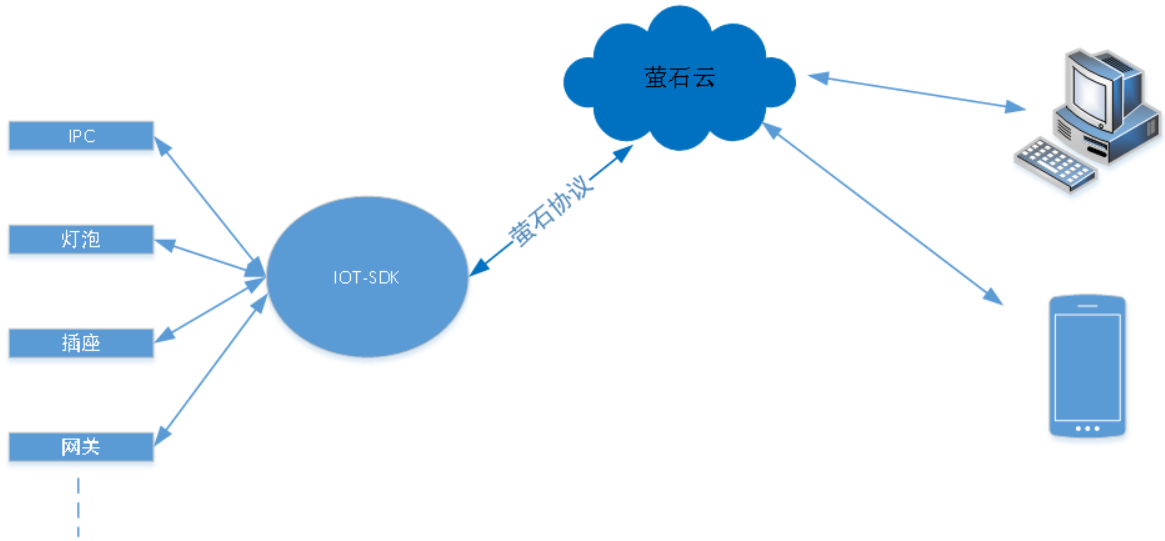
IoT-SDK接入文档

简介

萤石IoT-SDK是萤石IOT平台的一个重要组成部分，用户可以通过接入该产品使用萤石安全可靠的云服务。整个萤石云服务与硬件设备交互的流程如图所示：



IoT-SDK充当着智能设备与萤石云服务交互的桥梁，智能设备接入IoT-SDK后，可以通过萤石互联手机App实现远程控制设备的能力。该产品能够帮助只拥有硬件设备的用户或者企业快速实现自身硬件产品的智能化，提升硬件产品的使用体验，还可以通过强大的萤石云服务实现产品的增值。目前萤石IoT-SDK支持灯泡、IPC、网关、门锁等众多智能硬件产品的接入，用户可以通过访问萤石开放平台官网（open.ys7.com）获取完整的SDK产品及支持。

目录说明

```
+-- doc
    +-- 萤石云Iot_SDK接入文档.pdf   : 快速开始导引
+-- inc
    +-- ez_hal
        +--hal_net_tcp.h          : 网络相关接口
        +--hal_semaphore.h        : 信号量相关接口
        +--hal_thread.h           : 任务相关接口
        +--hal_time.h             : 计时器相关接口
```

```
    +-- ez_iot.h                : 设备上线、绑定等基本功能
    +-- ez_iot_errno.h          : 错误码定义
    +-- ez_iot_ota.h            : 在线升级相关功能
    +-- ez_iot_log.h            : 日志相关接口
    +-- ez_iot_hub.h            : 网关设备功能接口
    +-- ez_iot_tsl.h            : 物模型相关接口
+-- lib
    +-- libez_iot.a             : 萤石云功能业务模块
    +-- libez_linkcore.a        : 萤石云连接
```

快速开始

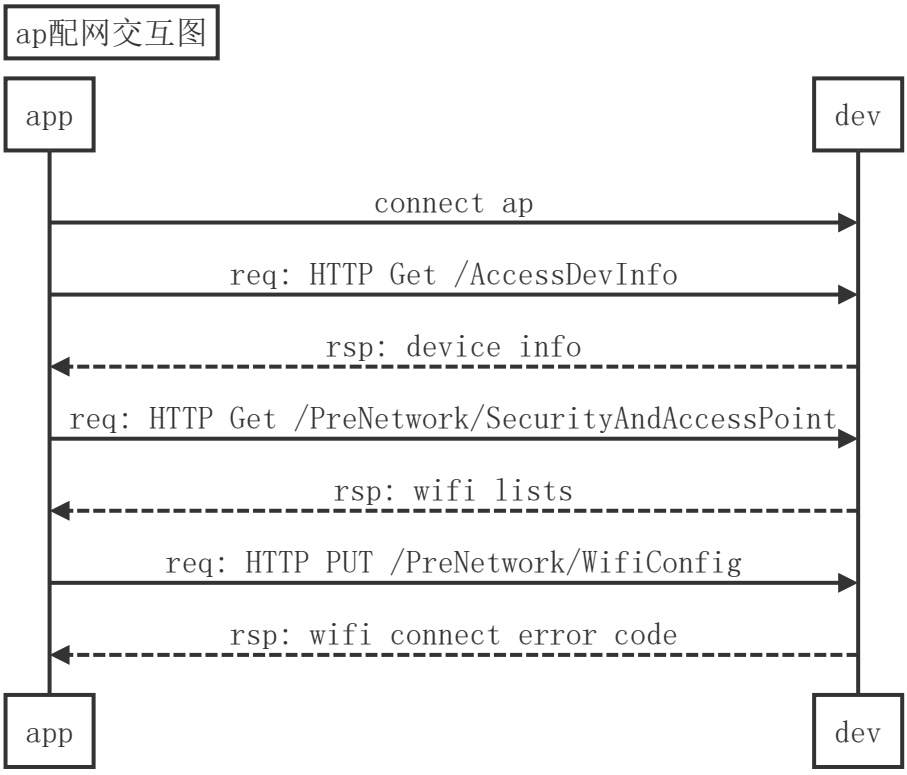
步骤1：获取License

license是设备接入萤石云的凭据，开发者需联系萤石技术支持或从萤石开放平台获取，并自行将其烧录到设备。license格式如下所示：

```
{
    "deviceId": "4LYV8SK7UKLBOUOVS6HXVX:A532NO6K0Q2O",
    "dev_deviceLicense": "J7Fv5gCsTczVe38aNLe4ev",
    "dev_deviceName": "A532NO6K0Q2O",
    "dev_productKey": "4LYV8SK7UKLBOUOVS6HXVX"
}
```

步骤2：配网

基于萤石互联APP完成配网，设备可获得**注册地址、设备绑定Token、WiFi SSID、WiFi 密码**四元组信息。配网期间设备需将WiFi模块切换成AP模式，作为Server响应以下HTTP协议。Server默认IP为192.168.4.1，端口80



- 获取设备相关信息

```
Get /AccessDevInfo
描述：APP端连接之后获取设备相关信息
入口数据：无
返回：
    {
         "ap_version":"1.0",                                          :必填-协议版本
         "dev_subserial":"4LYV8SK7UKLBOUOVS6HXVX:A532NO6K0Q2O", :必填-设备序列号，
最大64字节
         "dev_type":"4LYV8SK7UKLBOUOVS6HXVX",                    :必填-设备型号，最
大64字节
         "dev_firmwareversion":"V1.0.0 build 190823"             :必填-设备固件版本
号，最大64字节
    }
```

- 获取WiFi列表

```
Get /PreNetwork/SecurityAndAccessPoint
描述：AP接入wifi配置操作时，获取相关信息 （最大设置为20个）
入口数据：无
返回：
    {
      "access_point_list":[{
       "ssid":"test",            :必填-SSID，最大32字节
       "signal_strength":-20,    :必填-信号强度，取值范围-100至0
       "security_mode":"",       :可选-安全模式，取值范围"open,WEP,WPA-personal,WPA2-
personal,WPA-WPA2-personal,WPA2-enterprise"
      }]
    }
```

- 设置四元组

```
PUT /PreNetwork/WifiConfig
描述： AP接入Wifi配置操作（wifi连接可能会耗一定的时间，这里设置wifi连接超时时间为10s）
入口数据：
    {
      "token":"AABBCCDDEEFFGGHH",        :必填-用于认证，最大32字节
      "lbs_domain":"devcl.ys7.com",      :必填-设备注册平台地址
      "device_id":"",                    :选填-设备uuid
      "wifi_info":
      {
       "ssid":"",                        :必填-SSID，最大32字节
       "password":"",                    :必填-wifi的密码信息
      },
    }
返回：
    {
         "status_code": 104,              :必填-状态码，取值范围"104(成功),105(未知错
误),106(密码错误),201(未找到wifi)"
         "status_string": ""              :选填-状态描述
    }
```

步骤3：设备连接云平台

启动SDK，收到ez_iot_event_online事件回调则表示设备已连上云平台。

```
static int32_t ez_recv_event_cb(ez_iot_event_t event_type, void *data, int len)
{
    char file_path[128] = {0};
    switch (event_type)
    {
        case ez_iot_event_online:
        /* 设备已连接云平台 */
        break;
        case ez_iot_event_devid_update:
        /* 服务端颁发给设备uuid，保存至永久存储区域，后续SDK初始化时使用 */
        break;
        default:
        break;
    }
}

/* 初始化SDK参数 */
ez_iot_srv_info_t srv_info = {lbs_domain,xxx};
ez_iot_dev_info_t dev_info = {xxx};
ez_iot_callbacks_t cbs = {ez_recv_msg_cb, ez_recv_event_cb};
...

/* 初始化SDK */
ez_iot_init(&srv_info, &dev_info, &cbs);

/* 运行SDK */
ez_iot_start();
```

步骤4：绑定用户

如设备需要绑定到萤石APP账号下，有两种方式：

1. 通过AP配网进行绑定，设备端将从APP获取到的token发往平台完成绑定。
2. APP扫描设备二维码发起绑定，服务下发挑战码至设备，用户物理参与确认后调用相应接口完成挑战响应。

```
/**
  * @brief 设备绑定至用户账号
  *
  * @param dev_token: 最大长度64字节
  * @return ez_err_e
  */
ez_err_e ez_iot_binding(int8_t *dev_token);

/**
  * @brief 接触式绑定响应，需用户参与物理确认后调用，如按键等
  *
  * @param challenge 服务下发的挑战信息
  * @return ez_err_e
  */
ez_err_e ez_iot_contact_binding(ez_iot_challenge_t *challenge);
```

步骤5：数据传输

萤石IoT平台将设备的能力抽象为属性、操作、事件三个维度，开发者需在萤石开放平台参考产品功能的规范与标准，定义硬件产品的功能。连云成功后，用户可以根据业务需求向云端上报数据和接收云端的数据。

功能类型	说明
属性（Property）	描述设备运行时的状态，例如灯具开关状态、当前亮度等。
操作（action）	设备提供给外部调用的方法。可携带入参和出参。相比于属性，操作更侧重于设备响应的实时性。例如添加打开门窗，操作能一条命令获得执行结果，若定义成属性，则需要监控门窗的状态来获得执行结果。
事件（Event）	设备运行时的事件/告警，例如温度、湿度超标。

- 上报属性

```
tsl_things_callbacks_t tsl_things_cbs = {tsl_things_action2dev,
tsl_things_property2cloud, tsl_things_property2dev};
tsl_devinfo_t tsl_devinfo = {(int8_t *)EZ_IOT_DEV_UUID,
(int8_t*)EZ_IOT_DEV_PRODUCT_KEY, (int8_t *)EZ_IOT_DEV_FWVER};
tsl_key_info_t tsl_keyinfo = {.domain = (int8_t *)"attr_test", .key =
(int8_t*)"attr_r_bool"};
tsl_value_t tsl_value = {.size = 1, .type = tsl_data_type_bool, .value_bool =
false};
tsl_rsc_info_t rsc_info = {.res_type = (int8_t *)"PetDryerRes", .local_index =
(int8_t *)"0"};

ez_iot_init(&m_lbs_addr, &m_dev_info, &m_cbs, &m_kv_cbs);
ez_iot_tsl_init(&tsl_things_cbs);
ez_iot_start();
dev_event_waitfor(ez_iot_event_online, DEFAULT_TIMEOUT_S * 1000);
ez_iot_tsl_property_report(tsl_devinfo.dev_subserial, &rsc_info,
&tsl_keyinfo,&tsl_value);
```

- 上报事件

```
tsl_things_callbacks_t tsl_things_cbs = {tsl_things_action2dev,
tsl_things_property2cloud, tsl_things_property2dev};
tsl_devinfo_t tsl_devinfo = {(int8_t *)EZ_IOT_DEV_UUID,
(int8_t*)EZ_IOT_DEV_PRODUCT_KEY, (int8_t *)EZ_IOT_DEV_FWVER};
tsl_key_info_t tsl_keyinfo = {.domain = (int8_t *)"event_test", .key =
(int8_t*)"event_null"};
tsl_rsc_info_t rsc_info = {.res_type = (int8_t *)"PetDryerRes", .local_index =
(int8_t *)"0"};

ez_iot_init(&m_lbs_addr, &m_dev_info, &m_cbs, &m_kv_cbs);
ez_iot_tsl_init(&tsl_things_cbs);
ez_iot_start();
dev_event_waitfor(ez_iot_event_online, DEFAULT_TIMEOUT_S * 1000);
ez_iot_tsl_event_report(tsl_devinfo.dev_subserial, &rsc_info,
&tsl_keyinfo,NULL);
```

- 下发属性/操作

```
static int32_t tsl_things_action2dev(const int8_t *sn, const tsl_rsc_info_t*
rsc_info, const tsl_key_info_t *key_info, const tsl_value_t
*value_in,tsl_value_t *value_out)
```

```
{
    /* 处理下发操作 */
    return 0;
}
int32_t tsl_things_property2dev(const int8_t *sn, const tsl_rsc_info_t
*rsc_info, const tsl_key_info_t *key_info, const tsl_value_t *value)
{
    /* 处理下发属性 */
    return 0;
}

tsl_things_callbacks_t tsl_things_cbs = {tsl_things_action2dev,
tsl_things_property2cloud, tsl_things_property2dev};
ez_iot_init(&m_lbs_addr, &m_dev_info, &m_cbs, &m_kv_cbs);
ez_iot_tsl_init(&tsl_things_cbs);
ez_iot_start();
```

设备升级

升级流程：

1. 平台端部署升级包。
2. 设备启动后，上报设备当前的模块信息（型号、版本号，多模块信息）。
3. APP检测到有可用升级包，下发升级请求。
4. 设备响应升级请求。
5. 下载升级包、校验、烧录，上报升级进度。
6. 上报升级结果（成功/失败）

```
/**
* @brief 升级包下载数据回调
* @param total_len :package total size
* @param offset :offset of current download package
* @param data &len :current download package data & len
* @param remain_len:the size left to process in next cb.
* @param user_data :user input data
* @return int
*/
typedef int (*get_file_cb)(uint32_t total_len, uint32_t offset, void *data,
uint32_t len, void* user_data);

/**
* @brief 升级包下载过程通知回调
* @param result 下载结果,0表示成功 非0表示失败
* @param user_data: 用户自定义协议
* @return int
*/
typedef void (*notify_cb)(ota_cb_result_e result, void* user_data);

/**
* @brief 上报升级模块信息,同步接口，请勿在SDK的任何消息回调里调用该接口
* @param pres 设备信息，NULL标识当前设备，!NULL标识子设备
* @param pmodules 设备模块信息列表,可以有多个模块，
* @param timeout_ms 指定超时时间
* @return ez_err_e
*/
```

```
EZIOT_API ez_err_e ez_iot_ota_modules_report(const ota_res_t *pres, const
ota_modules_t* pmodules, uint32_t timeout_ms);
/**
* @brief 通知升级服务，设备待升级，设备开机上线上报一次即可。
*
* @param pres 设备信息，NULL标识当前设备，!NULL标识子设备
* @return ez_err_e
*/
EZIOT_API ez_err_e ez_iot_ota_status_ready(const ota_res_t *pres, int8_t*
pmodule);

/**
* @brief 升级成功信息上报
*
* @param pres 设备信息，NULL标识当前设备，!NULL标识子设备
* @return ez_err_e
*/
EZIOT_API ez_err_e ez_iot_ota_status_succ(const ota_res_t *pres, int8_t*
pmodule);

/**
* @brief 升级失败信息上报
*
* @param pres 设备信息，NULL标识当前设备，!NULL标识子设备
* @param pmodule 固件模块名称
* @param perr_msg 升级失败错误码
* @param code 升级失败错误码
* @return ez_err_e
*/
EZIOT_API ez_err_e ez_iot_ota_status_fail(const ota_res_t *pres, int8_t*
pmodule, int8_t* perr_msg, ota_errcode_e code);

/**
* @brief 上报设备升级进度,请按照服务下发间隔时间上报,app点击升级后,需要在10s时间内上报一次进
度信息,否则app端可能会直接提示升级成功
*
* @param pres 设备信息，NULL标识当前设备，!NULL标识子设备
* @param progress 升级进度 1-100
* @return ez_err_e
*/
EZIOT_API ez_err_e ez_iot_ota_progress_report(const ota_res_t *pres, int8_t*
pmodule, ota_status_e status, int16_t progress);

/**
* @brief 下载升级包
* @param input_info 升级信息
* @param get_file 下载数据回调
* @param notify 下载过程信息通知，包括超时,c
* @return ez_err_e
*/
EZIOT_API ez_err_e ez_iot_ota_download(ota_download_info_t *input_info,
get_file_cb file_cb, notify_cb notify, void *user_data);
```

网关管理子设备

```
typedef struct
{
```

```
    bool online;            ///< 设备是否在线
    int8_t subdev_sn[64];   ///< 子设备序列号
    int8_t subdev_ver[64];  ///< 子设备版本号
    int8_t subdev_type[64]; ///< 子设备型号
} hub_subdev_info_t;

/**
* @brief 添加子设备
*
* @param info 子设备信息
* @return ez_err_e
*/
ez_err_e ez_iot_hub_add(const hub_subdev_info_t *subdev_info);
/**
* @brief 删除子设备
*
* @param subdev_sn 子设备序列号
* @return ez_err_e
*/
ez_err_e ez_iot_hub_del(const int8_t *subdev_sn);
/**
* @brief 更新子设备版本号，常见于升级完成后
*
* @param info 子设备信息
* @return ez_err_e
*/
ez_err_e ez_iot_hub_ver_update(const int8_t *subdev_sn, const int8_t
*subdev_ver);

/**
* @brief 更新子设备联网状态
*
* @param online false不在线，true在线
* @return ez_err_e
*/
ez_err_e ez_iot_hub_status_update(const int8_t *subdev_sn, bool online);

/**
* @brief 根据序列号查询子设备信息
*
* @param subdev_sn 子设备序列号
* @param subdev_info 子设备信息，不能为空
* @return ez_err_e
*/
ez_err_e ez_iot_hub_subdev_query(const int8_t *subdev_sn, hub_subdev_info_t
*subdev_info);

/**
* @brief 枚举所有子设备信息
*
* @param subdev_info 子设备信息，不能为空;subdev_sn未空串标识获取首个子设备
* @return ez_err_e
*/
ez_err_e ez_iot_hub_subdev_next(hub_subdev_info_t *subdev_info);

/**
* @brief 清空所有子设备，常见于网关重置
*
```

```
 * @return ez_err_e
 */
ez_err_e ez_iot_hub_clean(void);
```