02 设备接入 IoT Device SDK API 参考(C)

02 设备接入 IoT Device SDK API 参考(C)

文档版本 01

发布日期 2020-05-26

华为技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： https://www.huawei.com

客户服务邮箱： support@huawei.com

客户服务电话： 4008302118

目 录

1 使用前必读

概述

物联网平台提供了IoT Device SDK(以下简称SDK)，帮助设备快速连接到物联网平台。支持TCP/IP协议栈的设备在集成IoT Device SDK后，可以直接与物联网平台通信。不支持TCP/IP协议栈的设备例如蓝牙设备、ZigBee设备等，可以通过集成了IoT Device SDK的网关将设备数据转发给物联网平台，与平台进行通信。

接口列表

SDK提供的接口功能如下所示。

- 直连设备：通过设备鉴权直接接入物联网平台的设备。
- 非直连设备：通过网关设备接入物联网平台的设备。

功能	接口	说明
直连设备接入	IOTA_Init	**3.1 初始化SDK资源**
	IOTA_Destroy	**3.2 释放SDK资源**
	IOTA_ConfigSetXXX /	**3.3 绑定配置**
	IOTA_SetPrintLogCallback	**3.4 设置日志打印回调函数**
	IOTA_SetCallbackXXX	**3.5 设置回调函数**
	IOTA_Auth	**3.6 设备连接**
	IOTA_DisAuth	**3.7 设备断开链路**
直连设备上报数据	IOTA_MessageReport	**4.1 直连设备上报自定义topic消息**
	IOTA_PropertiesReport	**4.2 直连设备上报属性**
	IOTA_CommandResponse	**4.3 直连设备上报命令执行结果**
	IOTA_PropertiesSetResponse	**4.4 直连设备上报设备属性设置结果**

功能	接口	说明
	IOTA_PropertiesGetResponse	4.5 直连设备上报设备属性查询结果
	IOTA_UpdateSubDeviceStatus	4.6 直连设备上报子设备状态
直连设备命令接收(相关回调接口)	HW_VOID (*PFN_CALLBACK_HANDLER)/ HW_VOID (*PFN_CALLBACK_HANDLER_WITH_TOPIC)	5.1 接收消息下发
		5.3 接收命令下发
		5.4 接收平台设置设备属性
		5.5 接收平台查询设备属性
		5.6 接收新增子设备通知
		5.7 接收删除子设备通知
子设备上报数据	IOTA_MessageReport	6.1 子设备上报消息
	IOTA_BatchPropertiesReport	6.2 批量上报设备属性
OTA升级	HW_VOID (*PFN_CALLBACK_HANDLER)	7.1 平台下发获取版本信息通知
	IOTA_OTAVersionReport	7.2 设备上报软固件版本
	HW_VOID (*PFN_CALLBACK_HANDLER)	7.3 平台下发升级通知
	IOTA_GetOTAPackages	7.4 设备请求下载包
	IOTA_OTAStatusReport	7.5 设备上报升级状态
设备影子	IOTA_GetDeviceShadow	8.1 设备请求获取平台的设备影子数据
	HW_VOID (*PFN_CALLBACK_HANDLER_WITH_TOPIC)	8.2 设备接收平台返回的设备影子数据

2 开发说明

2.1 数据类型说明

常用数据类型

类型名称	类型原型
HW_INT	int
HW_UINT	unsigned int
HW_CHAR	char
HW_UCHAR	unsigned char
HW_BOOL	int
HW_ULONG	unsigned long
HW_USHORT	unsigned short
HW_MSG	void*
HW_VOID	void
HW_NULL	0

函数标准返回值

返回值名称	值	类型原型
IOTA_SUCCESS	0	执行成功。
IOTA_FAILURE	-1	执行错误。

返回值名称	值	类型原型
IOTA_PARAMETER_EMPTY	-101	参数为空。
IOTA_RESOURCE_NOT_AVAILABLE	-102	资源不被允许。
IOTA_INITIALIZATION_REPEATED	-103	重复初始化。
IOTA_LIBRARY_LOAD_FAILED	-104	加载库文件失败。
IOTA_SECRET_ENCRYPT_FAILED	-105	加密密钥失败。
IOTA_MQTT_CONNECT_FAILED	-106	连接失败。
IOTA_MQTT_CONNECT_EXISTED	-107	连接已存在。
IOTA_CERTIFICATE_NOT_FOUND	-108	找不到证书。
IOTA_MQTT_DISCONNECT_FAILED	-109	断链失败。
IOTA_PARSE_JSON_FAILED	-110	解析字符串失败。
IOTA_PARAMETER_ERROR	-111	参数错误。
IOTA_NUMBER_EXCEEDS	-112	数字超过最大值。

3 直连设备接入

3.1 初始化 SDK 资源

接口功能

初始化SDK资源。

接口描述

```
HW_INT IOTA_Init(HW_CHAR *pcWorkPath)
```

参数说明

字段	必选/可选	类型	描述
pcWorkPath	必选	String	SDK工作路径，用于存放SDK的配置文件，工作路径必须有效，该参数必须带结束符‘\0’。建议设置为"."(当前路径)，默认的证书文件在当前路径的conf文件夹下。

接口返回值

参见**函数标准返回值**

示例

```
// 开发者调用该接口初始化资源
IOTA_Init(".");//当前目录
```

3.2 释放 SDK 资源

接口功能

调用此函数，SDK会释放申请的所有动态资源（内存、线程等等）。

接口描述

```
HW_INT IOTA_Destroy()
```

接口返回值

参见**函数标准返回值**

示例

```
// 开发者调用该接口销毁资源
IOTA_Destroy();
```

3.3 绑定配置

接口功能

配置SDK相关参数。

接口描述

```
HW_INT IOTA_ConfigSetStr(HW_INT iItem, HW_CHAR *pValue)
HW_INT IOTA_ConfigSetUint(HW_INT iItem, HW_UINT uiValue)
```


参数说明

字段	必选/可选	类型	描述
iItem (key)	必选	HW_UINT	设备绑定的配置项。 • 设备ID：EN_IOTA_CFG_DEVICEID • 设备密钥：EN_IOTA_CFG_DEVICESECRET • 平台IP：EN_IOTA_CFG_MQTT_ADDR • 平台端口：EN_IOTA_CFG_MQTT_PORT • 日志的facility类型：EN_IOTA_CFG_LOG_LOCAL_NUMBER（当用syslog时才生效。） • 日志的显示级别：EN_IOTA_CFG_LOG_LEVEL（当用syslog时才生效。） • MQTT链接保活时间：EN_IOTA_CFG_KEEP_ALIVE_TIME（客户端发送一个MQTT“ping”消息，不设置默认是120s • MQTT连接超时时间：EN_IOTA_CFG_CONNECT_TIMEOUT（不设置默认是30s） • MQTT尝试重连时间：EN_IOTA_CFG_RETRY_INTERVAL（不设置默认是10s） • 消息发布Qos设置：EN_IOTA_CFG_QOS（不设置默认设置为1） • 设备接入模式：EN_IOTA_CFG_AUTH_MODE（分为密码接入模式和证书接入模式）

字段	必选/可选	类型	描述
pValue/ uiValue (value)	必选	HW_CHAR */ HW_UINT	设置的值。 • 设备ID：设备注册时返回 • 设备密钥：设备注册时返回 • 平台IP：Agent Lite对接平台地址 • 平台端口：8883 • 日志的facility类型：记录日志的来源。可以选择LOG_LOCAL0~LOG_LOCAL7中的任意一个 • 日志的显示级别：可以按需选择LOG_ERR、LOG_WARNING、LOG_INFO、LOG_DEBUG中的一个 • MQTT链接保活时间：单位s • MQTT连接超时时间：单位s • MQTT尝试重连时间：单位s • 消息发布Qos设置：0：最多一次；1：至少一次；2是只发一次。不设置默认设置为1。 • 设备接入模式：EN_IOTA_CFG_AUTH_MODE_SECRET是密码接入模式，EN_IOTA_CFG_AUTH_MODE_CERT是证书接入模式

接口返回值

参见**函数标准返回值**

示例

```
// 开发者调用该接口设置参数
    IOTA_ConfigSetStr(EN_IOTA_CFG_MQTT_ADDR, serverIp_);
    IOTA_ConfigSetUint(EN_IOTA_CFG_MQTT_PORT, port_);
    IOTA_ConfigSetStr(EN_IOTA_CFG_DEVICEID, username_);
    IOTA_ConfigSetStr(EN_IOTA_CFG_DEVICESECRET, password_);
    IOTA_ConfigSetUint(EN_IOTA_CFG_AUTH_MODE, EN_IOTA_CFG_AUTH_MODE_SECRET);

#ifdef _SYS_LOG
    IOTA_ConfigSetUint(EN_IOTA_CFG_LOG_LOCAL_NUMBER, LOG_LOCAL7);
    IOTA_ConfigSetUint(EN_IOTA_CFG_LOG_LEVEL, LOG_INFO);
#endif
```

3.4 设置日志打印回调函数

接口功能

SDK的日志呈现方式可以自定义回调函数来实现。

接口描述

```
void IOTA_SetPrintLogCallback(PFN_LOG_CALLBACK_HANDLER pfnLogCallbackHandler);
```

参数说明

字段	必选/可选	类型	描述
pfnLogCallbackHandler	必选	PFN_LOG_CALLBACK_HANDLER	入参为自定义函数名称。

PFN_LOG_CALLBACK_HANDLER 类型：

```
HW_VOID (*PFN_LOG_CALLBACK_HANDLER)(int level, char* format, va_list args);
```

字段	必选/可选	类型	描述
level	必选	int	日志打印等级，可以填写如下四种： • EN_LOG_LEVEL_DEBUG • EN_LOG_LEVEL_INFO • EN_LOG_LEVEL_WARNING • EN_LOG_LEVEL_ERROR
format	必选	String	打印的日志内容 ，里面可以包含要输出的变量。
args	可选	String/int	要打印的变量值。

接口返回值

参见**函数标准返回值**

示例

```
//开发者自定义日志打印函数
void myPrintLog(int level, char* format, va_list args)
{
    vprintf(format, args); //日志打印在控制台
//  vsyslog(level, format, args); //日志打印在系统文件中
}

//开发者设置自定义日志打印函数
IOTA_SetPrintLogCallback(myPrintLog);
```


说明

1. 用vprintf函数打印的日志，显示在控制台中。
2. 用vsyslog函数打印的日志，显示在系统日志文件中。一般会在"/var/log/messages"文件里（可以考虑按需分包）。建议自行实现日志打印函数。
   1. 由于linux下的DEBUG日志需要在调试的时候查看，如果想让DEBUG日志打印在控制台上，可以调低DEBUG日志级别再打印。

      例如：

```
void myPrintLog(int level, char* format, va_list args)
{
    switch (level)
    {
        case EN_LOG_LEVEL_DEBUG:
            vsyslog(LOG_WARNING, format, args);//可以把DEBUG日志级别调低后打印
            break;
        case EN_LOG_LEVEL_INFO:
            vsyslog(LOG_INFO, format, args);
            break;
        case EN_LOG_LEVEL_WARNING:
            vsyslog(LOG_WARNING, format, args);
            break;
        case EN_LOG_LEVEL_ERROR:
            vsyslog(LOG_ERR, format, args);
            break;
        default:
            break;
    }
}
```

3.5 设置回调函数

接口功能

设备收到下行数据时，开发者可以通过提前设置回调函数，来对下行数据进行处理，下行数据主要包括协议层和业务层，SDK已实现自动订阅业务层相关的TOPIC ，业务层下行数据可参考接口参数说明。

接口描述

IOTA_SetCallbackWithTopic函数与IOTA_SetCallback函数第二个入参不同，PFN_CALLBACK_HANDLER_WITH_TOPIC函数的参数比PFN_CALLBACK_HANDLER函数的参数多一个requestId（消息的唯一标识）。

IOTA_SetCallbackWithTopic函数主要用于处理以下命令：设备收到下发的命令、设备收到属性设置命令、设备收到属性查询命令。

IOTA_SetCallback函数主要用于处理以下通知：协议层通知、设备收到下发的透传消息、设备收到子设备相关命令。

```
void IOTA_SetCallback(HW_INT iItem, PFN_CALLBACK_HANDLER pfnCallbackHandler)
void IOTA_SetCallbackWithTopic(HW_INT iItem, PFN_CALLBACK_HANDLER_WITH_TOPIC
pfnCallbackHandler)
```


参数说明

字段	必选/可选	类型	描述
iItem	必选	HW_INT	回调函数对应的通知： • 协议层： 1. 鉴权成功的通知：EN_IOTA_CALLBACK_CONNECT_SUCCESS 2. 鉴权失败的通知：EN_IOTA_CALLBACK_CONNECT_FAILURE 3. 链接断开的通知：EN_IOTA_CALLBACK_CONNECTION_LOST 4. 设备主动断链成功的通知：EN_IOTA_CALLBACK_DISCONNECT_SUCCESS 5. 设备主动断链失败的通知：EN_IOTA_CALLBACK_DISCONNECT_FAILURE 6. 设备订阅成功的通知：EN_IOTA_CALLBACK_SUBSCRIBE_SUCCESS 7. 设备订阅失败的通知：EN_IOTA_CALLBACK_SUBSCRIBE_FAILURE 8. 设备发布数据成功的通知：EN_IOTA_CALLBACK_PUBLISH_SUCCESS 9. 设备发布数据失败的通知：EN_IOTA_CALLBACK_PUBLISH_FAILURE • 业务层 1. 设备收到下发的透传消息：EN_IOTA_CALLBACK_MESSAGE_DOWN 2. 设备收到下发的命令：EN_IOTA_CALLBACK_COMMAND_REQUES 3. 设备收到属性设置命令：EN_IOTA_CALLBACK_PROPERTIES_SET 4. 设备收到属性查询命令：EN_IOTA_CALLBACK_PROPERTIES_GET 5. 设备收到事件相关命令（**子设备新增/删除、OTA升级命令均属**

字段	必选/可选	类型	描述
			于事件）：EN_IOTA_CALLBACK_EVENT_DOWN 6. 设备收到自定义topic消息：EN_IOTA_CALLBACK_USER_TOPIC 7. 设备收到影子数据：EN_IOTA_CALLBACK_DEVICE_SHADOW
pfnCallbackHandler	必选	PFN_CALLBACK_HANDLER	入参为自定义函数名称
pfnCallbackHandler	必选	PFN_CALLBACK_HANDLER_WITH_TOPIC	入参为自定义函数名称

PFN_CALLBACK_HANDLER类型：

```
HW_VOID (*PFN_CALLBACK_HANDLER)(HW_VOID* context, HW_INT messageId, HW_INT code, HW_CHAR *message);
```

PFN_CALLBACK_HANDLER_WITH_TOPIC 类型：

```
HW_VOID (*PFN_CALLBACK_HANDLER_WITH_TOPIC)(HW_VOID* context, HW_INT messageId, HW_INT code, HW_CHAR *message, HW_CHAR *requestId);
```

字段	必选/可选	类型	描述
context	必选	HW_VOID	指向任何应用程序特定上下文的指针。上下文指针被传递给每个回调函数，以提供对回调中的上下文信息的访问。
messageId	必选	HW_INT	消息ID
code	必选	HW_INT	消息码（业务层通知该值默认为0）
message	必选	HW_CHAR*	消息体
requestId	必选	HW_CHAR*	请求的唯一标识，设备收到的消息带该参数时，响应消息需要将该参数值返回给平台。

示例

```
// 开发者注册设置回调函数
void setMyCallbacks()
{
    IOTA_SetCallback(EN_IOTA_CALLBACK_CONNECT_SUCCESS, handleAuthSuccess);
    IOTA_SetCallback(EN_IOTA_CALLBACK_CONNECT_FAILURE, handleAuthFailure);
    IOTA_SetCallback(EN_IOTA_CALLBACK_CONNECTION_LOST, handleConnectionLost);
```



```
    IOTA_SetCallback(EN_IOTA_CALLBACK_DISCONNECT_SUCCESS, handleDisAuthSuccess);
    IOTA_SetCallback(EN_IOTA_CALLBACK_DISCONNECT_FAILURE, handleDisAuthFailure);

    IOTA_SetCallback(EN_IOTA_CALLBACK_SUBSCRIBE_SUCCESS, handleSubscribesuccess);
    IOTA_SetCallback(EN_IOTA_CALLBACK_SUBSCRIBE_FAILURE, handleSubscribeFailure);

    IOTA_SetCallback(EN_IOTA_CALLBACK_PUBLISH_SUCCESS, handlePublishSuccess);
    IOTA_SetCallback(EN_IOTA_CALLBACK_PUBLISH_FAILURE, handlePublishFailure);

    IOTA_SetCallback(EN_IOTA_CALLBACK_MESSAGE_DOWN, handleMessageDown);
    IOTA_SetCallbackWithTopic(EN_IOTA_CALLBACK_COMMAND_REQUEST, handleCommandRequest);
    IOTA_SetCallbackWithTopic(EN_IOTA_CALLBACK_PROPERTIES_SET, handlePropertiesSet);
    IOTA_SetCallbackWithTopic(EN_IOTA_CALLBACK_PROPERTIES_GET, handlePropertiesGet);
    IOTA_SetCallback(EN_IOTA_CALLBACK_EVENT_DOWN, HandleEventsDown);
    IOTA_SetCallbackWithTopic(EN_IOTA_CALLBACK_USER_TOPIC, HandleUserTopicMessageDown);
    IOTA_SetCallbackWithTopic(EN_IOTA_CALLBACK_DEVICE_SHADOW, HandleDeviceShadowRsp);
}
```

3.6 设备连接

接口功能

设备可以调用鉴权接口来连接IoT平台。

接口描述

```
HW_INT IOTA_Connect()
```

接口返回值

参见**函数标准返回值**

示例

```
 //开发者调用鉴权接口
int ret = IOTA_Connect();
if (ret != 0)
{
    printfLog(EN_LOG_LEVEL_ERROR, "AgentLiteDemo: IOTA_Connect() error, Auth failed, result %d\n", ret);
}
```

3.7 设备断开链路

接口功能

设备可以调用断开链路接口主动与IoT平台断开。

接口描述

```
HW_INT IOTA_DisConnect()
```

接口返回值

参见**函数标准返回值**

示例

```
// 开发者调用断开链路接口：
int ret = IOTA_DisConnect();
if (ret != 0)
{
   printfLog(EN_LOG_LEVEL_ERROR, "AgentLiteDemo: IOTA_DisConnect() error, DisAuth failed, result %d\n",
ret);
}
```


4 直连设备上报数据

4.1 直连设备上报自定义 topic 消息

接口功能

可以通过该接口上报平台不解析的透传消息。

接口描述

```
HW_INT IOTA_MessageReport(HW_CHAR *object_device_id, HW_CHAR *name, HW_CHAR *id, HW_CHAR *content, HW_CHAR *topicParas)
```

参数说明

字段名	必选/可选	类型	参数描述
object_device_id	可选	HW_CHAR*	消息对应的最终目标设备，传NULL则表示目标设备即网关设备。
name	可选	HW_CHAR*	消息名称
id	可选	HW_CHAR*	消息的唯一标识
content	必选	HW_CHAR*	消息内容。

字段名	必选/可选	类型	参数描述
topicParas	可选	HW_CHAR*	自定义topic参数，例如“devMsg”（前面不要加'/'或者特殊字符），如果设置为NULL，则用平台默认topic上报。

接口返回值

参见函数标准返回值

示例

```
// 开发者调用该接口进行消息上报
void Test_messageReport()
{
    //default topic
//    int messageId = IOTA_MessageReport(NULL, "data123", "123", "hello", NULL);

    //user topic
    int messageId = IOTA_MessageReport(NULL, "data123", "123", "hello", "devMsg");
    if (messageId != 0) {
        PrintfLog(EN_LOG_LEVEL_ERROR, "AgentLiteDemo: Test_messageReport() failed, messageId %d\n",
messageId);
    }
}
```

4.2 直连设备上报属性

接口功能

网关可以该接口上报平台解析的设备属性值，设备属性需与profile中设置的一致。

接口描述

```
HW_INT IOTA_PropertiesReport(ST_IOTA_SERVICE_DATA_INFO pServiceData[], HW_INT serviceNum)
```

参数说明

字段	必选/可选	类型	描述
pServiceData[]	必选	ST_IOTA_SERVICE_DATA_INFO	上报的属性数据结构体
serviceNum	必选	HW_INT	上报的服务个数

ST_IOTA_SERVICE_DATA_INFO 类型：

字段	必选/可选	类型	描述
service_id	必选	HW_CHAR*	设备服务的ID，可从profile中获取
event_time	可选	HW_CHAR*	设备采集数据UTC时间（格式：yyyyMMddTHHmmssZ），如：20161219T114920Z。 设备上报数据将参数设置为NULL时，则数据上报时间以平台时间为准。
properties	必选	HW_CHAR*	一个服务的数据，具体字段在profile里定义。注意：格式需能解析成JSON，可参考下方的样例。

接口返回值

参见**函数标准返回值**

示例

```
// 开发者上报设备属性
void Test_propertiesReport()
{
    int serviceNum = 2;//网关要上报的service个数
    ST_IOTA_SERVICE_DATA_INFO services[serviceNum];

    //---------------the data of service1--------------------------------
    char *service1 = "{\"mno\":\"5\",\"imsi\":\"6\"}";

//    services[0].event_time = GetEventTimesStamp();
    services[0].event_time = NULL;
    services[0].service_id = "LTE";
    services[0].properties = service1;

    //---------------the data of service2--------------------------------
    char *service2 = "{\"hostCpuUsage\":\"4\",\"containerCpuUsage\":9}";

//    services[1].event_time =  GetEventTimesStamp();
    services[0].event_time = NULL;
    services[1].service_id = "CPU";
    services[1].properties = service2;

    int messageId = IOTA_PropertiesReport(services, serviceNum);
    if(messageId != 0)
    {
        printfLog(EN_LOG_LEVEL_ERROR, "AgentLiteDemo: Test_propertiesReport() failed, messageId %d\n",
messageId);
    }
}
```

4.3 直连设备上报命令执行结果

接口功能

当网关收到平台下发的命令（5.3）后，可以调用该接口上报命令执行结果。

接口描述

```
HW_INT IOTA_CommandResponse(HW_CHAR *requestId, HW_INT result_code, HW_CHAR *response_name,
HW_CHAR *pcCommandRespense)
```

参数说明

字段	必选/可选	类型	描述
requestId	必选	HW_CHAR*	请求的唯一标识，设备收到的消息带该参数时，响应消息需要将该参数值返回给平台。
result_code	必选	HW_INT	标识命令的执行结果，0表示成功，其他表示失败。不带默认认为成功。
response_name	可选	HW_CHAR*	命令的响应名称，在设备关联的产品模型中定义。
pcCommandResponse	必选	HW_CHAR*	命令响应结果，需跟profile中定义的commandResponse保持一致。注意：格式需能解析成JSON，可参考下方的样例。

接口返回值

参见**函数标准返回值**

示例

```
// 开发者调用该接口上报命令执行结果
    char *pcCommandResponse = "{\"SupWh\": \"aaa\"}"; // in service accumulator

    int result_code = 0;
    char *response_name = "cmdResponses";
        char *requestId = "1005";

    int messageId = IOTA_CommandResponse(requestId, result_code, response_name, pcCommandResponse);
    if(messageId != 0)
    {
        printfLog(EN_LOG_LEVEL_ERROR, "AgentLiteDemo: Test_commandResponse() failed, messageId %d
\n", messageId);
    }
```

4.4 直连设备上报设备属性设置结果

接口功能

当网关收到平台下发的设置设备属性命令（5.4）后，可以调用该接口上报结果。

接口描述

```
HW_INT IOTA_PropertiesSetResponse(HW_CHAR *requestId, HW_INT result_code, HW_CHAR *result_desc)
```


参数说明

字段	必选/可选	类型	描述
requestId	必选	HW_CHAR*	请求的唯一标识，设备收到的消息带该参数时，响应消息需要将该参数值返回给平台。
result_code	必选	Integer	命令的执行结果，0表示成功，其他表示失败。不带默认认为成功。
result_desc	可选	String	属性设置的响应描述，可以设置为NULL。

接口返回值

参见**函数标准返回值**

示例

```
// 开发者调用该接口上报属性设置结果
//    int messageId = IOTA_PropertiesSetResponse(requestId, 0, "success");
    int messageId = IOTA_PropertiesSetResponse(requestId, 0, NULL);
    if(messageId != 0)
    {
        printfLog(EN_LOG_LEVEL_ERROR, "AgentLiteDemo: Test_propSetResponse() failed, messageId %d\n",
messageId);
    }
```

4.5 直连设备上报设备属性查询结果

接口功能

当网关收到平台下发的查询设备属性命令（5.5）后，可以调用该接口上报结果。

接口描述

```
HW_API_FUNC HW_INT IOTA_PropertiesGetResponse(HW_CHAR *requestId, ST_IOTA_SERVICE_DATA_INFO
serviceProp[], HW_INT serviceNum)
```

参数说明

字段	必选/可选	类型	描述
requestId	必选	HW_CHAR *	请求的唯一标识，设备收到的消息带该参数时，响应消息需要将该参数值返回给平台。
serviceProp	可选	ST_IOTA_SERVICE_DATA_INFO	设备属性结构体。 注意： 如果不携带该参数，需将serviceNum设置为0

字段	必选/可选	类型	描述
serviceNum	必选	HW_INT	要上报的服务个数

ST_IOTA_SERVICE_DATA_INFO 类型：

字段	必选/可选	类型	描述
service_id	必选	HW_CHAR *	设备服务的ID，可从profile中获取
event_time	可选	HW_CHAR *	设备采集数据UTC时间（格式：yyyyMMddTHHmmssZ），如：20161219T114920Z。 设备上报数据将参数设置为NULL时，则数据上报时间以平台时间为准。
properties	必选	HW_CHAR *	一个服务的数据，具体字段在profile里定义。注意：格式需能解析成JSON，可参考下方的样例。

接口返回值

参见**函数标准返回值**

示例

```
// 开发者调用该接口上报属性查询结果
    int serviceNum = 2;
    ST_IOTA_SERVICE_DATA_INFO serviceProp[serviceNum];

    char *property = "{\"mno\":\"5\",\"imsi\":\"6\"}";

       //serviceProp[0].event_time = GetEventTimesStamp();
    serviceProp[0].event_time = NULL;
    serviceProp[0].service_id = "LTE";
    serviceProp[0].properties = property;

    char *property2 = "{\"hostCpuUsage\":\"2\",\"containerCpuUsage\":\"4\"}";

//    serviceProp[1].event_time = GetEventTimesStamp();
    serviceProp[1].event_time = NULL;
    serviceProp[1].service_id = "CPU";
    serviceProp[1].properties = property2;

    int messageId = IOTA_PropertiesGetResponse(requestId, serviceProp, serviceNum);
    if(messageId != 0)
    {
        printfLog(EN_LOG_LEVEL_ERROR, "AgentLiteDemo: Test_propGetResponse() failed, messageId %d\n",
messageId);
    }
```


4.6 直连设备上报子设备状态

接口功能

网关更新子设备状态。

接口描述

```
HW_INT IOTA_UpdateSubDeviceStatus(ST_IOTA_DEVICE_STATUSES *device_statuses, HW_INT deviceNum)
```

参数说明

字段	必选/可选	类型	描述
device_statuses	必选	ST_IOTA_DEVICE_STATUSES*	上报的子设备状态信息列表
deviceNum	必选	HW_INT	上报的子设备个数

ST_IOTA_SERVICE_DATA_INFO 类型：

字段	必选/可选	类型	描述
event_time	可选	String	事件时间
device_statuses[]	必选	ST_IOTA_DEVICE_STATUS	子设备状态的列表

ST_IOTA_DEVICE_STATUS类型：

字段	必选/可选	类型	描述
device_id	必选	String	子设备ID
status	必选	String	子设备状态： OFFLINE：设备离线 ONLINE：设备上线；

接口返回值

参见函数标准返回值

示例

```
// 开发者上报子设备状态
void Test_UpdateSubDeviceStatus(char *deviceId) {
    int deviceNum = 1;
```



```
    ST_IOTA_DEVICE_STATUSES device_statuses;
    device_statuses.event_time = NULL;
    device_statuses.device_statuses[0].device_id = deviceId;
    device_statuses.device_statuses[0].status = ONLINE;
    int messageId = IOTA_UpdateSubDeviceStatus(&device_statuses, deviceNum);
    if (messageId != 0) {
        PrintfLog(EN_LOG_LEVEL_ERROR, "device_demo: Test_UpdateSubDeviceStatus() failed, messageId %d
\n", messageId);
    }
}
```


5 直连设备命令接收

设备可以接受平台命令（SDK已自动实现相关TOPIC的订阅）。主要有如下命令：设备消息下发、平台命令下发、平台设置设备属性、平台查询设备属性、平台通知网关新增子设备、平台通知网关删除子设备。

开发者可以通过提前设置回调函数，来对命令进行处理，回调函数的设置方法及参数说明请参考3.5。本章节主要讲命令**消息体**（回调函数的入参**message**），及回调函数的实现。

5.1 接收消息下发

命令描述

设备收到应用下发的透传消息，该消息平台不解析。

消息体

字段名	必选/可选	类型	参数描述
object_device_id	可选	String	消息对应的最终目标设备，没有携带则表示目标设备即网关设备。
name	可选	String	消息名称
id	可选	String	消息的唯一标识

字段名	必选/可选	类型	参数描述
content	必选	String	消息内容。

说明

- 可选参数由第三方服务或者应用决定是否下发。后续消息体中的可选参数均由第三方决定是否下发。
- 可以通过JSON * root = JSON_Parse(message) 将平台下发的消息转化为JSON，再进行相关属性的获取。

示例

```
//开发者实现消息下发处理
void handleMessageDown(void* context, int messageId, int code, char *message)
{
    printfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: handleMessageDown(), messageId %d, code %d,
messsage %s\n", messageId, code, message);

    JSON * root =  JSON_Parse(message);  //Convert string to JSON

    char* content = JSON_GetStringFromObject(root, "content", "-1");     //get value of content
    printfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: handleMessageDown(), content %s\n", content);

    char* object_device_id = JSON_GetStringFromObject(root, "object_device_id", "-1");     //get value of
object_device_id
    printfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: handleMessageDown(), object_device_id %s\n",
object_device_id);

    char* name = JSON_GetStringFromObject(root, "name", "-1");     //get value of name
    printfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: handleMessageDown(), name %s\n", name);

    char* id = JSON_GetStringFromObject(root, "id", "-1");        //get value of id
    printfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: handleMessageDown(), id %s\n", id);
        JSON_Delete(root);
}


//设置回调函数
IOTA_SetCallback(EN_IOTA_CALLBACK_MESSAGE_DOWN, handleMessageDown);
```

5.2 接收自定义 topic 消息

设备侧需先订阅自定义topic，才能接收到自定义topic消息。

5.2.1 订阅自定义 topic

接口功能

可以通过该接口自定义topic。

接口描述

```
HW_INT IOTA_SubscribeUserTopic(HW_CHAR *topicParas)
```

参数说明

字段名	必选/可选	类型	参数描述
topicParas	必选	HW_CHAR*	自定义topic参数，例如“devMsg”（前面不要加'/'或者特殊字符）

接口返回值

参见**函数标准返回值**

示例

```
// 开发者调用该接口进行自定义topic订阅
IOTA_SubscribeUserTopic("devMsg");
```

5.2.2 接收自定义 topic 消息

命令描述

设备收到应用下发的自定义topic消息，该消息平台不解析。

消息体

字段名	必选/可选	类型	参数描述
object_device_id	可选	String	消息对应的最终目标设备，没有携带则表示目标设备即网关设备。
name	可选	String	消息名称
id	可选	String	消息的唯一标识
content	必选	String	消息内容。

说明

- 可选参数由第三方服务或者应用决定是否下发。后续消息体中的可选参数均由第三方决定是否下发。
- 可以通过JSON * root = JSON_Parse(message) 将平台下发的消息转化为JSON，再进行相关属性的获取。

示例

```
//开发者实现自定义topic消息下发处理
void HandleUserTopicMessageDown(void *context, int messageId, int code, char *message, char *topicParas) {
    PrintfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: HandleUserTopicMessageDown(), messageId %d, code %d, messsage %s, topicParas %s \n", messageId, code, message, topicParas);
    JSON *root = JSON_Parse(message);  //Convert string to JSON

    char *content = JSON_GetStringFromObject(root, "content", "-1");     //get value of content
    PrintfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: HandleUserTopicMessageDown(), content %s\n", content);
```



```
    char *object_device_id = JSON_GetStringFromObject(root, "object_device_id", "-1");     //get value of
object_device_id
    PrintfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: HandleUserTopicMessageDown(), object_device_id %s
\n", object_device_id);

    char *name = JSON_GetStringFromObject(root, "name", "-1");     //get value of name
    PrintfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: HandleUserTopicMessageDown(), name %s\n", name);

    char *id = JSON_GetStringFromObject(root, "id", "-1");        //get value of id
    PrintfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: HandleUserTopicMessageDown(), id %s\n", id);

    JSON_Delete(root);

}

//设置回调函数
IOTA_SetCallback(EN_IOTA_CALLBACK_MESSAGE_DOWN, handleMessageDown);
```

5.3 接收命令下发

命令描述

用于平台向设备下发设备控制命令。

消息体

字段名	必选/可选	类型	参数描述
object_device_id	可选	String	命令对应的目标设备ID
service_id	可选	String	设备的服务ID。
command_name	可选	String	设备命令名称，在设备关联的产品模型中定义。
paras	必选	Object	设备命令的执行参数，具体字段在设备关联的产品模型中定义。

示例

```
//开发者实现命令下发处理
void handleCommandRequest(void* context, int messageId, int code, char *message, char *requestId)
{
    printfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: handleCommandRequest(), messageId %d, code %d,
messsage %s, requestId %s\n", messageId, code, message, requestId);

    JSON * root =  JSON_Parse(message);  //Convert string to JSON

    char* object_device_id = JSON_GetStringFromObject(root, "object_device_id", "-1");     //get value of
object_device_id
    printfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: handleCommandRequest(), object_device_id %s\n",
object_device_id);

    char* service_id = JSON_GetStringFromObject(root, "service_id", "-1");     //get value of service_id
    printfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: handleCommandRequest(), content %s\n", service_id);

    char* command_name = JSON_GetStringFromObject(root, "command_name", "-1");     //get value of
```



```
command_name
    printfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: handleCommandRequest(), name %s\n",
command_name);

    JSON* paras = JSON_GetObjectFromObject(root, "device_list");        //get value of data
    printfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: handleCommandRequest(), id %s\n", paras);

    if (paras)
    {
        char *property = JSON_GetStringFromObject(paras, "Load", NULL);  //get value of Load defined in
profile

        printfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: handleCommandRequest(), Load %s\n", property);
    }

    Test_commandResponse(requestId);      //command reponse
        JSON_Delete(root);
}

//设置回调函数
IOTA_SetCallbackWithTopic(EN_IOTA_CALLBACK_COMMAND_REQUEST, handleCommandRequest);
```

说明

Test_commandResponse(requestId)函数里实现了命令响应逻辑，请参考4.3。

5.4 接收平台设置设备属性

命令描述

用于平台设置设备属性。

消息体

字段名	必选/可选	类型	参数描述
object_device_id	可选	String	属性设置对应的目标设备ID
services	必选	List<ServiceData>	设备服务数据列表。

ServiceData结构定义：

字段名	必选/可选	类型	参数描述
service_id	必选	String	设备的服务ID。
properties	必选	Object	设备服务的属性列表，具体字段在产品模型里定义。

示例

```
//开发者实现设置设备属性命令处理
void handlePropertiesSet(void* context, int messageId, int code, char *message, char *requestId)
{
    printfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: handlePropertiesSet(), messageId %d, code %d,
messsage %s, requestId %s\n", messageId, code, message, requestId);

    JSON * root =  JSON_Parse(message);  //Convert string to JSON

    char* object_device_id = JSON_GetStringFromObject(root, "object_device_id", "-1");     //get value of
object_device_id
    printfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: handlePropertiesSet(), object_device_id %s\n",
object_device_id);

    JSON* services = JSON_GetObjectFromObject(root, "services");                          //get  services array
    printfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: handlePropertiesSet(), services %s\n", services);

    int dataSize = JSON_GetArraySize(services);                                           //get length of services array
    printfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: handlePropertiesSet(), dataSize %d\n", dataSize);


    if (dataSize > 0)
    {
        JSON* service = JSON_GetObjectFromArray(services, 0);                 //only get the first one to
demonstrate
        printfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: handleSubDeviceMessageDown(), service %s\n",
service);
        if (service)
        {
            char *service_id = JSON_GetStringFromObject(service, "service_id", NULL);

            printfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: handlePropertiesSet(), service_id %s\n",
service_id);

            JSON* properties = JSON_GetObjectFromObject(service, "properties");

            char *Load = JSON_GetStringFromObject(service, "Load", NULL);
            printfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: handlePropertiesSet(), Load %s\n", Load);
        }
    }

    Test_propSetResponse(requestId);  //command response

    JSON_Delete(root);
}


//设置回调函数
IOTA_SetCallbackWithTopic(EN_IOTA_CALLBACK_PROPERTIES_SET, handlePropertiesSet);
```

说明

Test_propSetResponse(requestId)函数里实现了命令响应逻辑，请参考4.4。

5.5 接收平台查询设备属性

命令描述

用于平台向设备查询属性信息。

消息体

字段名	必选/可选	类型	参数描述
object_device_id	可选	String	属性查询对应的目标设备ID，不设置默认是查询网关设备。
service_id	可选	String	设备的服务ID，不设置默认查询全部service。

示例

```
//开发者实现查询设备属性命令处理
void handlePropertiesGet(void* context, int messageId, int code, char *message, char *requestId)
{
    printfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: handlePropertiesGet(), messageId %d, code %d, messsage %s, requestId %s\n", messageId, code, message, requestId);

    JSON * root =  JSON_Parse(message);

    char* object_device_id = JSON_GetStringFromObject(root, "object_device_id", "-1");     //get value of object_device_id
    printfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: handlePropertiesGet(), object_device_id %s\n", object_device_id);

    char* service_id = JSON_GetStringFromObject(root, "service_id", "-1");     //get value of service_id
    printfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: handlePropertiesGet(), service_id %s\n", service_id);

    Test_propGetResponse(requestId);  //command response

    JSON_Delete(root);
}

//设置回调函数
IOTA_SetCallbackWithTopic(EN_IOTA_CALLBACK_PROPERTIES_GET, handlePropertiesGet);
```

说明

Test_propGetResponse(requestId)函数里实现了命令响应逻辑，请参考4.5。

5.6 接收新增子设备通知

命令描述

平台将该网关新增的子设备列表信息通知给设备。

消息体

参数说明

字段名	必选/可选	类型	参数描述
object_device_id	可选	String	事件对应的最终目标设备，没有携带则表示目标设备即网关设备

字段名	必选/可选	类型	参数描述
services	可选	List<ServiceEvent>	事件服务列表

ServiceEvent定义表

字段名	必选/可选	类型	参数描述
service_id	必选	String	sub_device_manager
event_type	必选	String	add_sub_device_notify
event_time	可选	String	事件时间
paras	必选	Object	事件参数JSON对象

paras参数列表

字段名	必选/可选	类型	参数描述
devices	必选	List<DeviceInfo>	设备列表
version	必选	Long	子设备信息版本

DeviceInfo定义表

字段名	必选/可选	类型	参数描述
parent_device_id	必选	String	父节点设备ID
node_id	必选	String	设备标识。
device_id	必选	String	设备ID
name	可选	String	设备名称
description	可选	String	设备描述
manufacturer_id	可选	String	厂商ID
model	可选	String	设备型号
product_id	可选	String	产品ID
fw_version	可选	String	固件版本
sw_version	可选	String	软件版本

status	可选	String	设备在线状态 ONLINE：设备在线 OFFLINE：设备离线

数据格式

```
{
    "object_device_id": "{object_device_id}",
    "services": [{
        "service_id": "sub_device_manager",
        "event_type": "add_sub_device_notify",
        "event_time": "20151212T121212Z",
        "paras": {
            "devices": [{
                "parent_device_id": "c6b39067b0325db34663d3ef421a42f6_12345678",
                "node_id": "subdevice11",
                "device_id": "2bb4ddba-fb56-4566-8577-063ad2f5a6cc",
                "name": "subDevice11",
                "description": null,
                "manufacturer_id": "ofo",
                "model": "twx2",
                "product_id": "c6b39067b0325db34663d3ef421a42f6",
                "fw_version": null,
                "sw_version": null,
                "status": "ONLINE"
            }],
            "version": 1
        }
    }]
}
```

示例

```
//设置回调函数
IOTA_SetCallback(EN_IOTA_CALLBACK_EVENT_DOWN, HandleEventsDown);

//开发者实现事件相关命令处理（接收子设备新增/删除、OTA升级通知）
void HandleEventsDown(void *context, int messageId, int code, char *message) {
    PrintfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: HandleEventsDown(), messageId %d, code %d,
messsage %s\n", messageId, code, message);

    // the demo of how to get the parameter
    JSON *root = JSON_Parse(message);

    char *object_device_id = JSON_GetStringFromObject(root, "object_device_id", "-1");          //get value of
object_device_id
    PrintfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: HandleEventsDown(), object_device_id %s\n",
object_device_id);

    JSON *service = JSON_GetObjectFromObject(root, "services");                                 //get object of
services
    PrintfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: HandleEventsDown(), services %s\n", service);

    int dataSize = JSON_GetArraySize(service);                                                  //get size of services
    PrintfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: HandleEventsDown(), dataSize %d\n", dataSize);

    if (dataSize > 0) {
        JSON *serviceEvent = JSON_GetObjectFromArray(service, 0);                               //get object of
ServiceEvent
        PrintfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: HandleEventsDown(), serviceEvent %s\n",
serviceEvent);
        if (serviceEvent) {
            char *service_id = JSON_GetStringFromObject(serviceEvent, "service_id", NULL);    //get value of
service_id
```

```
        PrintfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: HandleEventsDown(), service_id %s\n",
service_id);

        char *event_type = NULL; //To determine whether to add or delete a sub device
        event_type = JSON_GetStringFromObject(serviceEvent, "event_type", NULL);    //get value of
event_type
        PrintfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: HandleEventsDown(), event_type %s\n",
event_type);

        char *event_time = JSON_GetStringFromObject(serviceEvent, "event_time", NULL);    //get value of
event_time
        PrintfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: HandleEventsDown(), event_time %s\n",
event_time);

        JSON *paras = JSON_GetObjectFromObject(serviceEvent, "paras");                    //get object
of paras
        PrintfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: HandleEventsDown(), paras %s\n", paras);

        //sub device manager
        if (!strcmp(service_id, "$sub_device_manager")) {

            JSON *devices = JSON_GetObjectFromObject(paras, "devices");                   //get object
of devices
            PrintfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: HandleEventsDown(), devices %s\n", devices);

            int version = JSON_GetIntFromObject(paras, "version", -1);                    //get value of
version
            PrintfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: HandleEventsDown(), version %d\n", version);

            int devicesSize = JSON_GetArraySize(devices);                                 //get size of
devicesSize
            PrintfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: HandleEventsDown(), devicesSize %d\n",
devicesSize);

            //add a sub device
            if (!strcmp(event_type, "add_sub_device_notify")) {

                if (devicesSize > 0) {
                    JSON *deviceInfo = JSON_GetObjectFromArray(devices, 0);               //get
object of deviceInfo
                    PrintfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: HandleEventsDown(), deviceInfo %s\n",
deviceInfo);

                    char *parent_device_id = JSON_GetStringFromObject(deviceInfo, "parent_device_id",
NULL);    //get value of parent_device_id
                    PrintfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: HandleEventsDown(), parent_device_id
%s\n", parent_device_id);

                    char *node_id = JSON_GetStringFromObject(deviceInfo, "node_id", NULL);    //get value of
node_id
                    PrintfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: HandleEventsDown(), node_id %s\n",
node_id);

                    subDeviceId = JSON_GetStringFromObject(deviceInfo, "device_id", NULL);    //get value of
device_id
                    PrintfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: HandleEventsDown(), device_id %s\n",
subDeviceId);

                    char *name = JSON_GetStringFromObject(deviceInfo, "name", NULL);    //get value of name
                    PrintfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: HandleEventsDown(), name %s\n",
name);

                    char *description = JSON_GetStringFromObject(deviceInfo, "description", NULL);    //get
value of description
                    PrintfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: HandleEventsDown(), description %s\n",
description);

                    char *manufacturer_id = JSON_GetStringFromObject(deviceInfo, "manufacturer_id",
NULL);    //get value of manufacturer_id
```

```
                    PrintfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: HandleEventsDown(), manufacturer_id
%s\n", manufacturer_id);

                    char *model = JSON_GetStringFromObject(deviceInfo, "model", NULL);    //get value of
model
                    PrintfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: HandleEventsDown(), model %s\n",
model);

                    char *product_id = JSON_GetStringFromObject(deviceInfo, "product_id", NULL);    //get
value of product_id
                    PrintfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: HandleEventsDown(), product_id %s\n",
product_id);

                    char *fw_version = JSON_GetStringFromObject(deviceInfo, "fw_version", NULL);    //get
value of fw_version
                    PrintfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: HandleEventsDown(), fw_version %s\n",
fw_version);

                    char *sw_version = JSON_GetStringFromObject(deviceInfo, "sw_version", NULL);    //get
value of sw_version
                    PrintfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: HandleEventsDown(), sw_version %s\n",
sw_version);

                    char *status = JSON_GetStringFromObject(deviceInfo, "status", NULL);    //get value of
status
                    PrintfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: HandleEventsDown(), status %s\n",
status);

                    //command response
                    Test_BatchPropertiesReport();
                }

            }

            //delete a sub device
            if (!strcmp(event_type, "delete_sub_device_notify")) {

                if (devicesSize > 0) {
                    JSON *deviceInfo = JSON_GetObjectFromArray(devices, 0);                    //get
object of deviceInfo
                    PrintfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: HandleEventsDown(), deviceInfo %s\n",
deviceInfo);

                    char *parent_device_id = JSON_GetStringFromObject(deviceInfo, "parent_device_id",
NULL);    //get value of parent_device_id
                    PrintfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: HandleEventsDown(), parent_device_id
%s\n", parent_device_id);

                    char *node_id = JSON_GetStringFromObject(deviceInfo, "node_id", NULL);    //get value of
node_id
                    PrintfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: HandleEventsDown(), node_id %s\n",
node_id);

                    subDeviceId = JSON_GetStringFromObject(deviceInfo, "device_id", NULL);    //get value of
device_id
                    PrintfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: HandleEventsDown(), device_id %s\n",
subDeviceId);

                }

            }

        }

        //OTA
        if (!strcmp(service_id, "$ota")) {

            if (!strcmp(event_type, "version_query")) {
                //report OTA version
```

```
                Test_ReportOTAVersion();
            }
            //firmware_upgrade or software_upgrade
            if ((!strcmp(event_type, "firmware_upgrade")) || (!strcmp(event_type, "software_upgrade"))) {
                ota_version = JSON_GetStringFromObject(paras, "version", NULL);    //get value of version
                PrintfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: HandleEventsDown(), version %s\n",
ota_version);

                char *url = JSON_GetStringFromObject(paras, "url", NULL);    //get value of url
                PrintfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: HandleEventsDown(), url %s\n", url);

                int file_size = JSON_GetIntFromObject(paras, "file_size", -1);    //get value of file_size
                PrintfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: HandleEventsDown(), file_size %d\n",
file_size);

                char *access_token = JSON_GetStringFromObject(paras, "access_token", NULL);    //get value
of access_token
                PrintfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: HandleEventsDown(), access_token %s\n",
access_token);

                int expires = JSON_GetIntFromObject(paras, "expires", -1);    //get value of expires
                PrintfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: HandleEventsDown(), expires %d\n",
expires);

                char *sign = JSON_GetStringFromObject(paras, "sign", NULL);    //get value of sign
                PrintfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: HandleEventsDown(), sign %s\n", sign);

                //start to receive packages and firmware_upgrade or software_upgrade

                if (IOTA_GetOTAPackages(url, access_token, 1000) == 0) {
                    usleep(3000 * 1000);
                    //report successful upgrade status
                    Test_ReportUpgradeStatus(0, ota_version);
                } else {
                    //report failed status
                    Test_ReportUpgradeStatus(-1, ota_version);
                }

            }

        }

    }
  }

  JSON_Delete(root);
}
```

说明

Test_batchPropertiesReport()函数里实现了命令响应逻辑，请参考6.2。

- if (!strcmp(event_type, "add_sub_device_notify")) 判断是新增子设备事件通知，于是给子设备上报一条数据。
- else if(!strcmp(event_type, "delete_sub_device_notify"))判断是删除子设备事件通知。

5.7 接收删除子设备通知

命令描述

平台将该网关删除的子设备信息通知给设备。

消息体

参数说明

字段名	必选/可选	类型	参数描述
object_device_id	可选	String	事件对应的最终目标设备，没有携带则表示目标设备即网关设备
services	可选	List<ServiceEvent>	事件服务列表

ServiceEvent定义表

字段名	必选/可选	类型	参数描述
service_id	必选	String	sub_device_manager
event_type	必选	String	delete_sub_device_notify
event_time	可选	String	事件时间
paras	必选	Object	事件参数JSON对象

paras参数列表

字段名	必选/可选	类型	参数描述
devices	必选	List<DeviceInfo>	设备列表
version	必选	Long	子设备信息版本

DeviceInfo定义表

字段名	必选/可选	类型	参数描述
parent_device_id	必选	String	父节点设备ID
node_id	可选	String	设备标识。
device_id	必选	String	设备ID

数据格式

```
{
    "object_device_id": "{object_device_id}",
    "services": [{
```



```
    "service_id": "sub_device_manager",
    "event_type": "delete_sub_device_notify",
    "event_time": "20151212T121212Z",
    "paras": {
        "devices": [{
            "parent_device_id": "c6b39067b0325db34663d3ef421a42f6_12345678",
            "node_id": "subdevice11",
            "device_id": "2bb4ddba-fb56-4566-8577-063ad2f5a6cc"
        }],
        "version": 1
    }
  }]
}
```

示例

请参考5.6示例。

说明

新增子设备和删除子设备通知TOPIC一致，可通过消息体中的event_type字段来进行判断。

6 子设备上报数据

6.1 子设备上报消息

请参考4.1，将object_device_id设置为子设备Id，即可通过网关设备上报子设备消息。

6.2 批量上报设备属性

接口功能

用于批量设备上报数据给平台。网关设备可以用此接口同时上报多个子设备的数据。

接口描述

```
HW_INT IOTA_BatchPropertiesReport(ST_IOTA_DEVICE_DATA_INFO pDeviceData[], HW_INT deviceNum,
HW_INT serviceLenList[])
```

参数说明

字段	必选/可选	类型	描述
pServiceData[]	必选	ST_IOTA_DEVICE_DATA_INFO	上报的设备数据结构体数组
deviceNum	必选	HW_INT	上报的子设备个数
serviceLenList[]	必选	HW_INT	子设备上报的服务个数数组

ST_IOTA_SERVICE_DATA_INFO 类型：

字段	必选/可选	类型	描述
device_id	必选	HW_CHAR *	设备ID
services[MaxServiceReportNum]	必选	ST_IOTA_SERVICE_DATA_INFO	一个服务的数据，具体字段在profile里定义。注意：格式需能解析成JSON，可参考下方的示例。MaxServiceReportNum表示一个设备一次数据包含的最大服务个数，默认为10，开发者可以自定义。

ST_IOTA_SERVICE_DATA_INFO 类型：

字段	必选/可选	类型	描述
service_id	必选	HW_CHAR *	设备服务的ID，可从profile中获取
event_time	可选	HW_CHAR *	设备采集数据UTC时间（格式：yyyyMMddTHHmmssZ），如：20161219T114920Z。 设备上报数据将参数设置为NULL时，则数据上报时间以平台时间为准。
properties	必选	HW_CHAR *	一个服务的数据，具体字段在profile里定义。注意：格式需能解析成JSON，可参考下方的样例。

示例

```
// 开发者通过该接口批量上报设备属性
    int deviceNum = 1;      //要上报的子设备的个数
    ST_IOTA_DEVICE_DATA_INFO devices[deviceNum]; //子设备要上报的结构体数组
    int serviceList[deviceNum];  //对应存储每个子设备要上报的服务个数
    serviceList[0] = 2;       //子设备一要上报两个服务
//    serviceList[1] = 1;          //子设备二要上报一个服务


    char *device1_service1 = "{\"mno\":\"1\",\"imsi\":\"3\"}"; //service1要上报的属性数据，必须是json格式

    char *device1_service2 = "{\"hostCpuUsage\":\"2\",\"containerCpuUsage\":\"4\"}";//service2要上报的属性
数据，必须是json格式

    devices[0].device_id = subDeviceId;
    devices[0].services[0].event_time = GetEventTimesStamp();
    devices[0].services[0].service_id = "LTE";
    devices[0].services[0].properties = device1_service1;


    devices[0].services[1].event_time = GetEventTimesStamp();
    devices[0].services[1].service_id = "CPU";
    devices[0].services[1].properties = device1_service2;



//    char *device2_service1 = "{\"AA\":\"2\",\"BB\":\"4\"}";
//    devices[1].device_id = "subDevices22222";
//    devices[1].services[0].event_time = "d2s1";
```



```
//    devices[1].services[0].service_id = "device2_service11111111";
//    devices[1].services[0].properties = device2_service1;


    int messageId = IOTA_BatchPropertiesReport(devices, deviceNum, serviceList);
    if(messageId != 0)
    {
        printfLog(EN_LOG_LEVEL_ERROR, "AgentLiteDemo: Test_batchPropertiesReport() failed, messageId %d
\n", messageId);
    }
```


7 OTA 升级

7.1 平台下发获取版本信息通知

命令描述

平台下发获取版本信息通知。

消息体

参数说明

字段名	必选/可选	类型	参数描述
object_device_id	可选	String	事件对应的最终目标设备，没有携带则表示目标设备即topic中指定的设备
services	可选	List<ServiceEvent>	事件服务列表

ServiceEvent定义表

字段名	必选/可选	类型	参数描述
service_id	必选	String	$ota

字段名	必选/可选	类型	参数描述
event_type	必选	String	version_query
event_time	可选	String	事件时间
paras	必选	Object	事件参数JSON对象

paras参数列表

字段名	必选/可选	类型	参数描述

数据格式

```
{
   "object_device_id": "{object_device_id}",
   "services": [{
      "service_id": "$ota",
      "event_type": "version_query",
      "event_time": "20151212T121212Z",
       "paras": {
       }
   }]
}
```

示例

```
请参考5.6示例。
```

7.2 设备上报软固件版本

接口功能

可以通过该接口上报当前软固件版本号。

接口描述

```
HW_INT IOTA_OTAVersionReport(ST_IOTA_OTA_VERSION_INFO otaVersionInfo)
```

参数说明

字段	必选/可选	类型	描述
otaVersionInfo	必选	ST_IOTA_OTA_VERSION_INFO	软固件版本数据结构体

ST_IOTA_OTA_VERSION_INFO 类型：

字段	必选/可选	类型	描述
sw_version	必选	HW_CHAR*	软件版本（和fw_version必须保证有一个不为NULL）
fw_version	必选	HW_CHAR*	固件版本（和sw_version必须保证有一个不为NULL）
event_time	可选	HW_CHAR*	设备采集数据UTC时间（格式：yyyyMMddTHHmmssZ），如：20161219T114920Z。 设备上报数据将参数设置为NULL时，则数据上报时间以平台时间为准。
object_device_id	可选	HW_CHAR*	事件对应的最终目标设备，没有携带则表示目标设备即topic中指定的设备

接口返回值

参见**函数标准返回值**

示例

```
// 开发者调用该接口进行软固件版本上报
void Test_ReportOTAVersion() {
    ST_IOTA_OTA_VERSION_INFO otaVersion;

    otaVersion.event_time = NULL;
    otaVersion.sw_version = "v1.0";
    otaVersion.fw_version = "v1.0";
    otaVersion.object_device_id = NULL;

    int messageId = IOTA_OTAVersionReport(otaVersion);
    if (messageId != 0) {
        PrintfLog(EN_LOG_LEVEL_ERROR, "AgentLiteDemo: Test_ReportOTAVersion() failed, messageId %d\n",
messageId);
    }
}
```

7.3 平台下发升级通知

命令描述

平台下发升级通知。

消息体

参数说明

字段名	必选/可选	类型	参数描述
object_device_id	可选	String	事件对应的最终目标设备，没有携带则表示目标设备即topic中指定的设备
services	可选	List<ServiceEvent>	事件服务列表

ServiceEvent定义表

字段名	必选/可选	类型	参数描述
service_id	必选	String	$ota
event_type	必选	String	固件升级：firmware_upgrade 软件升级：software_upgrade
event_time	可选	String	事件时间
paras	必选	Object	事件参数JSON对象

paras参数列表

字段名	必选/可选	类型	参数描述
version	必选	String	软固件包版本号
url	必选	String	软固件包下载地址
file_size	必选	Integer	软固件包文件大小
access_token	可选	String	软固件包url下载地址的临时token
expires	可选	Integer	access_token的超期时间
sign	必选	String	软固件包MD5值

数据格式

```
{
    "object_device_id": "{object_device_id}",
    "services": [{
        "service_id": "$ota",
        "event_type": "firmware_upgrade",
        "event_time": "20151212T121212Z",
        "paras": {
            "version": "v1.2",
```



```
            "url": "https://10.1.1.1:8943/iodm/inner/v1.3.0/firmwarefiles/ca1d954771ae61e5098c7f83",
            "file_size": 81362928,
            "access_token": "595124473f866b033dfa1f",
            "expires": 86400,
            "sign": "595124473f866b033dfa1f7e831c8c99a12f6143f392dfa996a819010842c99d"
        }
    }]
}
```

示例

```
请参考5.6示例。
```

7.4 设备请求下载包

接口功能

网关可以该接口请求下载软/固件包，下载完毕后会在当前目录下生成一个软/固件包。

接口描述

```
HW_INT IOTA_GetOTAPackages(HW_CHAR *url, HW_CHAR *token, HW_INT timeout)
```

参数说明

字段	必选/可选	类型	描述
url	必选	HW_CHAR*	软固件包下载地址
token	必选	HW_CHAR*	软固件包url下载地址的临时token
serviceNum	必选	timeout	请求下载包超时时间，值需要大于300秒。建议小于24h。

接口返回值

参见**函数标准返回值**

示例

```
请参考5.6示例。
```

7.5 设备上报升级状态

接口功能

当设备升级完成后，可以通过该接口上报升级状态。

接口描述

```
HW_INT IOTA_OTAStatusReport(ST_IOTA_UPGRADE_STATUS_INFO otaStatusInfo)
```


参数说明

字段	必选/可选	类型	描述
otaStatusInfo	必选	ST_IOTA_UPGRADE_STATUS_INFO	升级状态数据结构体

ST_IOTA_UPGRADE_STATUS_INFO 类型：

字段	必选/可选	类型	描述
result_code	必选	HW_INT	设备的升级状态，结果码定义如下： 0 处理成功 1 设备使用中 2 信号质量差 3 已经是最新版本 4 电量不足 5 剩余空间不足 6 下载超时 7 升级包校验失败 8 升级包类型不支持 9 内存不足 10 安装升级包失败 255 内部异常
progress	可选	HW_INT	设备的升级进度，范围：0到100
event_time	可选	HW_CHAR*	设备采集数据UTC时间（格式：yyyyMMddTHHmmssZ），如：20161219T114920Z。 设备上报数据将参数设置为NULL时，则数据上报时间以平台时间为准。
object_device_id	可选	HW_CHAR*	事件对应的最终目标设备，没有携带则表示目标设备即topic中指定的设备
description	可选	HW_CHAR*	升级状态描述信息，可以返回具体升级失败原因。
version	必选	HW_CHAR*	设备当前版本号

接口返回值

参见**函数标准返回值**

示例

```
// 开发者调用该接口进行升级状态上报
void Test_ReportUpgradeStatus(int i, char *version) {
    ST_IOTA_UPGRADE_STATUS_INFO statusInfo;
    if (i == 0) {
        statusInfo.description = "success";
        statusInfo.progress = 100;
        statusInfo.result_code = 0;
        statusInfo.version = version;
    } else {
        statusInfo.description = "failed";
        statusInfo.result_code = 1;
        statusInfo.progress = 0;
        statusInfo.version = version;
    }

    statusInfo.event_time = NULL;
    statusInfo.object_device_id = NULL;

    int messageId = IOTA_OTAStatusReport(statusInfo);
    if (messageId != 0) {
        PrintfLog(EN_LOG_LEVEL_ERROR, "AgentLiteDemo: Test_ReportUpgradeStatus() failed, messageId %d
\n", messageId);
    }
}
```


8 设备影子

8.1 设备请求获取平台的设备影子数据

接口功能

用于设备向平台获取设备影子数据。

接口描述

```
HW_INT IOTA_GetDeviceShadow(HW_CHAR *requestId, HW_CHAR *object_device_id, HW_CHAR *service_id)
```

参数说明

字段	必选/可选	类型	描述
requestId	必选	HW_CHAR*	请求的唯一标识，设备收到的消息带该参数时，响应消息需要将该参数值返回给平台。
service_id	可选	HW_CHAR*	需要获取设备影子的设备服务ID,不带的话查询所有服务ID的设备影子数据。
object_device_id	可选	HW_CHAR*	事件对应的最终目标设备，没有携带则表示目标设备即topic中指定的设备

接口返回值

参见**函数标准返回值**

示例

```
// 开发者调用该接口进行请求影子数据
      //get device shadow
   IOTA_GetDeviceShadow("1232", NULL, NULL);
```

8.2 设备接收平台返回的设备影子数据

命令描述

用于接收平台返回的设备影子数据。

消息体

字段名	必选/可选	类型	参数描述
object_device_id	必选	String	设备影子的目标设备ID。
shadow	可选	List<ShadowData>	服务影子数据。

ShadowData结构定义：

字段名	必选/可选	类型	参数描述
service_id	必选	String	设备的服务ID。
desired	可选	PropertiesData	设备影子desired区的属性列表。
reported	可选	PropertiesData	设备影子reported区的属性列表。
version	可选	Integer	设备影子版本信息

PropertiesData结构定义：

字段名	必选/可选	类型	参数描述
properties	必选	Object	设备服务的属性列表，具体字段在设备关联的产品模型里定义。
event_time	可选	String	设备属性数据的UTC时间（格式：yyyyMMddTHHmmssZ），如：20161219T114920Z。

示例

```
//开发者实现影子数据处理
void HandleDeviceShadowRsp(void *context, int messageId, int code, char *message, char *requestId) {
    PrintfLog(EN_LOG_LEVEL_INFO, "AgentLiteDemo: HandleDeviceShadowRsp(), messageId %d, code %d,
messsage %s, requestId %s\n", messageId, code, message, requestId);
    //Start analyzing data, please refer to function HandleEventsDown
}

//设置回调函数
IOTA_SetCallbackWithTopic(EN_IOTA_CALLBACK_DEVICE_SHADOW, HandleDeviceShadowRsp);
```

