\_EVENT\_BEGIN*;  public static final int *SC\_CONFIG\_OK* = *SC\_BEGIN* + 1;  public static final int *SC\_CONFIG\_FAIL* = *SC\_BEGIN* + 2;  public static final int *SC\_END* = *SC\_BEGIN* + 99;  } |
| com.galaxywind.wukit.clibinterface.ClibModuleVersion | public class ClibModuleVersion {  public static final int UPDATE\_LANGE\_CH = 1;//简体中文  public static final int UPDATE\_LANGE\_EN = 2;//英文    /\* 固件版本信息，类似 1.2.3 \*/  public String soft\_version;  /\* 升级包版本信息，类似 1.2.4 \*/  public String upgrade\_version;  /\* 新版本信息 ，can\_update为true时有效 \*/  public String new\_version;  /\* true表示设备有新版本可以升级 \*/  public boolean can\_update;  /\* 新版本描述信息 \*/  public String release\_desc;  /\* 新版本URL \*/  public String release\_url;  /\* 新版本发布日期 \*/  public String release\_date;  /\*单片式能否升级\*/  public boolean stm\_can\_update;  /\*单片机当前版本\*/  public String stm\_cur\_version;  /\*单片式能否升级\*/  public String stm\_newest\_version;  /\*单片机升级版本\*/  public String stm\_release\_url;  } |
| com.galaxywind.wukit.user.KitLanDev | public class KitLanDev {  public int handle;//device unique handle 设备唯一标识  public long dev\_sn;//设备序列号  public int dev\_type;//主类型  public byte ext\_type;//扩展类型  public int dev\_run\_mode;//当前运行模式  public byte is\_upgrading;//正在升级标志  } |
| com.galaxywind.wukit.kits.KitRs | //返回值  public static final int TRUE = 1;//成功  public static final int FALSE = 0;//失败  public static final int OK = 0;//成功  public static final int FAIL = -1;//失败  public static final int NOT\_SUPPORT = -2;//不支持  public static final int EXIST = -3;//已存在  public static final int INNER = -4;//内部错误  public static final int NOT\_FOUND = -5;//没有找到  public static final int WIFI\_DISABLED = -6;//wifi没有开启  public static final int SSID\_EMPTY = -7;//SSID为空  public static final int CONFIG\_FREQ = -8;//一键配置，不支持的wifi频率  public static final int OVERTIME = -9;//超时  public static final int PROCESSING = -10;//正在处理  public static final int NET = -11;//网络问题  public static final int NOT\_INIT = -12;//未初始化  public static final int NOT\_LOGIN = -13;//尚未登录  public static final int OFF\_LINE = -14;//设备离线  public static final int PARAM\_INVALID = -15;//无效参数  public static final int BAD\_SN = -16;//错误序列号  public static final int BAD\_NICKNAME = -17;//错误昵称  public static final int BAD\_PASSWD = -18;//密码错误  public static final int BAD\_PHONE\_NUM = -19;//手机号错误  public static final int SERVER\_BUSY = -20;//服务器忙  public static final int CLONE = -21;//克隆机 |
| Com.galaxywind.wukit.clibinterface.ClibXYWkq | public class ClibXYWkq {  // 状态参数  public byte onoff;// 0:关闭，1:开启  public byte mode;// 1:恒温，2:智能，3:休假  public short root\_temp;// 室温，单位0.1℃  public short di\_temp;// 地温，发热体温度  public byte cur\_dst\_temp;// 当前目标温度  public byte heat;// 是否加热中  public byte lock;// 是否锁定中  public byte err;// 是否有错误  public byte cons\_temp;// 恒温模式目标温度  public byte holiday\_temp;// 休假模式目标温度  public short remain\_time;// 剩余休假时间单位，小时  //上次发送命令，用来过滤返回的数据更新  public byte last\_cmd;  // 本地区室外温度  public byte extern\_temp;  // 探头故障  public byte probe\_err;  //智能回家开关  public byte smarthome\_onoff;  public ClibXYWkqAdjust adjust;// 校正参数  // 智能模式配置参数,两个字段配合取得一周7\*48个节点的温度配置  public ClibXYWkqSmartMode smart\_mode;  ｝ |
| Com.galaxywind.wukit.clibinterface.ClibXYWkqAdjust | public class ClibXYWkqAdjust{  public byte temp\_adj;//温度校正值  public byte temp\_top;//可设定温度上限（5~85）  public byte temp\_tolr;//温控容差(1-9)  public byte temp\_otemp;//过热保护温度（15~85）  public byte temp\_prottrig;//加热器保护触发时间（1~100）  public byte temp\_protlen;//加热器保护时长（10~90）  ｝ |
| Com.galaxywind.wukit.clibinterface.ClibXYWkqSmartMode | /\*\*  \* 智能模式设置参数说明：  \* 以7天为一个周期，每半个小时为一个节点，每天有48个节点，每天最多可以设置6个定时区间。智能模式里面又有三种模式：分别对应 每天、工作日+周末、工作日+周六+周日.  \* 每一个节点都可以设置开始时间，结束时间，温度，是否生效这四个字段.  \* timepoint的记录条数就应该是：7\*6\*3 = 126条  \* 第一种模式：每天 对应0 -42条 第二种模式：工作日+周末 对应43-84条，第三种模式：工作日+周六+周日 对应 85-126条  \* 一周7天，一天六个定时区间。按时间先后顺序对应在timepoint的记录：星期一 0-6条，星期二7-12条......  \* 一天有6个定时区间：对应0-24小时。具体逻辑可以自己设计。建议：按顺序设置定时区间，当前一个定时区间生效的情况下，后面一个定时区间才能生效。当前一个定时区间的  \* 结束时间已经到24时了，后一个定时区间不生效。  \*/  public class ClibXYWkqSmartMode{  public byte mode;  public ClibXYWkqTp[] timepoint;  ｝ |
| Com.galaxywind.wukit.clibinterface.ClibXYWkqTp | public class ClibXYWkqTp{  public byte valid;//是否生效  public byte start\_index;//开始时间  public byte end\_index;//结束时间  public byte temp;//区域时间内的温度设定  ｝ |