WELCOME TO USE WHATIESDK-ANDROID

Shared device turn off event

Shared device offline event

Set countdown success event

Cancel countdown success event

Add/open timer success event

Delete/close timer success event

3.5 SmartConfig

(1)Define inner class “WhatieAsyncTask” in activity that config network：

(2)After get smartconfig token：

(3)Result of config device:

Attention

Welcome to use WhatieSDK-Android

1、Introduction

WhatieSDK is an SDK provided by ATI TECHNOLOGY (WUHAN) CO.,LTD for third party access to IOT platform quickly. Through this SDK, users can register and log in, find and add devices such as nearby smart sockets, and control device switches.

2、Import SDK

IDE：Android Studio

(1)Import WhatieSDK-x.x.x.aar:

```
Import “.aar” file into “lib” of your project.
```

(2)Add following code to “build.gralde” file in your project:

```
compile(name:'WhatieSDK-x.x.x', ext:'aar')

repositories {
        flatDir {
           dirs 'libs'
        }
    }
```

(3)Import the required third-party plug-in:

```
compile 'com.mylhyl:zxingscanner:2.0.0'    //encode and decode QRcode
compile 'com.lzy.net:okgo:3.0.4'           // HTTP connection
compile 'com.alibaba:fastjson:1.2.20'       //json2object
compile 'org.eclipse.paho:org.eclipse.paho.client.mqttv3:1.1.0'     //about mqtt
compile 'org.eclipse.paho:org.eclipse.paho.android.service:1.1.1'   //about mqtt
compile 'org.greenrobot:eventbus:3.0.0'        // communations among threads
```

3、Usage

3.1 Initialise

(1)Add the following code to the “onCreate()” method of your Application.class :

```
EHomeInterface.getINSTANCE().init(this)。
```

(2)Add the following code to AndroidManifest.xml:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

```
<uses-permission android:name="android.permission.CHANGE_WIFI_MULTICAST_STATE" />
<uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
<uses-permission android:name="android.permission.WAKE_LOCK"/>
<uses-permission android:name="android.permission.CAMERA"/>
```

(3) Declare the following service of AndroidManifest.xml:

```
<service android:name="org.eclipse.paho.android.service.MqttService"/>
<service android:name="com.d9lab.ati.whatiesdk.mqtt.MyMqttService"/>
<service android:name="com.d9lab.ati.whatiesdk.tcp.TcpService"/>
<service android:name="com.d9lab.ati.whatiesdk.udp.UdpService"/>
```

3.2 HTTP callback

Registration / login / auto login of user

```
/**
     * ensure single sign-on, or the background service will report a mistake
     * @param mContext  The activity that uses this method
     * @param email      user' s email
     * @param password  user' s password
     * @param accessId   company' s accessId
     * @param accessKey  company' s accessKey
     * @param callback    callback of network cummunicaiton
     */
EHomeInterface.getINSTANCE().login(mContext, email, password, accessId, accessKey,
        new UserCallback() {
            @Override
            public void onSuccess(Response<BaseModelResponse<User>> response) {
                if (response.body().isSuccess()) {
                  /**
                   * The following methods must be called when login success.
                   */
                    EHome.getInstance().setLogin(true);
                    EHome.getInstance().setmUser(response.body().getValue());
                    EHome.getInstance().setToken(response.body().getToken());
                } else {
                    Toast.makeText(mContext, "Login fail.", Toast.LENGTH_SHORT).show();
                }
            }
            @Override
            public void onError(Response<BaseModelResponse<User>> response) {
                super.onError(response);
                Toast.makeText(mContext, "Login fail.", Toast.LENGTH_SHORT).show();
            }
        });
```

Log off

```
/**
```

```
 *
 * @param mContext
 * @param accessId
 * @param accessKey
 * @param baseCallback
 */
EHomeInterface.getINSTANCE().logOut(mContext, Constant.ACCESS_ID, Constant.ACCESS_KEY,
   new BaseCallback() {
    @Override
    public void onSuccess(Response<BaseResponse> response) {
         EHome.getInstance().logOut(); // This method must be called when logout success.
    }
   @Override
   public void onError(Response<BaseResponse> response) {
         super.onError(response);
    }
});
```

Change user's password

```
/**
 *
 * @param mContext
 * @param email
 * @param oldPwd
 * @param newPwd
 * @param accessId
 * @param accessKey
 * @param baseCallback
 */
EHomeInterface.getINSTANCE().changePwd(mContext, email, oldPwd, newPwd, accessId, accessKey,
        new BaseCallback() {
            @Override
            public void onSuccess(Response<BaseResponse> response) {
            }
            @Override
            public void onError(Response<BaseResponse> response) {
                super.onError(response);
            }
        });
```

Get my device list

```
/**
 * After get device list, “saveDevices” method must be called.
 * @param mContext
 * @param accessId
 * @param accessKey
 * @param callback
 */
```

```
EHomeInterface.getINSTANCE().getMyDevices(mContext, accessId, accessKey, new DevicesCallback() {
    @Override
    public void onSuccess(Response<BaseListResponse<DeviceVo>> response) {
        if (response.body().isSuccess()){
            EHome.getInstance().saveDevices(response.body().getList());
        }
    }

    @Override
    public void onError(Response<BaseListResponse<DeviceVo>> response) {
        super.onError(response);
    }
});
```

Edit device's name

```
/**
 *
 * @param mContext
 * @param deviceId            device' s devId
 * @param newName           device' s new name
 * @param accessId
 * @param accessKey
 * @param devicesCallback
 */
EHomeInterface.getINSTANCE().changeDeviceName(mContext, deviceId, newName, accessId, accessKey,
        new BaseCallback() {
            @Override
            public void onSuccess(Response< BaseResponse > response) {
                if (response.body().isSuccess()) {
                }
            }
            @Override
            public void onError(Response<BaseResponse> response) {
                super.onError(response);
            }
        });
```

Add sharing device

```
/**
 *
 * @param mContext
 * @param adminId                    device owner' s id
 * @param deviceId                    device' s id (not devId)
 * @param timestamp                  time when generate the QRcode
 * @param accessId
 * @param accessKey
 * @param baseCallback
 */
```

```
EHomeInterface.getINSTANCE().addSharedDevice(mContext,adminId,deviceId,timestamp,accessId,
accessKey
    new BaseCallback() {
    @Override
    public void onSuccess(Response<BaseResponse> response) {
        if(response.body().isSuccess()){
        }
    }

    @Override
    public void onError(Response<BaseResponse> response) {
        super.onError(response);
    }
});
```

Invite new home member

```
/**
 *
 * @param tag
 * @param homeId
 * @param userEmail                  new home member' s Email
 * @param timeStamp                  time when generate the QRcode
 * @param baseCallback
 */
EHomeInterface.getINSTANCE().inviteHomeMember(mContext, homeId, userEmail, timeStamp,timeStamp,
 new BaseCallback() {
            @Override
            public void onSuccess(Response<BaseResponse> response) {
            }

            @Override
            public void onError(Response<BaseResponse> response) {
                super.onError(response);
            }
        });
```

Transfer home to someone else

```
/**
 *
 * @param tag
 * @param homeId                     id of the home to be transferred
 * @param hostId                     id of the former owner
 * @param userEmail                  new owner' s Email
 * @param timeStamp
 * @param baseCallback
 */
EHomeInterface.getINSTANCE().transferHome(mContext, homeId, hostId, userEmail, timestamp,
```

```
new BaseCallback(){
                    @Override
                    public void onSuccess(Response<BaseResponse> response) {
                    }

                    @Override
                    public void onError(Response<BaseResponse> response) {
                        super.onError(response);
                    }
                });
```

Get production type

```
/**
 *
 * @param mContext
 * @param current                    current page
 * @param size                       item size per page
 * @param accessId
 * @param accessKey
 * @param baseCallback
 */
EHomeInterface.getINSTANCE().getProductTypePage(mContext,current,size, accessId, accessKey,
    new ProductionCallback() {
    @Override
    public void onSuccess(Response<BaseListResponse<Product>> response) {
        if(response.body().isSuccess()){
        }
    }
    @Override
    public void onError(Response<BaseListResponse<Product>> response) {
        super.onError(response);
    }
});
```

Get smartconfig token

```
/**
 *
 * @param mContext
 * @param accessId
 * @param accessKey
 * @param baseStringCallback
 */
EHomeInterface.getINSTANCE().getNetToken(mContext, accessId, accessKey, new BaseStringCallback() {
    @Override
    public void onSuccess(Response<BaseModelResponse<String>> response) {
        if (response.body().isSuccess()){
```

```
        }
    }
    @Override
    public void onError(Response<BaseModelResponse<String>> response) {
        super.onError(response);
    }
});
```

Unbind offline device

```
/**
 *
 * @param mContext
 * @param id                device's id
 * @param accessId
 * @param accessKey
 * @param baseCallback
 */
EHomeInterface.getINSTANCE().deleteDevice(mContext, id, accessId, accessKey,
    new BaseCallback() {
    @Override
    public void onSuccess(Response<BaseResponse> response) {
        if (response.body().isSuccess()) {
            EHome.getInstance().removeDevice(devId);   //this method must be called after delete
success.
        }
    }
    @Override
    public void onError(Response<BaseResponse> response) {
        super.onError(response);
    }
});
```

Delete sharing device

```
/**
 *
 * @param mContext
 * @param devId                  device's devId
 * @param accessId
 * @param accessKey
 * @param baseCallback
 */
EHomeInterface.getINSTANCE().deleteSharedDevice(mContext, devId, accessId, accessKey,
    new BaseCallback() {
    @Override
    public void onSuccess(Response<BaseResponse> response) {
        if (response.body().isSuccess()) {
            EHome.getInstance().removeDevice(devId);   //this method must be called after delete
```

```
success.
        }
    }
    @Override
    public void onError(Response<BaseResponse> response) {
        super.onError(response);
    }
});
```

Initialize device

```
/**
 * After receiving the MqttBindSuccessEvent, this method must be called.
 * @param mContext
 * @param devId             device' s devId
 * @param name             device' s name
 * @param accessId
 * @param accessKey
 * @param baseCallback
 */
EHomeInterface.getINSTANCE().getStarted(mContext,devId,name, accessId, accessKey,,
        new BaseCallback() {
            @Override
            public void onSuccess(Response<BaseResponse> response) {
                if(response.body().isSuccess()){
                }
            }

            @Override
            public void onError(Response<BaseResponse> response) {
                super.onError(response);
            }
        });
```

Get shared device list

```
/**
 * devices sharing from others
 * @param mContext
 * @param accessId
 * @param accessKey
 * @param devicesCallback
 */
EHomeInterface.getINSTANCE().getSharedDevices(mContext, accessId, accessKey, new DevicesCallback() {
    @Override
    public void onSuccess(Response<BaseListResponse<DeviceVo>> response) {
        if (response.body().isSuccess()) {
        }
    }
    @Override
```

```
    public void onError(Response<BaseListResponse<DeviceVo>> response) {
        super.onError(response);
    }
});
```

Get device's countdown

```
/**
 *
 * @param mContext
 * @param deviceId  //device' s id
 * @param accessId
 * @param accessKey
 * @param clockCallback
 */
EHomeInterface.getINSTANCE().getCountdown(mContext,deviceId, accessId, accessKey,
        new ClockCallback() {
            @Override
            public void onSuccess(Response<BaseListResponse<ClockVo>> response) {
            }
            @Override
            public void onError(Response<BaseListResponse<ClockVo>> response) {
                super.onError(response);
            }
        });
```

Get device's timer list

```
/**
 *
  * @param mContext
 * @param deviceId  //device' s id
 * @param accessId
 * @param accessKey
 * @param callback
 */
EHomeInterface.getINSTANCE().getTimerList(mContext, deviceId, accessId, accessKey,
        new ClockCallback() {
            @Override
            public void onSuccess(Response<BaseListResponse<ClockVo>> response) {
            }

            @Override
            public void onError(Response<BaseListResponse<ClockVo>> response) {
                super.onError(response);
            }
        });
```

Add timer

```
/**
 *
 * @param mContext
 * @param deviceId
 * @param timerType //A sevent-bit binary string. From the lowest bit to highest bit, indicating Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday respectively. “1” means this timer will execute, and “0” means not. For example, “1100000” means timer will execute on Sunday and Saturday, “0000000” means timer will execute only once without repeat.
  * @param hour  //from “00” to “23”
 * @param min   //from “00” to “59”
 * @param dps   //execution state of device
 * @param accessId
 * @param accessKey
 * @param baseCallback
 */
EHomeInterface.getINSTANCE().addTimer(mContext,    deviceId,    timetype,    finishHour,    finishMin, deviceState, accessId, accessKey,
        new BaseCallback() {
            @Override
            public void onSuccess(Response<BaseResponse> response) {
            }

            @Override
            public void onError(Response<BaseResponse> response) {
                super.onError(response);
            }
        });
```

Delete timer

```
/**
 *
 * @param mContext
 * @param clockId
 * @param accessId
 * @param accessKey
 * @param baseCallback
 */
EHomeInterface.getINSTANCE().deleteTimer(mContext, clockId, accessId, accessKey,
        new BaseCallback() {
            @Override
            public void onSuccess(Response<BaseResponse> response) {
            }
            @Override
            public void onError(Response<BaseResponse> response) {
                super.onError(response);
            }
```

```
        });
```

Change timer's information

```
/**
 *
 * @param mContext
 * @param clockId  //timer' s id
 * @param timerType
 * @param hour
 * @param min
 * @param dps
 * @param accessId
 * @param accessKey
 * @param baseCallback
 */
EHomeInterface.getINSTANCE().editTimer(mContext,    clockId,    timetype,    finishHour,    finishMin,
deviceState, accessId, accessKey,
        new BaseCallback() {
            @Override
            public void onSuccess(Response<BaseResponse> response) {
            }
            @Override
            public void onError(Response<BaseResponse> response) {
                super.onError(response);
            }
        });
```

Open/close timer

```
/**
 *
 * @param mContext
 * @param clockId
 * @param state  //state of timer
 * @param accessId
 * @param accessKey
 * @param baseCallback
 */
EHomeInterface.getINSTANCE().editTimerStatus(mContext, clockId, state, accessId, accessKey,
        new BaseCallback() {
            @Override
            public void onSuccess(Response<BaseResponse> response) {
            }
            @Override
            public void onError(Response<BaseResponse> response) {
                super.onError(response);
            }
        });
```

3.3 Device Control

The device can only be controlled in the case of a MQTT or TCP connection.

(1) Determine whether the MQTT is connected or not:

```
EHome.getInstance().isMqttOn()
```

(2) Determine whether there is an TCP connection with the device:

```
EHome.getLinkedTcp().containsKey(devId);
```

Turn on the device

```
/**
 *
 * @param devId  device' s devId
 */
EHomeInterface.getINSTANCE().turnOn(devId);
```

Turn off the device

```
/**
 *
 * @param devId  device' s devId
 */
EHomeInterface.getINSTANCE().turnOff(devId);
```

Unbind the device

```
/**
 *
 * @param devId  device' s devId
 */
EHomeInterface.getINSTANCE().unbind(devId);
```

Set countdown

```
/**
 *
 * @param devId    device' s devId
 * @param state     execution state of device
 * @param duration   countdown time = hour*60*60+min*60
 */
 EHomeInterface.getINSTANCE().setCountdown(devId, state, duration);
```

Cancel countdown

```
/**
 *
 * @param devId   device' s devId
 */
EHomeInterface.getINSTANCE().cancelCountdown(devId);
```

3.4 Subscription event

All event means can be found in the corresponding class file. Please check the use of Eventbus at first.

Turn on event

```
@Subscribe(threadMode = ThreadMode.MAIN, priority = 1, sticky = true)
public void onEventMainThread(MqttReceiveOnEvent event) {}
```

Turn off event

```
@Subscribe(threadMode = ThreadMode.MAIN, priority = 1, sticky = true)
public void onEventMainThread(MqttReceiveOffEvent event) {}
```

Unbind event

```
@Subscribe(threadMode = ThreadMode.MAIN, priority = 1, sticky = true)
public void onEventMainThread(MqttReceiveUnbindEvent event) {}
```

Offline event

```
@Subscribe(threadMode = ThreadMode.MAIN, priority = 1, sticky = true)
public void onEventMainThread(MqttReceiveStatusEvent event) {}
```

Bind success event

```
@Subscribe(threadMode = ThreadMode.MAIN, priority = 1, sticky = true)
public void onEventMainThread(MqttBindSuccessEvent event) {}
```

Already bind event

```
@Subscribe(threadMode = ThreadMode.MAIN, priority = 1, sticky = true)
public void onEventMainThread(MqttAlreadyBindEvent event) {}
```

Shared device turn on event

```
@Subscribe(threadMode = ThreadMode.MAIN, priority = 1, sticky = true)
public void onEventMainThread(MqttReceiveSharedOnEvent event) {}
```

Shared device turn off event

```
@Subscribe(threadMode = ThreadMode.MAIN, priority = 1, sticky = true)
public void onEventMainThread(MqttReceiveSharedOffEvent event) {}
```

Shared device offline event

```
@Subscribe(threadMode = ThreadMode.MAIN, priority = 1, sticky = true)
public void onEventMainThread(MqttReceiveSharedStatusEvent event) {}
```

Set countdown success event

```
@Subscribe(threadMode = ThreadMode.MAIN, priority = 1, sticky = true)
public void onEventMainThread(MqttSetCdSuccessEvent event) {}
```

Cancel countdown success event

```
@Subscribe(threadMode = ThreadMode.MAIN, priority = 1, sticky = true)
```

```
public void onEventMainThread(MqttCancelCdSuccessEvent event) {}
```

Add/open timer success event

```
@Subscribe(threadMode = ThreadMode.MAIN, priority = 1, sticky = true)
public void onEventMainThread(MqttSetTimerSuccessEvent event) {}
```

Delete/close timer success event

```
@Subscribe(threadMode = ThreadMode.MAIN, priority = 1, sticky = true)
public void onEventMainThread(MqttCancelTimerSuccessEvent event) {}
```

3.5 SmartConfig

(1)Define inner class "WhatieAsyncTask" in activity that config network：

```
private class WhatieAsyncTask extends AsyncTask<String, Void, List<IEsptouchResult>> {

    @Override
    protected void onPreExecute() {
    }

    @Override
    protected List<IEsptouchResult> doInBackground(String... params) {
        int taskResultCount = -1;
        synchronized (mLock) {
            String apSsid = mWifiAdmin.getWifiConnectedSsidAscii(params[0]);
            String apBssid = params[1];
            String apPassword = params[2];
            String taskResultCountStr = params[3];
            taskResultCount = Integer.parseInt(taskResultCountStr);
            mEsptouchTask = new EsptouchTask(apSsid, apBssid, apPassword, mContext);
        }
        List<IEsptouchResult> resultList = mEsptouchTask.executeForResults(taskResultCount);
        return resultList;
    }

    @Override
    protected void onPostExecute(List<IEsptouchResult> result) {
        IEsptouchResult firstResult = result.get(0);
        if (!firstResult.isCancelled()) {
            if (firstResult.isSuc()) {
                //TODO
            } else {
                //TODO
            }
        }
    }
}
```

(2)After get smartconfig token：

```
private EspWifiAdminSimple mWifiAdmin = new EspWifiAdminSimple(this);
String apSsid = mWifiAdmin.getWifiConnectedSsid();
String apBssid = mWifiAdmin.getWifiConnectedBssid();

/**
 * Must be connected to 2.4 G Wi-Fi,
 * and router, mobile phone and device are close enough
 * @param apSsid
 * @param apBssid
 * @param tokenAndPwd   token + router’s password
 */

new WhatieAsyncTask().execute(apSsid, apBssid, tokenAndPwd, "1");
```

(3)Result of config device:

Bind Success:
Receiving the MqttBindSuccessEvent event.
Bind failed:
Receiving the MqttAlreadyBindEvent event means the device has been bound by others.

Attention

1、When activity enters onPause state，please call the following method:

```
EHome.getInstance().cancelTag(this);
```

2、When turn on/ off a device, please determine whether this device has countdown. If so, cancel countdown at same time.