

# 移动智能终端补充设备标识体 系统一调用 SDK

## 开发者说明文档

|        |                  |
|--------|------------------|
| 编写人    | 移动安全联盟           |
| 文档版本   | v1.13            |
| SDK 版本 | v1.0.13          |
| 最新修订日期 | 2020 年 02 月 06 日 |

## 权利声明

任何单位或个人使用移动智能终端补充设备标识体系统一调用 SDK 前，应当仔细阅读本权利声明并确认同意本权利声明所述内容，否则不得使用本 SDK。您理解并承诺，您使用 SDK 的行为，即视为您已了解并完全同意本权利声明的各项内容，且您将基于这些声明承担相应的法律责任与义务。

1、移动智能终端补充设备标识体系统一调用 SDK 由中国信息通信研究院泰尔终端实验室、移动安全联盟整合提供，知识产权归中国信息通信研究院所有。任何单位或个人未经授权不得修改、复制、发行、出租、传播或翻译 SDK，不得逆向、破解、篡改、二次打包、公开、贩卖 SDK，不得向任何第三方披露 SDK，不得进行二次分发，不得利用 SDK 直接获利或用于其他商业用途，如基于本 SDK 向第三方提供与本 SDK 相似功能。如有违反中国信息通信研究院保留追究其法律责任的权利。

2、移动智能终端补充设备标识体系依据电信终端产业协会（TAF）、移动安全联盟（MSA）联合推出的团体标准《移动智能终端补充设备标识规范》开发，移动智能终端补充设备标识体系统一调用 SDK 集成设备厂商提供的接口，并已获得相应设备厂商的授权。

3、任何单位或个人使用本 SDK 获取相关标识数据，并欲据此标识数据进行使用、分析、交易或其他行为前，应慎重考虑这些行为的合法性、合规性、真实性和安全性等，并采取相应措施。如因未采取措施所导致的信息泄露、权利侵犯、财产毁损、人身伤亡等违法违规行为及因该等行为所造成的损害后果，与中国信息通信研究无关，中国信息通信研究亦不承担任何直接、间接、附带或衍生的损失和责任。

4、移动智能终端补充设备标识体系统一调用 SDK 由中国信息通信研究院泰尔终端实验室、移动安全联盟共同负责 SDK 的合规管理和后期维护，移动安全联盟官方网站是目前唯一合法发布本 SDK 版本、代码、文档的组织（目前发布地址 <http://www.msa-alliance.cn/col.jsp?id=120>），任何其他企业或组织发布的均为非法行为，请仔细甄别。中国信息通信研究院保留追究其法律责任的权利。

5、本次版本为试用版。

中国信息通信研究院

2020 年 3 月 17 日

## 一. 覆盖范围

| 厂商名称      | 支持版本               |
|-----------|--------------------|
| 华为        | 全版本                |
| 小米        | MIUI 10.2 及以上版本    |
| vivo      | FuntouchOS 9 及以上版本 |
| OPPO      | Color OS 7.0 及以上版本 |
| Lenovo    | ZUI 11.4 及以上版本     |
| 三星        | Android 10 版本      |
| 魅族        | Android 10 版本      |
| 努比亚       | Android 10 版本      |
| 中兴        | Android 10 版本      |
| 华硕        | Android 10 版本      |
| 一加        | Android 10 版本      |
| Freeme OS | Android 10 版本      |
| Ssui OS   | Android 10 版本      |

## 二. SDK 获取方式

MSA 统一 SDK 下载地址:

移动安全联盟官网: <http://www.msa-alliance.cn/>

## 三. 调用方法

1、把 miit\_mdid\_x.x.x.aar 拷贝到项的 libs 目录, 并设置依赖, 其中 x.x.x 代表版本号。

2、将 supplierconfig.json 拷贝到项目 assets 目录下, 并修改 边对应内容, 特别是需要设置 appid 的部分。需要设置 appid 的部分需要去对应厂商的应用商店 注册自己的 app。

3、设置依赖

```
implementation files('libs/miit_mdid_x.x.x.aar')
```

4、混淆设置

```
-keep class com.bun.miitmdid.core.** {*;} 
```

5、设置 gradle 编译选项, 这块可以根据自己对平台的选择进 合 配置

```
ndk {
```

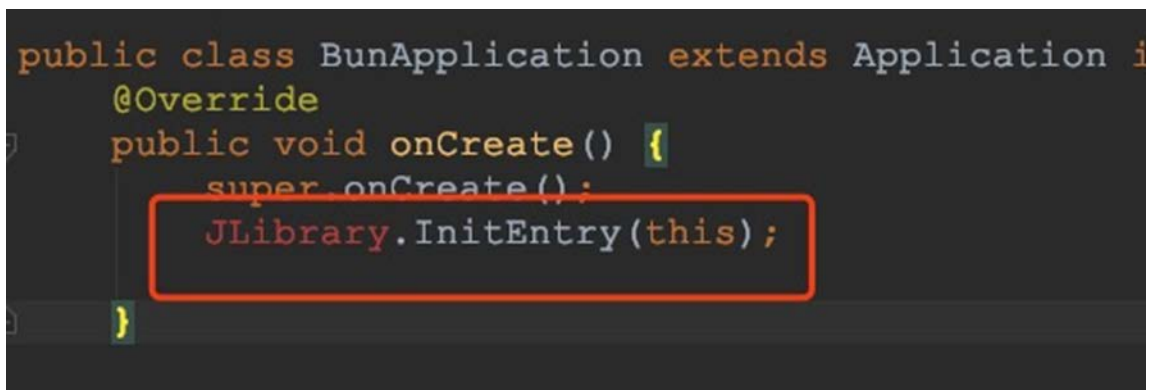
```
abiFilters 'armeabi-v7a', 'x86', 'arm64-v8a', 'x86_64', 'armeabi'
}
```

```
packagingOptions {
doNotStrip "*/armeabi-v7a/*.so"
doNotStrip "*/x86/*.so"
doNotStrip "*/arm64-v8a/*.so"
doNotStrip "*/x86_64/*.so"
doNotStrip "armeabi.so"
}
```

## 6、代码调用

### a、初始化 sdk

在应用的 **application** 的 **onCreate** 中方法调用方法：



```
public class BunApplication extends Application {
    @Override
    public void onCreate() {
        super.onCreate();
        JLibrary.InitEntry(this);
    }
}
```

**JLibrary.InitEntry(base);**

### b、获取设备 ID，有两种方法调用，事例代码详见附件 MiitHelper.java

b1：直接调用，由于安卓 9 以后类加载的限制，所以安卓 9 以后不建议采用，如需使用，需要做 **MdidSdk.InitSdk** 和 **JLibrary.InitEntry()** 不能在一个类里，还要注意包含这两个调用的类不能有依赖关系。

```
private int DirectCall(Context cxt){
    MdidSdk sdk = new MdidSdk();
    return sdk.InitSdk(cxt, this);
}
```

b2：反射调用，它的好处是 android 版本号无关，缺点是调用时间估计会长，经过测试跟直接调用在用时上区别不大。

```
private int CallFromReflect(Context cxt){
    return MdidSdkHelper.InitSdk(cxt, true, this);
}
```

}

c、初始化 sdk 返加的错误码

表一、错误信息，引用 ErrorCode 类

| 错误信息                              | 值       | 说明                         |
|-----------------------------------|---------|----------------------------|
| INIT_ERROR_MANUFACTURER_NOSUPPORT | 1008611 | 不支持的厂商                     |
| INIT_ERROR_DEVICE_NOSUPPORT       | 1008612 | 不支持的设备                     |
| INIT_ERROR_LOAD_CONFIGFILE        | 1008613 | 加载配置文件失败                   |
| INIT_ERROR_RESULT_DELAY           | 1008614 | 信息将会延迟返回，获取数据可能在异步线程，取决于设备 |
| INIT_HELPER_CALL_ERROR            | 1008615 | 反射调用失败                     |

#### 四. IdSupplier 接口说明

补充设备标识获取接口包括补充设备标识状态获取接口、匿名设备标识符获取接口、开发者匿名设备标识符获取接口、应用匿名设备标识符获取接口和关闭接口。

##### 1、补充设备标识状态获取接口

该接口用于获取移动智能终端是否支持补充设备标识体系，确认支持后，可以继续获取所需设备标识符。

**public boolean isSupported()**

| 参数 | 返回                     | 说明                   |
|----|------------------------|----------------------|
| 无  | boolean: 是否支持补充设备标识符获取 | true 为支持, false 为不支持 |

##### 2、匿名设备标识符获取接口

**String getOAID()**

| 参数 | 返回                     | 说明   |
|----|------------------------|--|
| 无  | String: 返回匿名设备标识符或空字符串 | 匿名设备标识符最长 64 位，返回空字符串表示不支持，异常状态包括网络异常、appid 异常、应用异常等 |

##### 3、开发者匿名设备标识符获取接口

**String getVAID()**

| 参数 | 返回                | 说明           |
|----|-------------------|--------------|
| 无  | String: 返回开发者匿名设备 | 开发者匿名设备标识符最长 |

|  |          |  |
|--|----------|--|
|  | 标识符或空字符串 | 64 位，返回空字符串表示不支持，异常状态包括网络异常、appid 异常、应用异常等 |
|--|----------|--|

#### 4、应用匿名设备标识符获取接口

**String getAAID()**

| 参数 | 返回                       | 说明   |
|----|--------------------------|--|
| 无  | String: 返回应用匿名设备标识符或空字符串 | 应用匿名设备标识符最长 64 位，返回空字符串表示不支持，异常状态包括网络异常、appid 异常、应用异常等 |

#### 5、关闭接口

**String shutDown()**

| 参数 | 返回   | 说明   |
|----|------|--|
| 无  | void | 有些厂商使用 bind service 实现的，所以使用完 IdSupplier 对象之后，要调用该方法释放连接 |

## 五. 使用建议

1、调用 MdidSdk 的 InitSdk 后，先检查返回值，如果返回值是不支持的设备或厂商，监听器也不会触发，如果是加载配置文件失败，联系我们的客服。

2、VAID/AAID 在初次调用时生成，生成需要访问网络，请确保网络通畅并可访问公网。

3、同一设备中存有多多个同一开发者应用，若需在单个应用卸载时保证 VAID 不被重置，需在应用被卸载前，已有另外同一开发者 ID 的应用也读取过 VAID，否则认定该开发者无需使用 VAID，值将被重置。

4、部分厂商，若应用未在其开发者平台后台上架，则认定未非法应用，无法生成 VAID，手机 LOG 中将会有相关异常值输出。

5、在用户手机处于弱网、无法访问公网或非法应用情形下频繁调用 VAID 和 AAID 读取接口，终端会累计其调用次数，并限时限制其读取。

6、由于返回值可能为 null，使用逻辑判断中建议做判空处理。

7、若有表一中的异常出现，会有相关 Log 打出。

8、INIT\_ERROR\_DEVICE\_NOSUPPORT 和 INIT\_HELPER\_CALL\_ERROR 这两个暂时不会走回调，后续会调整。

## 六. F&Q

详情参见《移动智能终端补充设备标识体系统一调用 SDK F&Q》，其他任何疑问可扫描下方二维码或发送邮件至 [msa@caict.ac.cn](mailto:msa@caict.ac.cn)，注明公司、联系人、问

题详情。



## 附录一

### 代码片断 - 初始化 sdk

```
public class BunApplication extends Application {  
    @Override  
    public void onCreate() {  
        super.onCreate();  
    }  
  
    @Override  
    protected void attachBaseContext(Context base) {  
        super.attachBaseContext(base);  
        JLibrary.InitEntry(base);  
    }  
}
```



## 代码片断 - 调用功能—MiitHelper.java

```
public class MiitHelper implements IIdentifierListener {

    private AppIdsUpdater _listener;

    public MiitHelper(AppIdsUpdater callback) {
        _listener=callback;
    }

    public void getDeviceIds(Context cxt){
        long timeb=System.currentTimeMillis();
        int nres = CallFromReflect(cxt);
        //      int nres=DirectCall(cxt);
        long timee=System.currentTimeMillis();
        long offset=timee-timeb;
        if(nres == ErrorCode.INIT_ERROR_DEVICE_NOSUPPORT) { //不支持
的设备

            }else if( nres == ErrorCode.INIT_ERROR_LOAD_CONFIGFILE) { //
加载配置文件出错

            }else if(nres ==
ErrorCode.INIT_ERROR_MANUFACTURER_NOSUPPORT) { //不支持的设备厂商

            }else if(nres == ErrorCode.INIT_ERROR_RESULT_DELAY) { //获取
接口是异步的，结果会在回调中返回，回调执行的回调可能在工作线程

            }else if(nres == ErrorCode.INIT_HELPER_CALL_ERROR) { //反射调
用出错

        }
    }
}
```

```
Log.d(getClass().getSimpleName(), "return value:
"+String.valueOf(nres));

}

/*
 * 通过反射调用，解决 android 9 以后的类加载升级，导致找不到 so 中
的方法
 *
 * */
private int CallFromReflect(Context cxt) {
    return MdidSdkHelper.InitSdk(cxt, true, this);
}

/*
 * 直接 java 调用，如果这样调用，在 android 9 以前没有题，在
android 9 以后会抛找不到 so 方法的异常
 * 解决办法是和 JLibrary.InitEntry(cxt)，分开调用，比如在 A 类中
调用 JLibrary.InitEntry(cxt)，在 B 类中调用 MdidSdk 的方法
 * A 和 B 不能存在直接和间接依赖关系，否则也会报错
 *
 * */
/* private int DirectCall(Context cxt) {
    MdidSdk sdk = new MdidSdk();
    return sdk.InitSdk(cxt, this);
}*/

@Override
public void OnSupport(boolean isSupport, IdSupplier _supplier)
{
    if(_supplier==null) {
        return;
    }
}
```

```
    }  
    String oaid=_supplier.getOAID();  
    String vaid=_supplier.getVAID();  
    String aaid=_supplier.getAAID();  
    String udid=_supplier.getUDID();  
    StringBuilder builder=new StringBuilder();  
    builder.append("support:"  
").append(isSupport?"true":"false").append("\n");  
    builder.append("UDID: ").append(udid).append("\n");  
    builder.append("OAID: ").append(oaid).append("\n");  
    builder.append("VAID: ").append(vaid).append("\n");  
    builder.append("AAID: ").append(aaid).append("\n");  
    String idstext=builder.toString();  
    _supplier.shutdown();  
    if(_listener!=null){  
        _listener.OnIdsAvalid(idstext);  
    }  
}  
  
public interface AppIdsUpdater{  
    void OnIdsAvalid(@NonNull String ids);  
}
```