



微信硬件云设备直连 SDK 使用说明

SDK 接口说明、业务数据示例

发布时间：2016 - x - x

版本：beta

版本记录

| 作者 | 发布日期 | 版本 | 备注 |
|---------|------------|-----|---|
| zorrowu | 2015-12-24 | 0.1 | 初始文档起草 |
| zorrowu | 2015/12/28 | 0.2 | 重新排版，按功能划分模块 |
| zorrowu | 2015-12-29 | 0.3 | 增加嵌入式平台，增加 QCA4004 示例 |
| zorrowu | 2016/1/18 | 0.4 | 更新 Android 接口，增加设备信息 API |
| zorrowu | 2016/1/19 | 0.5 | 更新嵌入式接口，更新需要实现的外部函数，增加事件通知接口 |
| zorrowu | 2016/2/23 | 0.6 | 增加获取版本接口、C++版本增加 C 调用接口、C++接口返回的 String 类型改为 const char * |
| zorrowu | 2016/3/2 | 0.7 | 取消异步反馈 |

评审记录

| 评审人 | 评审时间 | 评审内容 | 评审意见 |
|-----|------|------|------|
| | | | |
| | | | |

目标读者

设备厂商开发者，微信硬件应用开发者，测试工程师

目录

| | |
|--------------------------------|-----------|
| 术语说明 | 4 |
| 接入须知 | 5 |
| 1 通信数据流说明 | 6 |
| 1.1 设备主动上报数据 | 6 |
| 1.2 服务器推送控制请求 | 8 |
| 1.3 服务器推送查询请求 | 10 |
| 2 各平台 SDK 接口说明 | 12 |
| 2.1 ANDROID 版本使用说明 | 12 |
| 2.1.1 加载动态库 | 14 |
| 2.1.2 初始化启动 SDK | 14 |
| 2.1.3 API 接口说明 | 14 |
| 2.2 LINUX 等 C++ 平台版本 | 18 |
| 2.2.1 SDK 文件组成说明 | 18 |
| 2.2.2 初始化启动 SDK | 19 |
| 2.2.3 C++API 接口说明 | 20 |
| 2.2.4 C 语言 API 接口说明 | 25 |
| 2.3 嵌入式 C 版本使用说明 | 30 |
| 2.3.1 SDK 文件组成说明 | 30 |
| 2.3.2 初始化启动 SDK | 31 |
| 2.3.3 API 接口说明 | 33 |
| 2.3.4 SDK 在不同平台的 Porting | 38 |
| 3 附录 | 45 |
| 3.1 ERROR_CODE | 45 |
| 3.2 ASY_ERROR_CODE | 45 |
| 3.3 微信硬件云平台业务 FUNCID | 45 |

声明

本文档目前仍处于密集修订待发布状态，SDK 也处于快速迭代中，**API 命名及参数类型等有可能进行改动**，所有内容仅供预先讨论、培训和技术交流用途。在文档正式发布前，**暂不适合用于批量生产的设备开发**。

术语说明

设备直连 SDK

微信硬件平台对外提供的设备 SDK 库，为了描述方便，以下简称 SDK。

设备厂商

采用微信硬件直连 SDK 开发设备的厂家。

设备应用层

从 SDK 的角度看，设备的软件包括两个部分，设备应用层和微信硬件直连 SDK，其中设备应用层由设备厂商开发，通过调用 SDK 的 API 来实现功能。

设备直连

指硬件厂商采用本 SDK 后，通过接口传入硬件设备信息后，在设备正常联网的情况下，SDK 就能够自动登录上微信硬件云平台，厂家需要上报数据到微信，只需要调用接口传入数据即可，SDK 会通过回调接口返回结果，同时 SDK 收到微信硬件云推送下来的消息时也会通过回调接口通知厂家。

微信硬件云

运行微信硬件相关服务、接口，提供设备接入、云端 API 和数据交换等服务的微信硬件云平台。

SDK 的 Porting

指要在目标平台上运行 SDK 时，需要做的适配工作，主要是一些外部功能函数的实现。

接入须知

目前 SDK 属于测试阶段，使用过程中有任何问题可以发邮件反馈：

收件人：wxthings-dev@foxmail.com

标题：微信硬件设备直连 SDK 内测反馈

邮件正文需要提供的信息有：

1. 厂商帐号，即目前的 deviceType，注意后面命名可能会变动。
2. 设备 ID，在微信硬件平台进行过授权，并运行于 SDK 中的 deviceId，注意没有在平台授权过的设备，SDK 将无法正常工作。
3. 使用的平台，目前 SDK 有多个平台的版本，反馈问题时说明一下所使用的平台及版本。
4. 出现场景，如果有重现的流程及场景的话请告知，越详细的信息能够帮助我们快速定位和解决问题

需要申请直连 SDK 库的厂家，请填写申请表格并做为附件发送到以下的电子邮件地址：

收件人：wxthings@foxmail.com

标题：微信硬件设备直连 SDK 静态库申请

1 通信数据流说明

目前设备直连 SDK 主要为硬件创建了上行和下行的数据通道。设备应用层不需要处理登录,鉴权等网络逻辑,只需要负责按微信硬件云平台的格式生成数据并调用 SDK 接口上报数据,同时解析和处理服务器返回或推送下来的业务数据即可。本章节将对设备上报的数据格式、服务响应数据格式、服务器推送数据格式进行说明,在下一章节中将对不同平台的 SDK 使用方式进行说明。

由于微信硬件云提供的业务较多,同时不同业务间的数据格式不一致,为了实现 SDK 对不同业务之间的兼容性,SDK 给微信硬件云平台的各项业务分配了唯一的 ID,本文所描述的是微信硬件云平台设备能力项业务,对应的 ID 为 1,厂家在调用接口或收到消息时通过业务 ID 来使用对应的数据格式解析报文。

1.1 设备主动上报数据

在设备联网,并且 SDK 处于运行状态下时,设备应用层可以通过 SDK 的发送数据接口上报数据给微信服务器。本章节使用的接口名称只为介绍方便,具体的接口格式请参照各平台 SDK 的介绍部分。在上报数据接口中需要传入设备所使用的业务 ID(如本文描述的设备能力项业务的 ID 为 1),以及要发送的数据内容,注意数据内容的格式要按业务要求进行封装。调用上报数据的接口如果返回值为 0 表示创建发送任务失败,很可能是没有调用 SDK 的初始化接口导致,下面以设备能力项业务为例进行说明,上行通道中的数据主要包含两种,一是设备主动上报一些状态,二是设备在收到服务器推送下来的通知后上报响应数据给服务器。设备主动上报的示例格式如下,注意本文所有的数据内容仅仅作为举例说明,详细的数据格式,支持的数据类型请结合参考文档《微信硬件设备云端数据接口》,《微信硬件产品能力定义指引》:

```

{
  "msg_type" : "notify" ,
  "services" : {
    "operation_status" : {
      "status" : 1
    },
    "air_conditioner" : {
      "tempe_indoor" : 26,
      "tempe_outdoor" : 31,
      "tempe_target" : 26,
      "fan_speed" : 50
    },
    .....
  },
  .....
}

```

微信硬件云端响应 JSON 格式包，里面带有本次 notify 的唯一标识 msg_id，设备收到下面的回包时表示服务器已经承接了设备的该项请求。SDK 会通过 onResponseCallback(int taskid, int errcode, int funcid, byte[] data)；接口将服务器的回包返回给设备应用层，taskid 为调用发送数据接口时的返回值，errcode 为 SDK 链路层的错误码，跟业务(body)中的数据无关，成功时为 0，funcid 为设备能力项业务 ID，本例中为 1，data 即为下面示例中的内容：

```

{
  "error_code" : 0,
  "error_msg" : "ok" ,
  "msg_id" : 1234567890123456
}

```

| 标签 | 类型 | 取值限制 | 含义 |
|----|----|------|----|
|----|----|------|----|

| | | | |
|------------------|--------|--------|--|
| msg_type | string | notify | 消息类型命令字，notify 代表通知消息 |
| services | string | | 能力项键值集合，即微信硬件云平台规范的能力项键值 |
| operation_status | string | | 通知微信侧，必须带有运行状态能力项服务 |
| status | int8 | | 通知微信侧，必须带有运行状态能力项属性 |
| air_conditioner | string | | 微信硬件云平台规范的一项能力及其属性值，其他服务及完整的属性参考《产品能力定义指引》 |
| error_code | int16 | | 微信是否成功接收，0 代码成功（其他错误码见附录） |
| error_msg | string | | 微信返回成功接收的信息 |
| msg_id | int64 | 64 位整型 | 本消息序列号，用于异步通信 |

1.2 服务器推送控制请求

对于服务器发送过来的请求，SDK 会通过 onNotifyCallback 接口通知设备应用层，如下是微信硬件云推送下来的控制设备的请求：

```

{
  "msg_id" : 1234567890123456,
  "user" : "user" ,
  "msg_type" : "set" ,
  "services" : {
    "air_conditioner" : {
      "tempe_indoor" : 26,
      "tempe_outdoor" : 31,
      "tempe_target" : 26,
      "fan_speed" : 50
      ....
    }
    ...
  }
}

```

| 标签 | 类型 | 取值限制 | 含义 |
|----------|--------|--------|---------------------------------|
| msg_id | int64 | 64 位整型 | 消息序列号，用于异步通信，由微信生成，接收方异步返回的时候带上 |
| user | string | 无 | operator 操作者唯一标识 ID |
| msg_type | string | set | 消息类型命令字，set 代表设备控制消息 |
| services | string | | 能力项键集合 |

设备应用层收到上述控制命令以后，解析 services 中的字段，并按要求控制设备，完成控制以后，调用 sendDataToServer 接口上报结果，上报内容示例如下：

```

{
  "asy_error_code" : 0,
  "asy_error_msg" : "ok" ,
  "msg_id" : 1234567890123456,
  "msg_type" : "set" ,
  "services" : {
    "operation_status" : {
      "status" : 1
    }
  },
  .....
}

```

| 标签 | 类型 | 取值限制 | 含义 |
|------------------|--------|--------------------------|---------------------|
| asy_error_code | int16 | | 设备厂商异步设置结果，其他错误请见附录 |
| asy_error_msg | string | | 设备厂商异步查询结果的，消息文本描述 |
| msg_id | int64 | 和请求体里面 msgi_id 一致 | |
| msg_type | string | 和请求里面的 msg_type 一样,为 set | |
| services | string | | 能力项键集合（定义详见附录） |
| operation_status | string | | 必须返回运行状态能力项服务 |
| status | int8 | | 必须返回运行状态能力项属性 |

1.3 服务器推送查询请求

在某些情况下，微信硬件云需要查询设备的状态信息，这个时候微信硬件云推送下来的获取设备状态的报文格式如下：

```
{
  "msg_id" : 1234567890123456,
  "user" : "user" ,
  "msg_type" : "get" ,
  "services" : {
    "operation_status" : {
      "status" : 0
    },
    "air_conditioner" : {
      "tempe_indoor" : 0,
      "tempe_outdoor" : 0,
      "tempe_target" : 0,
      "fan_speed" : 0
    },
    .....
  },
  ...
}
```

设备接收到该请求以后，根据请求的 services 内容打包数据，并调用 sendDataToServer 接口上报结果，示例如下：

```
{  
  "msg_id" : 1234567890123456,  
  "user" : "user" ,  
  "msg_type" : "get" ,  
  "services" : {  
    "operation_status" : {  
      "status" : 0  
    },  
    "air_conditioner" : {  
      "tempe_indoor" : 26,  
      "tempe_outdoor" : 31,  
      "tempe_target" : 26,  
      "fan_speed" : 50  
      .....  
    }  
  }  
}
```

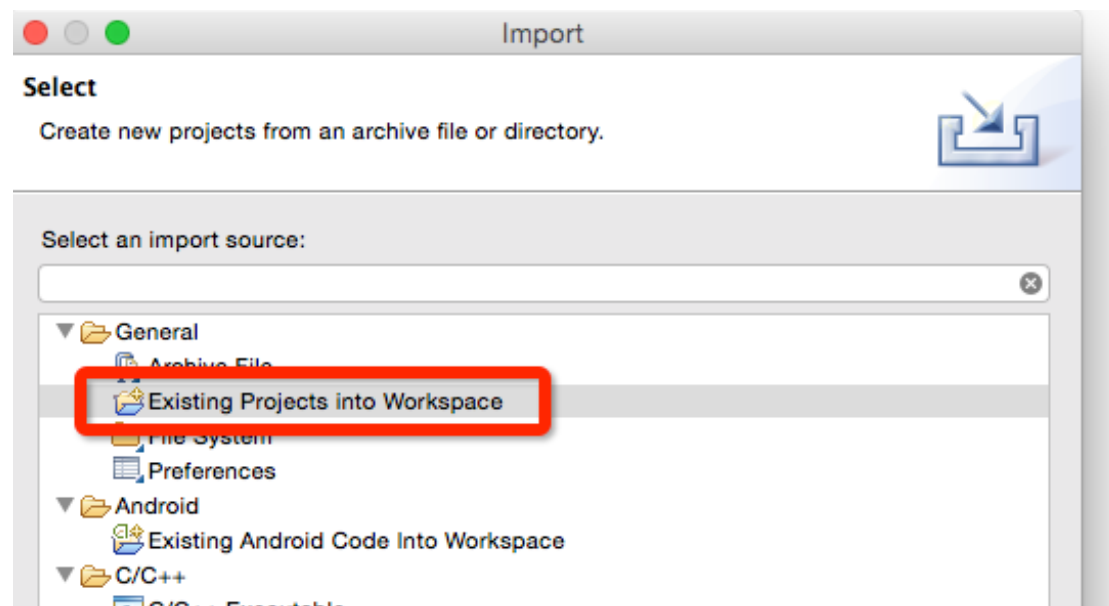
2 各平台 SDK 接口说明

目前微信硬件云平台提供了多个平台的直连 SDK，设备直连 SDK 主要为硬件创建了上行和下行的数据通道，设备应用层只需要处理业务流数据，并执行对设备的控制等操作即可。

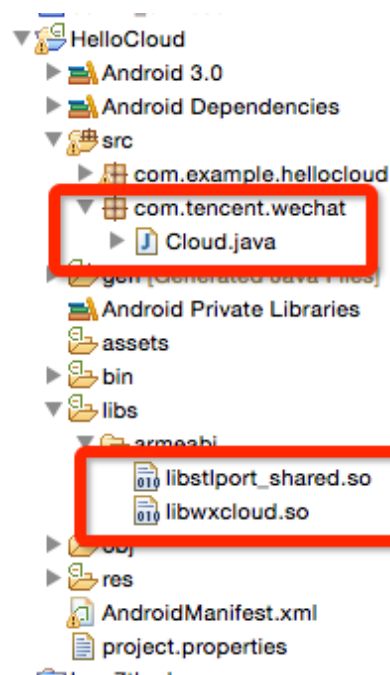
2.1 Android 版本使用说明

对于运行于 Android 系统的设备，微信提供对应的 SDK（.so 库）且不依赖于

设备所使用的 Android 版本 ,开发者只需要将 SDK 动态库和 API 类文件加载到 App 中即可 , 目前文件包括 : libwxcloud.so , libstlport_shared.so , /com.tencent/wechat/Cloud.java 共 3 个文件 , 注意不得修改包名及方法名 否则动态库无法使用 , SDK 运行中会通过 Logcat 打印数据 , 使用过程中遇到问题可以把 log 导出。详细的工程和代码示例请参考 android_demo 工程 , 可以通过 Eclipse 直接 import 的方式导入 :



动态库和接口文件在工程中的结构如下 :



2.1.1加载动态库

由于 SDK 需要用到网络，如果是新建工程记得在 xml 中添加网络权限。设备 App 启动以后，需要加载对应的动态库，如果 App 有 Application 入口，可以放在 Application 入口处：

```
static {
    System.loadLibrary("stlport_shared");
    System.loadLibrary("wxcloud");
}
```

2.1.2初始化启动 SDK

App 启动后，通过 Cloud.init(deviceLicence);接口初始化并启动 SDK，其中 deviceLicence 为直连设备在微信后台授权注册时由微信生成并返回的设备证书，格式为 String，详细信息请查阅微信后台设备授权注册接口，**每个设备都有唯一的一个证书，每个证书只能在一台设备上使用**：

```
//请根据实际情况传递参数，否则程序无法正确运行
String testdevlicence = "XXXXXXXXXXXXXXXXXXXXXXXXXXXX";
if (Cloud.init(testdevlicence)) {
    Log.d(LOG_TAG, "SDK is running!");
    sdk_run_state = true;
}
else {
    Log.e(LOG_TAG, "Device licence error!!!");
}
```

2.1.3API 接口说明

初始化启动 SDK

| | | |
|------|------------------------------------|--|
| 接口原型 | public static native boolean | |
|------|------------------------------------|--|

| | | |
|-----|-----------------------------|----------------------------|
| | init(String deviceLicence); | |
| 参数 | String deviceLicence, | 设备证书，由微信后台生成并返回，查看设备授权注册接口 |
| 返回值 | True/False | 初始化成功/失败 |

发送数据给服务器

| | | |
|------|---|--|
| 接口原型 | public static native int sendDataToServer(int funcid, String body); | |
| 参数 | int funcid, | 微信硬件云平台业务 ID，能力服务项业务为 1 |
| | String body, | 要发送的数据内容，按业务要求填充报文 |
| 返回值 | 0/taskid | 返回 0 表示创建上报任务失败，可能是没有调用初始化接口，非 0 的话为 SDK 返回给设备应用层用于标识该任务的 taskid， 注意固件更新检查有专门 API，不通过本 API。 |

检查是否有新固件

| | | |
|------|---|-------------------------------|
| 接口原型 | public static native int checkUpdate(String body); | 微信硬件平台提供的固件管理业务 API |
| 参数 | String body, | 当前的版本信息，按微信硬件平台业务要求填充报文，示例：暂无 |

| | | |
|-----|----------|---|
| 返回值 | 0/taskid | 返回 0 表示创建上报任务失败，可能是没有调用初始化接口，非 0 的话为 SDK 返回给设备应用层用于标识该任务的 taskid。 |
|-----|----------|---|

获取当前证书对应的厂商 ID

| | | |
|------|---|---|
| 接口原型 | <code>public static native String getVenderId();</code> | |
| 参数 | 无 | 调用 init 接口后可以再调用本接口获取当前 Licence 对应的 VenderID ,之前命名为 deviceType |
| 返回值 | 字符串 | 返回空字符串表示证书无效 |

获取当前证书对应的设备 ID

| | | |
|------|---|--|
| 接口原型 | <code>public static native String getDeviceId();</code> | |
| 参数 | 无 | 调用 init 接口后可以再调用本接口获取当前 Licence 对应的 DeviceID |
| 返回值 | 字符串 | 返回空字符串表示证书无效 |

获取当前直连 SDK 的版本

| | | |
|------|---|--------------|
| 接口原型 | <code>public static native String getSDKVersion();</code> | |
| 参数 | 无 | |
| 返回值 | 字符串 | 反馈问题时需要提供该信息 |

关闭微信直连 SDK

| | | |
|------|--------------------------------------|--|
| 接口原型 | public static native void release(); | |
| 参数 | 无 | |
| 返回值 | 无 | |

任务执行结果回调接口

| | | |
|------|---|-----------------------|
| 接口原型 | public static void onResponseCallback(int taskid, int errcode, int funcid, byte[] data) | |
| 参数 | int taskid, | 该响应对应的任务 id |
| | int errcode, | 0 的话表示网络链路层正常，非 0 为失败 |
| | int funcid | 标识数据是属于微信硬件云平台的哪个业务的。 |
| | byte[] data | 数据内容 |
| 返回值 | 无 | |

服务器推送消息回调接口

| | | |
|------|--|-----------------------|
| 接口原型 | public static void onNotifyCallback(int funcid, byte[] data) | |
| 参数 | int funcid | 标识数据是属于微信硬件云平台的哪个业务的。 |
| | byte[] data | 数据内容 |

| | | |
|-----|---|--|
| 返回值 | 无 | |
|-----|---|--|

SDK 事件通知回调接口

| | | |
|------|--|---|
| 接口原型 | public static void onEventCallback(int event) | |
| 参数 | int event | 事件类型，取值范围： private static int EVENT_VALUE_LOGIN = 1; private static int EVENT_VALUE_LOGOUT = 2; |
| 返回值 | 无 | |

2.2 Linux 等 C++ 平台版本

对于运行 Linux 系统的设备，微信提供对应的 C++ 版本静态库，由于 Linux 版本众多，编译工具链也不一，对于不在支持列表中的平台，可以下载对应的申请表填写后发到申请表中的邮箱地址进行申请，一般运行环境需要支持-lrt 和 -lpthread 两个外部库，同时本版本静态库也提供 C 的调用接口，方便使用 C 的开发者进行开发。

2.2.1 SDK 文件组成说明

目前 SDK 提供的文件主要包括以下几个：

WeChatAPI.h：

SDK 提供给设备应用层调用的 API 接口，包括 SDK 初始化，回调函数注册，发送数据等接口。

WeChatAPI_C.h：

SDK 提供给设备应用层调用的 API 接口 (C 接口), 包括 SDK 初始化, 回调函数注册, 发送数据等接口, 如果开发者应用层代码是用 C 写的则可以包含本头文件进行开发。

libwxcloud.a :

SDK 静态库文件, 需要链接到工程中。

2.2.2初始化启动 SDK

在设备连接上路由器以后, 调用 WeChatAPI::instance()->start ()接口完成 SDK 的初始化, 该 API 返回 bool 型, false 的话一般是设备 Licence 无效。为了接收相关的回调, 需要注册 3 个回调函数, 通过 WeChatAPI::instance()->setCallBack , WeChatAPI::instance()->setNotifyCallBack , WeChatAPI::instance()->setSDKEventCallBack 这三个函数实现注册。至此在设备证书合法的情况下, SDK 就开始正常运行了, 示例代码如下:

```
std::cout << "Try to start plugins!" << std::endl;
WeChatAPI::instance()->start(devlicence.c_str(), devlicence.size());
WeChatAPI::instance()->setCallBack(onWeChatCallBack);
WeChatAPI::instance()->setNotifyCallBack(onRecevNotify);
WeChatAPI::instance()->setSDKEventCallBack(onHandleEvent);
std::cout << "VenderID:" << WeChatAPI::instance()->getVenderId() << std::endl;
std::cout << "DeviceID:" << WeChatAPI::instance()->getDeviceId() << std::endl;
```

如果设备需要向服务器发送业务数据, 则调用 WeChatAPI::instance()->sendDataToServer 接口, 第一个参数为业务 ID (注意固件升级检查不能用此 API), 可以参考附录最后的表格, 目前 WeChatAPI::instance()->checkUpdate 是微信硬件平台提供的固件管理业务, 用于检查是否有新固件。

对应的 C 接口调用示例代码如下:

```

printf("Try to start plugins!\r\n");
if (false == WeChatAPI_start(devlicence, strlen(devlicence))) {
    printf("Start SDK failed check licence!\r\n");
    return -1;
}
WeChatAPI_setCallBack(onWeChatCallBack);
WeChatAPI_setNotifyCallBack(onRecevNotify);
WeChatAPI_setSDKEventCallBack(onHandleEvent);
printf("SDK Version:%s!\r\n", WeChatAPI_getSDKVersion());
printf("VenderID:%s!\r\n", WeChatAPI_getVenderId());
printf("DeviceID:%s!\r\n", WeChatAPI_getDeviceId());

sleep(1);
taskid = WeChatAPI_sendDataToServer(WECHAT_CLOUD_SERVICE, state, strlen(state));
printf("task id is:%d\r\n", taskid);

```

2.2.3 C++ API 接口说明

初始化启动 SDK

| | | |
|------|--|----------------------------|
| 接口原型 | bool start(const char* devlicence, unsigned int licencelen); | |
| 参数 | const char* devlicence | 设备证书，由微信后台生成并返回，查看设备授权注册接口 |
| | unsigned int licencelen | 证书长度 |
| 返回值 | True/False | 初始化成功/失败 |

发送数据给服务器

| | | |
|------|---|-------------------------|
| 接口原型 | int sendDataToServer(unsigned int funcid, const unsigned char* body, unsigned int bodylen); | |
| 参数 | unsigned int funcid, | 微信硬件云平台业务 ID，能力服务项业务为 1 |
| | const unsigned char* body | 要发送的数据内容，按业务要求填充报文 |
| | unsigned int bodylen | 报文长度 |

| | | |
|-----|----------|--|
| 返回值 | 0/taskid | 返回 0 表示创建上报任务失败，可能是没有调用初始化接口，非 0 的话为 SDK 返回给设备应用层用于标识该任务的 taskid， 注意固件更新检查有专门 API，不通过本 API。 |
|-----|----------|--|

检查是否有新固件

| | | |
|------|---|---|
| 接口原型 | int checkUpdate(const unsigned char* body, unsigned int bodylen); | 微信硬件平台提供的固件管理业务 API |
| 参数 | const unsigned char* body, | 当前的版本信息，按微信硬件平台业务要求填充报文，示例：暂无 |
| | unsigned int bodylen | 数据长度 |
| 返回值 | 0/taskid | 返回 0 表示创建上报任务失败，可能是没有调用初始化接口，非 0 的话为 SDK 返回给设备应用层用于标识该任务的 taskid。 |

获取当前证书对应的厂商 ID

| | | |
|------|----------------------------|--|
| 接口原型 | const char* getVenderId(); | |
| 参数 | 无 | 调用 start 接口后可以再调用本接口获取当前 Licence 对应的 VenderID， 之前命名为 deviceType |
| 返回值 | 字符串 | 返回空字符串表示证书无效 |

获取当前证书对应的设备 ID

| | | |
|------|----------------------------|---|
| 接口原型 | const char* getDeviceId(); | |
| 参数 | 无 | 调用 start 接口后可以再调用本接口获取当前 Licence 对应的 DeviceID |
| 返回值 | 字符串 | 返回空字符串表示证书无效 |

获取当前 SDK 的版本信息

| | | |
|------|------------------------------|--|
| 接口原型 | const char* getSDKVersion(); | |
| 参数 | 无 | |
| 返回值 | 字符串 | |

关闭微信直连 SDK

| | | |
|------|-----------------|--|
| 接口原型 | void release(); | |
| 参数 | 无 | |
| 返回值 | 无 | |

注册任务执行结果回调接口

| | | |
|------|---|------------------------|
| 接口原型 | void setCallBack(onReceiveResponse _callback); | 所有的 task 都是通过本接口 返回 |
| 参数 | onReceiveResponse _callback | 回调函数入口地址 |
| 返回值 | 无 | |
| 示例 | void onWeChatCallBack(int taskid, int errcode, unsigned int funcid, const unsigned char* body, unsigned int bodylen) { | |

| | |
|----|---|
| | <pre> std::cout << "Receive WeChat CallBack!" << std::endl; std::cout << " taskid = " << taskid << ", errcode = " << errcode << ", funcid = " << funcid << ", body = " << body << ", bodylen = " << bodylen << std::endl; std::cout << "End of WeChat CallBack!" << std::endl; } WeChatAPI::instance()->setCallBack(onWeChatCallBack); </pre> |
| 说明 | <p>注意不要在 onWeChatCallBack 做数据处理或阻塞的操作，因为该函数将运行于 SDK 的线程中，建议将数据拷贝到自己的线程中去处理，taskid 为任务 ID，errcode 为链路层错误码，为 0 表示正常，funcid 为微信业务 ID，可参考最后附录，body 是具体的数据，bodylen 是数据长度。</p> |

服务器推送消息回调接口

| | | |
|------|---|----------|
| 接口原型 | void setNotifyCallBack(onReceiveNotify_callback); | |
| 参数 | onReceiveNotify_callback | 回调函数入口地址 |
| 返回值 | 无 | |
| 示例 | <pre> void onRecevNotify(unsigned int funcid, const unsigned char* _body, unsigned int _bodylen) { std::string body = std::string((const char *)_body, _bodylen); std::cout << "Receive Notify funcid:" << funcid << ", body:" << _body << ", len:" << _bodylen << std::endl; } WeChatAPI::instance()->setNotifyCallBack(onRecevNotify); </pre> | |

| | |
|----|---|
| 说明 | 注意不要在 onRecevNotify 做数据处理或阻塞的操作，因为该函数将运行于 SDK 的线程中，建议将数据拷贝到自己的线程中去处理，funcid 为微信业务 ID，可参考最后附录，_body 是具体的数据，_bodylen 是数据长度。 |
|----|---|

SDK 事件通知回调接口

| | | |
|------|---|----------|
| 接口原型 | void setSDKEventCallBack(onSDKEvent Callback _callback); | |
| 参数 | onSDKEventCallback _callback | 回调函数入口地址 |
| 返回值 | 无 | |
| 示例 | <pre> void onHandleEvent (EventValue event_value) { std::cout << "Receive Event:" << event_value << std::endl; switch (event_value) { case EVENT_VALUE_LOGIN: std::cout << "Device login!!" << std::endl; break; case EVENT_VALUE_LOGOUT: std::cout << "Device logout!!" << std::endl; break; default: std::cout << "Unknown event!!" << std::endl; break; } } WeChatAPI::instance()->setSDKEventCallBack(onHandleEvent); </pre> | |

| | |
|----|--|
| | <pre>enum EventValue { EVENT_VALUE_LOGIN = 1, EVENT_VALUE_LOGOUT = 2 };</pre> |
| 说明 | 注意不要在 onRecevNotify 做数据处理或阻塞的操作，因为该函数将运行于 SDK 的线程中，建议将数据拷贝到自己的线程中去处理，event_value 为对应的事件。 |

2.2.4C 语言 API 接口说明

初始化启动 SDK

| | | |
|------|---|----------------------------|
| 接口原型 | <pre>bool WeChatAPI_start(const char* devlicence, unsigned int licencelen);</pre> | |
| 参数 | const char* devlicence | 设备证书，由微信后台生成并返回，查看设备授权注册接口 |
| | unsigned int licencelen | 证书长度 |
| 返回值 | True/False | 初始化成功/失败 |

发送数据给服务器

| | | |
|------|---|----------------|
| 接口原型 | <pre>int WeChatAPI_sendDataToServer (unsigned int funcid, const unsigned char* body, unsigned int bodylen);</pre> | |
| 参数 | unsigned int funcid, | 微信硬件云平台业务 ID，能 |

| | | |
|--|---------------------------|--------------------|
| | | 力服务项业务为 1 |
| | const unsigned char* body | 要发送的数据内容，按业务要求填充报文 |
| | unsigned int bodylen | 报文长度 |

检查是否有新固件

| | | |
|------|--|---|
| 接口原型 | int WeChatAPI_checkUpdate (const unsigned char* body, unsigned int bodylen); | 微信硬件平台提供的固件管理业务 API |
| 参数 | const unsigned char* body, | 当前的版本信息，按微信硬件平台业务要求填充报文，示例：暂无 |
| | unsigned int bodylen | 数据长度 |
| 返回值 | 0/taskid | 返回 0 表示创建上报任务失败，可能是没有调用初始化接口，非 0 的话为 SDK 返回给设备应用层用于标识该任务的 taskid。 |

获取当前证书对应的厂商 ID

| | | |
|------|---|---|
| 接口原型 | const char* WeChatAPI_getVenderId(); | |
| 参数 | 无 | 调用 start 接口后可以再调用本接口获取当前 Licence 对应的 VenderID ,之前命名为 deviceType |
| 返回值 | 字符串 | 返回空字符串表示证书无效 |

获取当前证书对应的设备 ID

| | | |
|------|--------------------------------------|---|
| 接口原型 | const char* WeChatAPI_getDeviceId(); | |
| 参数 | 无 | 调用 start 接口后可以再调用本接口获取当前 Licence 对应的 DeviceID |
| 返回值 | 字符串 | 返回空字符串表示证书无效 |

获取当前 SDK 的版本信息

| | | |
|------|--|--|
| 接口原型 | const char* WeChatAPI_getSDKVersion(); | |
| 参数 | 无 | |
| 返回值 | 字符串 | |

关闭微信直连 SDK

| | | |
|------|----------------------------|--|
| 接口原型 | void WeChatAPI_release (); | |
| 参数 | 无 | |
| 返回值 | 无 | |

注册任务执行结果回调接口

| | | |
|------|--|--------------------|
| 接口原型 | void WeChatAPI_setCallBack (onReceiveResponse _callback); | 所有的 task 都是通过本接口返回 |
| 参数 | onReceiveResponse _callback | 回调函数入口地址 |
| 返回值 | 无 | |
| 示例 | void onWeChatCallBack(int taskid, int errcode, unsigned int funcid, const unsigned char* body, unsigned int bodylen) { | |

| | |
|----|--|
| | <pre>printf("Receive WeChat CallBack!\r\n"); printf("hashCode=%d, errcode=%d, funcid=%d, bodylen=%d!\r\n", hashCode, errcode, funcid, bodylen); } WeChatAPI_setCallBack(onWeChatCallBack);</pre> |
| 说明 | <p>注意不要在 onWeChatCallBack 做数据处理或阻塞的操作，因为该函数将运行于 SDK 的线程中，建议将数据拷贝到自己的线程中去处理，taskid 为任务 ID，errcode 为链路层错误码，为 0 表示正常，funcid 为微信业务 ID，可参考最后附录，body 是具体的数据，bodylen 是数据长度。</p> |

服务器推送消息回调接口

| | | |
|------|---|----------|
| 接口原型 | <pre>void WeChatAPI_setNotifyCallBack (onReceiveNotify _callback);</pre> | |
| 参数 | onReceiveNotify _callback | 回调函数入口地址 |
| 返回值 | 无 | |
| 示例 | <pre>void onRecevNotify(unsigned int funcid, const unsigned char* _body, unsigned int _bodylen) { printf("Receive WeChat onRecevNotify!\r\n"); printf("funcid=%d, bodylen=%d!\r\n", funcid, _bodylen); } WeChatAPI_setNotifyCallBack(onRecevNotify);</pre> | |
| 说明 | <p>注意不要在 onRecevNotify 做数据处理或阻塞的操作，因为该函数将运行于 SDK 的线程中，建议将数据拷贝到自己的线程中去处理，funcid 为微信业务 ID，可参考最后附录，_body 是具体的数据，_bodylen 是数据长度。</p> | |

SDK 事件通知回调接口

| | | |
|------|--|----------|
| 接口原型 | <code>void WeChatAPI_setSDKEventCallBack (onSDKEventCallback _callback);</code> | |
| 参数 | <code>onSDKEventCallback _callback</code> | 回调函数入口地址 |
| 返回值 | 无 | |
| 示例 | <pre>void onHandleEvent (enum EventValue event_value) { printf("Receive WeChat Event!\r\n"); switch (event_value) { case EVENT_VALUE_LOGIN: printf("Device login!!\r\n"); break; case EVENT_VALUE_LOGOUT: printf("Device logout!!\r\n"); break; default: printf("Unknown event!!\r\n"); break; } } WeChatAPI_setSDKEventCallBack(onHandleEvent); enum EventValue { EVENT_VALUE_LOGIN = 1, EVENT_VALUE_LOGOUT = 2 };</pre> | |

| | |
|----|--|
| 说明 | 注意不要在 onRecevNotify 做数据处理或阻塞的操作，因为该函数将运行于 SDK 的线程中，建议将数据拷贝到自己的线程中去处理，event_value 为对应的事件。 |
|----|--|

2.3 嵌入式 C 版本使用说明

对于运行嵌入式操作系统或者无操作系统的硬件平台，微信提供对应的静态库版本，对于不在支持列表中的平台，可以下载对应的申请表填写后发到申请表中的邮箱地址进行申请。

由于嵌入式平台种类较多，系统的资源和性能各异，微信硬件平台将核心逻辑代码进行了封装，将需要跨平台实现的代码通过外部函数的方式放出来，因此在将静态库添加到项目工程以后，开发者还需要根据 airkiss_porting.h 中定义的函数原型来实现函数实体，微信硬件平台也提供了几个平台的实现 Demo，开发者可以下载下来参考着实现，下面以 QCA4004 为示例平台进行说明。

2.3.1 SDK 文件组成说明

目前 SDK 提供的文件主要包括以下几个：

airkiss_types.h：

SDK 使用到的数据类型、数据结构定义头文件。

airkiss_cloudapi.h：

SDK 提供给设备应用层调用的 API 接口，包括 SDK 初始化，回调函数注册，发送数据等接口。

airkiss_porting.h：

SDK 使用到的外部函数的原型，开发者需要根据本文件定义的所有函数原型实现相应的函数，并添加到工程中，否则编译链接静态库的时候会报错找不到函数，如本 Demo 实现的 airkiss_porting_4004.c 文件。

libwxcloud.a：

SDK 静态库文件，需要链接到工程中。

2.3.2 初始化启动 SDK

在设备连接上路由器以后，调用 `airkiss_cloud_init()` 接口完成 SDK 的初始化，然后再调用 `airkiss_regist_callbacks()` 接口注册相关的回调函数，这两个步骤完成以后，开发者就可以通过 `airkiss_cloud_sendmessage()` 接口发送数据给服务器了。如果运行的环境支持多线程，那么开发者可以直接开一个线程来调用 `airkiss_cloud_loop()`，该函数的返回值为下一次期望调用 `airkiss_cloud_loop()` 的时间值，单位为毫秒(ms)。对于不支持多线程的系统，开发者可以在程序大循环中调用 `airkiss_cloud_loop()`，然后在函数返回后再去处理数据或运行自己的逻辑代码，示例代码如下所示：

```
#ifdef AIRKISS_SUPPORT_MULTITHREAD
while (0 != airkiss_cloud_init((uint8_t *)testdevlicence, (uint32_t)strlen((const char *)
testdevlicence), &m_task_mutex, &m_malloc_mutex, heapbuf, sizeof(heapbuf))) {
    A_PRINTF("SDK init failed!!!\r\n");
    qcom_thread_msleep(1000);
}
#else
while (0 != airkiss_cloud_init((uint8_t *)testdevlicence, (uint32_t)strlen((const char *)
testdevlicence), 0, 0, heapbuf, sizeof(heapbuf))) {
    A_PRINTF("SDK init failed!!!\r\n");
    qcom_thread_msleep(1000);
}
#endif

//regist callback functions
airkiss_callbacks_t cbs;
cbs.m_notifycb = ReceiveNotifyCB;
cbs.m_respcb = ReceiveResponseCB;
cbs.m_eventcb = ReceiveEventCB;
airkiss_regist_callbacks(&cbs);
```

本示例 Demo 开启了多线程，SDK 单独运行一个线程，示例代码如下，其中 `ak_loop_run_sign` 是一个全局变量，在设备成功连网以后会进行置位，充当信号的作用，实现比较简单，开发者可以用信号量或其他方式实现：

```
void airkiss_cloud_thread(ULONG which_thread) {
    uint32_t sleep_time;
    while(ak_loop_run_sign == 0) {
        qcom_thread_msleep(2000);
    }
    A_PRINTF("Everything is ready!!\r\n");
    for (;;) {
        sleep_time = airkiss_cloud_loop();
        qcom_thread_msleep(sleep_time);
    }
}
```


在设备应用线程中，每隔 15s 调用 `airkiss_cloud_sendmessage` 函数发起一次任务，示例代码如下：

```
#ifndef AIRKISS_SUPPORT_MULTITHREAD
//ready to run airkiss_cloud_loop
ak_loop_run_sign = 1;
for (;;) {
    A_PRINTF("App thread sleep!!\r\n");
    qcom_thread_msleep(15000);
    taskid = airkiss_cloud_sendmessage(1, (uint8_t *)("{\"Power\":1}", strlen("{\"Power\":1}")));
    A_PRINTF("Wake up from app thread and start a task:%d!\r\n", taskid);
}
#else
taskid = airkiss_cloud_sendmessage(1, (uint8_t *)("{\"Power\":1}", strlen("{\"Power\":1}")));
A_PRINTF("Start task finish:%d!\r\n", taskid);
for (;;) {
    airkiss_cloud_loop();
    qcom_thread_msleep(100);
}
#endif
```

数据内容为测试用，不是实际业务数据。对于不支持多线程的系统则直接可以在 `airkiss_cloud_loop()` 函数返回后调用 `airkiss_cloud_sendmessage` 函数发送数据。本示例的接收响应回调函数定义如下：

```
void ReceiveResponseCB(uint32_t taskid, uint32_t errcode, uint32_t funcid, const uint8_t* body, uint32_t bodylen) {
    A_PRINTF("Enter Resp Callback:id:%d, err:%d, funcid:%d, len:%d\r\n", taskid, errcode, funcid, bodylen);
    printfstrlen(body, bodylen);
    A_PRINTF("\r\n");
}
```

在多线程的环境下，由于 `ReceiveResponseCB` 函数实际运行的是 SDK 的线程，为了 SDK 运行正常，这里建议开发者只做数据拷贝工作，数据的具体解析和处理逻辑放到别的线程中去执行，如果是单线程的则没有关系。另外注意 `body` 指针指向的内容在函数返回后就会进行释放，所以多线程的话要对数据进行拷贝，不能复制指针。本示例的接收推送消息的回调函数定义如下：

```
void ReceiveNotifyCB(uint32_t funcid, const uint8_t* body, uint32_t bodylen) {
    A_PRINTF("Recv notify Callback funcid:%d, len:%d\r\nData:", funcid, bodylen);
    printfstrlen(body, bodylen);
    A_PRINTF("\r\n");
}
```

推送消息回调函数的处理方式与接收响应回调函数的处理方式一致。在 SDK 运行的过程中会发送一些事件消息给应用开发者，目前包括登录态，示例如下：

```

void ReceiveEventCB(EventValue event_value) {
    A_PRINTF("Recv Event Callback:%d\r\n", event_value);
    A_PRINTF("\r\n");
    switch (event_value) {
        case EVENT_VALUE_LOGIN:
            A_PRINTF("Device Login!\r\n");
            break;
        case EVENT_VALUE_LOGOUT:
            A_PRINTF("Device Logout!\r\n");
            break;
    }
}

```

2.3.3 API 接口说明

获取 SDK 版本信息

| | | |
|------|--------------------------------------|----------------|
| 接口原型 | const char* airkiss_cloud_version(); | |
| 参数 | 无 | |
| 返回值 | 以\0 结尾的字符串 | 可以直接打印,反馈问题需提供 |

初始化 SDK

| | | |
|------|--|-------------------------------------|
| 接口原型 | int32_t airkiss_cloud_init(uint8_t *devlicence, uint32_t licencelen, ak_mutex_t* ak_task_mt, ak_mutex_t* ak_mem_mt, void* heap, uint32_t heaplen); | |
| 参数 | uint8_t* devlicence | 设备证书,由微信后台生成并返回,查看设备授权注册接口 |
| | uint32_t licencelen | licence 的有效长度 |
| | ak_mutex_t* ak_task_mt, ak_mutex_t* ak_mem_mt, | 多线程方式运行 SDK 时提供常量指针,如本示例中的传入参数为全局变量 |

| | | |
|-----|------------------|--|
| | void* heap | 提供给 SDK 的堆空间起始地址,为了避免地址对齐问题, 建议定义一个全局的 int 数组 ,并把数组地址传入,为保证 SDK 正常运行,最少 4KB,4KB 情况下允许同时运行 2 个任务,8KB 运行同时运行 4 个任务,前提是每个任务的数据不要超过 1KB,空间用完启动任务会返回失败,需要等其他任务完成或超时才能继续启动新任务 |
| | uint32_t heaplen | 堆空间大小,以字节为单位 |
| 返回值 | 0 为成功,非 0 表示失败 | |

注册回调函数

| | | |
|------|---|----------------------|
| 接口原型 | void airkiss_regist_callbacks(airkiss_callbacks_t* _callbacks); | |
| 参数 | airkiss_callbacks_t* _callbacks | 回调函数的结构体指针,如: |
| | typedef struct { airkiss_onresponse_fn m_respcb; airkiss_onnitify_fn m_notifycb; airkiss_onsdkevent_fn m_eventcb; } airkiss_callbacks_t; | 回调函数实现原型示例及参数赋值见上一小节 |

| | | |
|-----|---|--|
| 返回值 | 无 | |
|-----|---|--|

任务执行结果回调接口实现示例

| | | |
|------|---|--|
| 接口原型 | void ReceiveResponseCB(uint32_t taskid, uint32_t errcode, uint32_t funcid, const uint8_t* body, uint32_t bodylen) | 函数名可以自定，返回值和参数类型固定，实现后把函数指针通过 airkiss_regist_callbacks 传给 SDK。 |
| 参数 | uint32_t taskid, | 该响应对应的任务 id |
| | uint32_t errcode, | 0 的话表示网络链路层正常，非 0 为失败 |
| | uint32_t funcid | 标识数据是属于微信硬件云平台的哪个业务的。 |
| | const uint8_t* body | 数据内容 |
| | uint32_t bodylen | 数据内容长度 |
| 返回值 | 无 | |

服务器推送消息回调接口实现示例

| | | |
|------|--|--|
| 接口原型 | void ReceiveNotifyCB(uint32_t funcid, const uint8_t* body, uint32_t bodylen) | 函数名可以自定，返回值和参数类型固定，实现后把函数指针通过 airkiss_regist_callbacks 传给 SDK。 |
| 参数 | uint32_t funcid | 标识数据是属于微信硬件云平台的哪个业务的。 |
| | const uint8_t* body | 数据内容 |
| | uint32_t bodylen | 数据内容长度 |

| | | |
|-----|---|--|
| 返回值 | 无 | |
|-----|---|--|

SDK 事件消息回调接口实现示例

| | | |
|------|---|--|
| 接口原型 | void ReceiveEventCB(EventValue event_value) | 函数名可以自定，返回值和参数类型固定，实现后把函数指针通过 airkiss_regist_callbacks 传给 SDK。 |
| 参数 | EventValue event_value | 事件类型 |
| 返回值 | 无 | |

发送数据给服务器

| | | |
|------|--|--|
| 接口原型 | uint32_t airkiss_cloud_sendmessage(uint32_t funcid, uint8_t *body, uint32_t bodylen); | |
| 参数 | uint32_t funcid, | 微信硬件云平台业务 ID，能力服务项业务为 1 |
| | uint8_t *body, | 要发送的数据内容，按业务要求填充报文 |
| | uint32_t bodylen | 报文长度 |
| 返回值 | 0/taskid | 返回 0 表示创建上报任务失败，可能是没有调用初始化接口，非 0 的话为 SDK 返回给设备应用层用于标识该任务的 taskid， 注意固件更新检查有专门 API，不通过本 API。 |

| | | |
|------|--|---|
| 接口原型 | uint32_t airkiss_cloud_checkupdate(uint8_t *body, uint32_t bodylen); | 微信硬件平台提供的固件管理业务 API |
| 参数 | uint8_t *body, | 要发送的数据内容，按业务要求填充报文 |
| | uint32_t bodylen | 报文长度 |
| 返回值 | 0/taskid | 返回 0 表示创建上报任务失败，可能是没有调用初始化接口，非 0 的话为 SDK 返回给设备应用层用于标识该任务的 taskid。 |

| | | |
|------|---|--|
| 接口原型 | const uint8_t* airkiss_get_venderid(); | |
| 参数 | 无 | 调用 init 接口后可以再调用本接口获取当前 Licence 对应的 VenderID ,之前命名为 deviceType |
| 返回值 | 字符串，以\0 结尾 | 返回空字符串表示证书无效 |

| | | |
|------|---|--|
| 接口原型 | const uint8_t* airkiss_get_deviceid(); | |
| 参数 | 无 | 调用 init 接口后可以再调用本接口获取当前 Licence 对应的 DeviceID |

| | | |
|-----|------------|--------------|
| 返回值 | 字符串，以\0 结尾 | 返回空字符串表示证书无效 |
|-----|------------|--------------|

2.3.4 SDK 在不同平台的 Porting

由于嵌入式平台种类较多，系统的资源和性能各异，微信硬件平台将核心逻辑代码进行了封装，将需要跨平台实现的代码通过外部函数的方式公开出来，因此在将静态库添加到项目工程以后，开发者还需要根据 `airkiss_porting.h` 中定义的函数原型来实现函数实体，本例以 4004 平台为例，示例各外部接口函数的实现原理，由于对 4004 的平台接触不多，以下代码仅供参考，需要实现的函数列表如下：

| | |
|------|---|
| 函数原型 | <code>int airkiss_printfImp(const char *fmt, ...);</code> |
| 函数说明 | 标准打印函数，用于 debug，关闭 debug 可以不需要，申请静态库需确认是否启用 debug |
| 示例 | <pre>int airkiss_printfImp(const char *fmt, ...){ va_list ap; int ret; va_start(ap, fmt); ret = cmnos_vprintf(cmnos_write_char, fmt, ap); va_end(ap); return (ret); }</pre> |

| | |
|------|---|
| 函数原型 | <code>int airkiss_mutex_create(ak_mutex_t *mutex_ptr);</code> |
| 函数说明 | 多线程模式下提供，创建 mutex，成功返回 0 |
| 示例 | <code>struct ak_mutex_t{</code> |

| | |
|--|--|
| | <pre> TX_MUTEX m_mutex; }; int airkiss_mutex_create(ak_mutex_t *mutex_ptr) { return tx_mutex_create(&(mutex_ptr->m_mutex), "ak_mutex", TX_NO_INHERIT); } </pre> |
|--|--|

| | |
|------|---|
| 函数原型 | int airkiss_mutex_lock(ak_mutex_t *mutex_ptr); |
| 函数说明 | 多线程模式下提供，获取锁，成功返回 0 |
| 示例 | <pre> struct ak_mutex_t{ TX_MUTEX m_mutex; }; int airkiss_mutex_lock(ak_mutex_t *mutex_ptr) { return tx_mutex_get(&(mutex_ptr->m_mutex), TX_WAIT_FOREVER); } </pre> |

| | |
|------|--|
| 函数原型 | int airkiss_mutex_unlock(ak_mutex_t *mutex_ptr); |
| 函数说明 | 多线程模式下提供，释放锁，成功返回 0 |
| 示例 | <pre> struct ak_mutex_t{ TX_MUTEX m_mutex; }; int airkiss_mutex_unlock(ak_mutex_t *mutex_ptr) { return tx_mutex_put(&(mutex_ptr->m_mutex)); } </pre> |

| | |
|------|--|
| 函数原型 | int airkiss_dns_gethost(char* url, uint32_t* ipaddr); |
| 函数说明 | DNS , 根据域名获取 IP , IP 写入 ipaddr , 返回 AK_DNS_WAITING 时 , SDK 会在后续的运行中调用 airkiss_dns_checkstate , IP 地址组成伪代码 :192.168.1.1=>(uint32_t)((192<<24) (168<<16) (1<<8) 1) |
| 返回值 | typedef enum { AK_DNS_SUCCESS = 0, //succeed AK_DNS_FAILED = -1, //failed AK_DNS_WAITING = 2, //waiting } airkiss_dns_state; |
| 示例 | int airkiss_dns_gethost(char* url, uint32_t* ipaddr) { if (0 == qcom_dnsc_get_host_by_name(url, ipaddr)){ return AK_DNS_SUCCESS; } else { return AK_DNS_FAILED; } } |

| | |
|------|---|
| 函数原型 | int airkiss_dns_checkstate(uint32_t* ipaddr); |
| 函数说明 | airkiss_dns_gethost 接口返回 AK_DNS_WAITING 时 , SDK 会在后续的运行中调用本接口 ,本接口用于 airkiss_dns_gethost 是非阻塞方式的平台 , 本接口的返回与 airkiss_dns_gethost 一致 |
| 示例 | 无 |

| | |
|------|---|
| 函数原型 | ak_socket airkiss_tcp_socket_create(); |
| 函数说明 | 创建 TCP 套接字，失败返回-1 |
| 示例 | <pre> ak_socket airkiss_tcp_socket_create() { ak_socket sock; sock = qcom_socket (AF_INET, SOCK_STREAM, 0); if (sock < 0) { return -1; } return sock; } </pre> |

| | |
|------|---|
| 函数原型 | int airkiss_tcp_connect(ak_socket sock, char* ipaddr, uint16_t port); |
| 函数说明 | <p>连接指定的 IP 地址和端口，建议实现非阻塞方式，尝试 300ms 左右不成功的话返回失败，ipaddr 为字符串格式，例如：“192.168.1.100”，需要异步链接的平台，返回 AK_TCP_CONNECT_WAITING，后续 SDK 会调用 airkiss_tcp_checkstate 检查进度，返回值：</p> <pre> typedef enum { AK_TCP_CONNECT_SUCCESS = 0, //succeed AK_TCP_CONNECT_FAILED = -1, //failed AK_TCP_CONNECT_WAITING = 2, //waiting } airkiss_tcp_state; </pre> |
| 示例 | <pre> int airkiss_tcp_connect(ak_socket sock, char* ipaddr, uint16_t port) { </pre> |

| | |
|--|---|
| | <pre> uint32_t addr; int ret; struct sockaddr_in sock_addr; if (0 != parse_ipv4_ad(&addr, ipaddr)) { return -1; } sock_addr.sin_addr.s_addr = addr; sock_addr.sin_port = htons(port); sock_addr.sin_family = AF_INET; ret = qcom_connect(sock, (struct sockaddr *) &sock_addr, sizeof (struct sockaddr_in)); if (ret < 0) { A_PRINTF("Failed to connect socket %d.\n", sock); return -1; } return 0; } </pre> |
|--|---|

| | |
|------|--|
| 函数原型 | int airkiss_tcp_checkstate(ak_socket sock); |
| 函数说明 | <p>检查 TCP 链接的状态，在 airkiss_tcp_connect 接口返回 AK_TCP_CONNECT_WAITING 时会被调用，返回值：</p> <pre> typedef enum { AK_TCP_CONNECT_SUCCESS = 0, //succeed AK_TCP_CONNECT_FAILED = -1, //failed AK_TCP_CONNECT_WAITING = 2, //waiting } airkiss_tcp_state; </pre> |

| | |
|----|---|
| 示例 | 无 |
|----|---|

| | |
|------|--|
| 函数原型 | int airkiss_tcp_send(ak_socket socket, char*buf, uint32_t len); |
| 函数说明 | 通过 TCP 套接字发送数据，返回成功发送的数据大小，-1 为失败，建议非阻塞方式发送，发送时长可定 200ms~300ms |
| 示例 | <pre>int airkiss_tcp_send(ak_socket socket, char*buf, uint32_t len) { return qcom_send(socket, buf, len, 0); }</pre> |

| | |
|------|--|
| 函数原型 | int airkiss_tcp_recv(ak_socket socket, char *buf, uint32_t size, uint32_t timeout); |
| 函数说明 | 通过 TCP 套接字接收数据，返回成功收到的数据大小，-1 为失败，timeout 为建议接收的时长 |
| 示例 | <pre>int airkiss_tcp_recv(ak_socket socket, char *buf, uint32_t size, uint32_t timeout) { int recvBytes; q_fd_set sockSet; A_INT32 fdAct = 0; struct timeval tmo; FD_ZERO(&sockSet); FD_SET(socket, &sockSet); tmo.tv_sec = timeout/1000; timeout = timeout%1000; tmo.tv_usec = timeout*1000; fdAct = qcom_select(socket + 1, &sockSet, NULL, NULL, &tmo);</pre> |

| | |
|--|---|
| | <pre> if (0 != fdAct) { A_PRINTF("fdAct is not 0:%d\r\n", fdAct); if (FD_ISSET(socket, &sockSet)) { recvBytes = qcom_recv(socket, buf, size, 0); if (recvBytes < 0) { return -1; } else { return recvBytes; } } } return 0; } </pre> |
|--|---|

| | |
|------|--|
| 函数原型 | uint32_t airkiss_gettime_ms(); |
| 函数说明 | 返回当前时间或系统已经启动运行了多长时间，单位为 ms |
| 示例 | <pre> uint32_t airkiss_gettime_ms() { return time_ms(); } </pre> |

3 附录

3.1 ERROR_CODE

| 取值 | 描述 |
|-------|------------------------------------|
| 0 | 请求成功 |
| -1 | 系统繁忙，此时请开发者稍候再试 |
| 11000 | 未注册到微信平台，参考接入须知 |
| 11001 | URL 参数不合法 |
| 11002 | POST 数据不合法 |
| 11003 | signature 不合法，请参考接入须知 gnature 生成规则 |
| 11004 | 缺少必选能力项/属性值 |
| 11005 | 异步通知的 device_id 和请求的 device_id 不一致 |
| 11006 | device_id 不合法，请注册 device_id |
| 11007 | 异步通知的 msg_type 和请求的 msg_type 不一致 |
| 11008 | msg_id 的相关的会话已经关闭 |

3.2 ASY_ERROR_CODE

| 取值 | 描述 |
|-------|--------------------|
| 0 | 厂家异步处理成功 |
| 11500 | 系统繁忙 |
| 11501 | 设备没联网 |
| 11502 | 设备已经关机 |
| 11503 | 设备暂时无法操作，请微信平台稍后重试 |

3.3 微信硬件云平台业务 FuncID

| 取值 | 描述 |
|----|----|
|----|----|

| | |
|--------|---------------------|
| 0x0001 | 微信硬件平台能力项业务 FuncID |
| 0x0020 | 微信硬件平台固件管理业务 FuncID |