02 设备接入 IoT Device SDK API 参考(C)-正式发布

02 设备接入 IoT Device SDK API 参考(C)-正式发布

文档版本 01

发布日期 2023-05-25

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暗示的声明或保证。

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_Connect	3.6 设备连接
	IOTA_DisConnect	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
	IOTA_AddSubDevice	4.7 直连设备上报新增子设备请求
	IOTA_DelSubDevice	4.8 直连设备上报删除子设备请求
topic订阅	HW_API_FUNC HW_INT IOTA_SubscribeUserTopic(HW_CHAR *topicParas)	5.2.1 订阅自定义topic
	HW_API_FUNC HW_VOID IOTA_SetUserTopicMsgCallback(PFN_USER_TOPIC_MSG_CALLBACK_HANDLER pfnCallbackHandler)	5.2.2 接收自定义topic消息
	HW_API_FUNC HW_INT IOTA_SubscribeBoostrap()	5.2.3 订阅bootstrap topic
直连设备命令接收(相关回调接口)	HW_VOID (*PFN_MESSAGE_CALLBACK_HANDLER)(EN_IOTA_MESSAGE *message)	5.1 接收消息下发
	HW_VOID (*PFN_CMD_CALLBACK_HANDLER)(EN_IOTA_COMMAND *message)	5.3 接收命令下发
	HW_VOID (*PFN_PROP_SET_CALLBACK_HANDLER) (EN_IOTA_PROPERTY_SET *message)	5.4 接收平台设置设备属性
	HW_VOID (*PFN_PROP_GET_CALLBACK_HANDLER) (EN_IOTA_PROPERTY_GET *message)	5.5 接收平台查询设备属性
	HW_VOID (*PFN_EVENT_CALLBACK_HANDLER)(EN_IOTA_EVENT *message)	5.6 接收平台新增子设备通知

功能	接口	说明
	HW_VOID (*PFN_EVENT_CALLBACK_HANDLER)(EN_IOTA_EVENT *message)	5.7 接收平台删除子设备通知
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	7.4.1 当前路径存储OTA请求下载包
	IOTA_GetOTAPackages_Ext	7.4.2 指定路径存储OTA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
设备时间同步	IOTA_GetNTPTime	9.1 设备上报时间同步请求
设备信息上报	IOTA_ReportDeviceInfo	10.1 设备信息上报
设备日志上报	IOTA_ReportDeviceLog	11.2 设备上报日志内容到平台
远程SSH登录	IOTA_RptDeviceConfigRst	12.1 上报远程SSH配置结果
端侧规则引擎	IOTA_EnableDeviceRuleStorage	13.1 端侧规则本地存储
M2M通信	IOTA_M2MSendMsg	14.1 M2M通信发送消息
端云安全组件	IOTA_GetLatestSoftBusInfo	15.1 获取云平台配置的最新软总线信息

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	描述
IOTA_SUCCESS	0	执行成功。
IOTA_FAILURE	-1	执行错误。

返回值名称	值	描述
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	HW_CHAR*	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_INT	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：SDK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	char*	打印的日志内容 ，里面可以包含要输出的变量。
args	可选	char*/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

- 用vprintf函数打印的日志，显示在控制台中。
- 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。

  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

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。

3.5.1 设置协议层回调函数

接口描述

IOTA_SetProtocolCallback函数主要用于设置协议层回调函数。

```
HW_VOID IOTA_SetProtocolCallback(HW_INT iItem, PFN_PROTOCOL_CALLBACK_HANDLER
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： 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
pfnCallbackHandler	必选	PFN_PROTOCOL_CALLBACK_HANDLER	自定义函数名称

PFN_CALLBACK_HANDLER类型：

```
HW_VOID (*PFN_PROTOCOL_CALLBACK_HANDLER)(EN_IOTA_MQTT_PROTOCOL_RSP* message)
```

EN_IOTA_MQTT_PROTOCOL_RSP类型：

字段	必选/可选	类型	描述
mqtt_msg_info	必选	EN_IOTA_MQTT_MSG_INFO	mqtt协议层返回信息

字段	必选/可选	类型	描述
message	必选	HW_CHAR*	消息体

EN_IOTA_MQTT_MSG_INFO类型：

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。当前预留，默认为NULL。
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）

示例

```
// 开发者设置回调函数
//HandleConnectSuccess、HandleConnectFailure等函数可参考SDK中src/device_demo/device_demo.c中的实现
void SetMyCallbacks() {
    IOTA_SetProtocolCallback(EN_IOTA_CALLBACK_CONNECT_SUCCESS, HandleConnectSuccess);
    IOTA_SetProtocolCallback(EN_IOTA_CALLBACK_CONNECT_FAILURE, HandleConnectFailure);

    IOTA_SetProtocolCallback(EN_IOTA_CALLBACK_DISCONNECT_SUCCESS, HandleDisConnectSuccess);
    IOTA_SetProtocolCallback(EN_IOTA_CALLBACK_DISCONNECT_FAILURE, HandleDisConnectFailure);
    IOTA_SetProtocolCallback(EN_IOTA_CALLBACK_CONNECTION_LOST, HandleConnectionLost);

    IOTA_SetProtocolCallback(EN_IOTA_CALLBACK_SUBSCRIBE_SUCCESS, HandleSubscribesuccess);
    IOTA_SetProtocolCallback(EN_IOTA_CALLBACK_SUBSCRIBE_FAILURE, HandleSubscribeFailure);

    IOTA_SetProtocolCallback(EN_IOTA_CALLBACK_PUBLISH_SUCCESS, HandlePublishSuccess);
    IOTA_SetProtocolCallback(EN_IOTA_CALLBACK_PUBLISH_FAILURE, HandlePublishFailure);
}
```

3.5.2 设置业务层回调函数

接口描述

以下函数均为设置业务层回调函数。

函数	功能	入参描述
HW_VOID IOTA_SetMessageCallback(PFN_MESSAGE_CALLBACK_HANDLER pfnCallbackHandler)	设置消息回调	入参为自定义回调函数指针，请参考5.1

函数	功能	入参描述
HW_VOID IOTA_SetUserTopicMsgCallback(PFN_USER_TOPIC_MSG_CALLBACK_HANDLER pfnCallbackHandler)	设置自定义TOPIC消息回调	入参为自定义回调函数指针，请参考5.2
HW_VOID IOTA_SetCmdCallback(PFN_CMD_CALLBACK_HANDLER pfnCallbackHandler)	设置命令回调	入参为自定义回调函数指针，请参考5.3
HW_VOID IOTA_SetPropSetCallback(PFN_PROP_SET_CALLBACK_HANDLER pfnCallbackHandler)	设置平台属性设置回调	入参为自定义回调函数指针，请参考5.4
HW_VOID IOTA_SetPropGetCallback(PFN_PROP_GET_CALLBACK_HANDLER pfnCallbackHandler)	设置平台属性查询回调	入参为自定义回调函数指针，请参考5.5
HW_VOID IOTA_SetEventCallback(PFN_EVENT_CALLBACK_HANDLER pfnCallbackHandler)	设置事件回调（**子设备新增/删除、OTA升级命令均属于事件**）	入参为自定义回调函数指针，子设备相关处理可参考5.6、5.7，OTA相关可参考7.1、7.3。
HW_VOID IOTA_SetShadowGetCallback(PFN_SHADOW_GET_CALLBACK_HANDLER pfnCallbackHandler)	设置设备影子回调	入参为自定义回调函数指针，请参考8.2
HW_API_FUNC HW_VOID IOTA_SetDeviceConfigCallback(PFN_DEVICE_CONFIG_CALLBACK_HANDLER pfnCallbackHandler)	设置设备远程配置结果回调	入参为自定义回调函数指针，请参考12.2 设置设备远程配置结果

函数	功能	入参描述
HW_API_FUNC HW_VOID IOTA_SetM2mCallback(PFN_M2M_CALLBACK_HANDLER pfnCallbackHandler)	设置M2M回调	入参为自定义回调函数指针，请参考
HW_API_FUNC HW_VOID IOTA_SetDeviceRuleSendMsgCallback(PFN_DEVICE_RULE_SEND_MSG_CALLBACK_HANDLER pfnCallbackHandler)	设置端侧规则引擎消息发送回调	入参为自定义回调函数指针，请参考陈星利

示例

```
// 开发者设置回调函数
//HandleMessageDown、HandleUserTopicMessageDown等函数是device_demo中演示的demo函数
void SetMyCallbacks() {
        //协议层回调函数
    IOTA_SetProtocolCallback(EN_IOTA_CALLBACK_CONNECT_SUCCESS, HandleConnectSuccess);
    IOTA_SetProtocolCallback(EN_IOTA_CALLBACK_CONNECT_FAILURE, HandleConnectFailure);
    IOTA_SetProtocolCallback(EN_IOTA_CALLBACK_DISCONNECT_SUCCESS, HandleDisConnectSuccess);
    IOTA_SetProtocolCallback(EN_IOTA_CALLBACK_DISCONNECT_FAILURE, HandleDisConnectFailure);
    IOTA_SetProtocolCallback(EN_IOTA_CALLBACK_CONNECTION_LOST, HandleConnectionLost);
    IOTA_SetProtocolCallback(EN_IOTA_CALLBACK_SUBSCRIBE_SUCCESS, HandleSubscribesuccess);
    IOTA_SetProtocolCallback(EN_IOTA_CALLBACK_SUBSCRIBE_FAILURE, HandleSubscribeFailure);
    IOTA_SetProtocolCallback(EN_IOTA_CALLBACK_PUBLISH_SUCCESS, HandlePublishSuccess);
    IOTA_SetProtocolCallback(EN_IOTA_CALLBACK_PUBLISH_FAILURE, HandlePublishFailure);

        //业务层回调函数
    IOTA_SetMessageCallback(HandleMessageDown);
        IOTA_SetM2mCallback(HandleM2mMessageDown);
    IOTA_SetUserTopicMsgCallback(HandleUserTopicMessageDown);
    IOTA_SetCmdCallback(HandleCommandRequest);
    IOTA_SetPropSetCallback(HandlePropertiesSet);
    IOTA_SetPropGetCallback(HandlePropertiesGet);
    IOTA_SetEventCallback(HandleEventsDown);
    IOTA_SetShadowGetCallback(HandleDeviceShadowRsp);
        IOTA_SetDeviceRuleSendMsgCallback(HandleDeviceRuleSendMsg);
        IOTA_SetDeviceConfigCallback(HandleDeviceConfig);
}
```

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, HW_INT compressFlag, void *context)
```

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。
compressFlag	必选	HW_INT	是否压缩上报，0代表非压缩上报，1代表压缩上报。压缩上报会消耗一定的内存，如果没有特定流量要求的话，建议使用非压缩上报。
context	可选	void *	指向任何数据体的上下文指针，该指针会在成功或者失败的回调函数中返回。不使用该指针可以传递NULL。

接口返回值

参见**函数标准返回值**

示例

```
// 开发者调用该接口进行消息上报
void Test_messageReport()
{
    int messageId = IOTA_MessageReport(NULL, "data123", "123", "hello", NULL, 0, NULL);
    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,
HW_INT compressFlag, void *context)
```

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

字段	必选/可选	类型	描述
compressFlag	必选	HW_INT	是否压缩上报，0代表非压缩上报，1代表压缩上报。压缩上报会消耗一定的内存，如果没有特定流量要求的话，建议使用非压缩上报。
context	可选	void *	指向任何数据体的上下文指针，该指针会在成功或者失败的回调函数中返回。不使用该指针可以传递NULL。

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[1].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 *pcCommandResponse, void *context)
```

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。
context	可选	void*	指向任何数据体的上下文指针，该指针会在成功或者失败的回调函数中返回。不使用该指针可以传递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,
NULL);
    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,
void *context)
```

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	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	HW_CHAR*	属性设置的响应描述，可以设置为NULL。
context	可选	void*	指向任何数据体的上下文指针，该指针会在成功或者失败的回调函数中返回。不使用该指针可以传递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", NULL);
    int messageId = IOTA_PropertiesSetResponse(requestId, 0, NULL, 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_INT IOTA_PropertiesGetResponse(HW_CHAR *requestId, ST_IOTA_SERVICE_DATA_INFO serviceProp[],
HW_INT serviceNum, void *context);
```

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
serviceNum	必选	HW_INT	要上报的服务个数
context	可选	void *	指向任何数据体的上下文指针，该指针会在成功或者失败的回调函数中返回。不使用该指针可以传递NULL。

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 = 1;
    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;

    int messageId = IOTA_PropertiesGetResponse(requestId, serviceProp, serviceNum, NULL);
    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,
void *context)
```

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
context	可选	void *	指向任何数据体的上下文指针，该指针会在成功或者失败的回调函数中返回。不使用该指针可以传递NULL。

ST_IOTA_SERVICE_DATA_INFO 类型：

字段	必选/可选	类型	描述
event_time	可选	HW_CHAR *	事件时间

字段	必选/可选	类型	描述
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	HW_CHAR *	子设备ID
status	必选	HW_CHAR *	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, NULL);
    if (messageId != 0) {
        PrintfLog(EN_LOG_LEVEL_ERROR, "device_demo: Test_UpdateSubDeviceStatus() failed, messageId %d
\n", messageId);
    }
}
```

4.7 直连设备上报新增子设备请求

接口功能

网关新增子设备。

接口描述

```
HW_INT IOTA_AddSubDevice(ST_IOTA_SUB_DEVICE_INFO *subDevicesInfo, HW_INT deviceNum, void
*context)
```


参数说明

字段	必选/可选	类型	描述
subDevicesInfo	必选	ST_IOTA_SUB_DEVICE_INFO *	子设备信息结构体
deviceNum	必选	HW_INT	新增的子设备个数，单次最大为50。有多个子设备时，建议分多次添加。
context	可选	void *	指向任何数据体的上下文指针，该指针会在成功或者失败的回调函数中返回。不使用该指针可以传递NULL。

ST_IOTA_SUB_DEVICE_INFO 类型：

字段	必选/可选	类型	描述
event_time	可选	HW_CHAR *	事件时间。不携带则平台采用平台的UTC时间。默认传NULL。
event_id	可选	HW_CHAR *	事件ID，用来标识该请求。不携带着平台默认生成。默认传NULL
deviceInfo	必选	ST_IOTA_DEVICE_INFO	添加的子设备信息

ST_IOTA_DEVICE_INFO类型：

字段名	必选/可选	类型	参数描述
parent_device_id	可选	HW_CHAR *	父节点设备ID
node_id	必选	HW_CHAR *	设备标识。
device_id	可选	HW_CHAR *	设备ID
name	可选	HW_CHAR *	设备名称
description	可选	HW_CHAR *	设备描述
product_id	必选	HW_CHAR *	产品ID
extension_info	可选	HW_CHAR *	设备扩展信息。用户可以自定义任何想要的扩展信息。字段值大小上限为1K。

接口返回值

参见**函数标准返回值**

平台响应参数请参考**5.8 接收网关新增子设备响应**

示例

```
// 开发者调用直连设备新增子设备接口
void Test_GtwAddSubDevice() {

    ST_IOTA_SUB_DEVICE_INFO subDeviceInfos;
    int deviceNum = 2;

        subDeviceInfos.deviceInfo[0].description = "description";
    subDeviceInfos.deviceInfo[0].device_id = "device_id123";
    subDeviceInfos.deviceInfo[0].extension_info = NULL;
    subDeviceInfos.deviceInfo[0].name = "sub_device111";
    subDeviceInfos.deviceInfo[0].node_id = "node_id123";
    subDeviceInfos.deviceInfo[0].parent_device_id = NULL;
    subDeviceInfos.deviceInfo[0].product_id = "your_product_id";

        subDeviceInfos.deviceInfo[1].description = "description";
    subDeviceInfos.deviceInfo[1].device_id = "device_id1234";
    subDeviceInfos.deviceInfo[1].extension_info = NULL;
    subDeviceInfos.deviceInfo[1].name = "sub_device222";
    subDeviceInfos.deviceInfo[1].node_id = "node_id123";
    subDeviceInfos.deviceInfo[1].parent_device_id = NULL;
    subDeviceInfos.deviceInfo[0].product_id = "your_product_id";
    subDeviceInfos.deviceInfo[1].product_id = "5f58768785edc002bc69cbf2";

    subDeviceInfos.event_id = "123123";
    subDeviceInfos.event_time = NULL;

    int messageId = IOTA_AddSubDevice(&subDeviceInfos, deviceNum, NULL);
    if (messageId != 0) {
        PrintfLog(EN_LOG_LEVEL_ERROR, "device_demo: Test_GtwAddSubDevice() failed, messageId %d\n",
messageId);
    }
}
```

4.8 直连设备上报删除子设备请求

接口功能

网关删除子设备。

接口描述

```
HW_INT IOTA_DelSubDevice(ST_IOTA_DEL_SUB_DEVICE *delSubDevices, HW_INT deviceNum, void *context)
```

参数说明

字段	必选/可选	类型	描述
delSubDevices	必选	ST_IOTA_DEL_SUB_DEVICE*	子设备信息结构体

字段	必选/可选	类型	描述
deviceNum	必选	HW_INT	删除的子设备个数，单次最大为50。有多个子设备时，建议分多次删除。
context	可选	void *	指向任何数据体的上下文指针，该指针会在成功或者失败的回调函数中返回。不使用该指针可以传递NULL。

ST_IOTA_DEL_SUB_DEVICE类型：

字段	必选/可选	类型	描述
event_time	可选	HW_CHAR *	事件时间。不携带则平台采用平台的UTC时间。默认传NULL。
event_id	可选	HW_CHAR *	事件ID，用来标识该请求。不携带着平台默认生成。默认传NULL
delSubDevice	必选	HW_CHAR **	删除的子设备ID列表

接口返回值

参见**函数标准返回值**

平台响应参数请参考**5.9 接收网关删除子设备响应**

示例

```
// 开发者调用直连设备新增子设备接口
void Test_GtwDelSubDevice () {
    ST_IOTA_DEL_SUB_DEVICE delSubDevices;
    int deviceNum = 3;

    delSubDevices.event_id = NULL;
    delSubDevices.event_time = NULL;
    delSubDevices.delSubDevice[0] = "device_id123";
    delSubDevices.delSubDevice[1] = "device_id1234";
    delSubDevices.delSubDevice[2] = "device_id12345";

    int messageId = IOTA_DelSubDevice(&delSubDevices, deviceNum, NULL);
    if (messageId != 0) {
        PrintfLog(EN_LOG_LEVEL_ERROR, "device_demo: Test_GtwDelSubDevice() failed, messageId %d\n",
messageId);
    }
}
```


5 直连设备接收数据

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等。

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请参考**3.5.2**。本章节主要讲命令**消息体**（回调函数的入参），及回调函数的实现。

说明

- 下行函数的结构体由SDK自行释放，如需保存结构体中的内容，需在业务处理时进行拷贝。
- 消息体中的可选参数均由第三方决定是否下发。

5.1 接收消息下发

接口描述

设备收到应用下发的透传消息，该消息平台不解析。

消息体

EN_IOTA_MESSAGE结构体：

字段名	必选/可选	类型	参数描述
mqtt_msg_info	必选	EN_IOTA_MQTT_MSG_INFO*	mqtt协议层信息
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。

EN_IOTA_MQTT_MSG_INFO类型：

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。当前预留，默认为NULL。
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）

示例

```
//开发者实现消息下发处理
void HandleMessageDown (EN_IOTA_MESSAGE *rsp) {
    if (rsp == NULL) {
        return;
    }
    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleMessageDown(), content %s\n", rsp->content);
    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleMessageDown(), id %s\n", rsp->id);
    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleMessageDown(), name %s\n", rsp->name);
    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleMessageDown(), object_device_id %s\n", rsp->object_device_id);
}

//设置回调函数
IOTA_SetMessageCallback(HandleMessageDown);
```

5.2 topic 处理

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

接口描述

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

EN_IOTA_USER_TOPIC_MESSAGE结构体：

字段名	必选/可选	类型	参数描述
mqtt_msg_info	必选	EN_IOTA_MQTT_MSG_INFO*	mqtt协议层信息
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。
topic_para	必选	String	自定义topic参数，例如“devMsg”

EN_IOTA_MQTT_MSG_INFO类型：

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。当前预留，默认为NULL。
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）

示例

```
//开发者实现自定义topic消息下发处理
void HandleUserTopicMessageDown(EN_IOTA_USER_TOPIC_MESSAGE *rsp) {
    if (rsp == NULL) {
        return;
    }
    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleMessageDown(), topic_para %s\n", rsp-
>topic_para);
    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleMessageDown(), content %s\n", rsp->content);
    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleMessageDown(), id %s\n", rsp->id);
    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleMessageDown(), name %s\n", rsp->name);
    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleMessageDown(), object_device_id %s\n", rsp-
>object_device_id);
}

//设置回调函数
IOTA_SetUserTopicMsgCallback(HandleUserTopicMessageDown);
```

5.2.3 订阅 bootstrap topic

接口功能

用于订阅bootstrap topic。

接口描述

```
HW_API_FUNC HW_INT IOTA_SubscribeBoostrap()
```

示例

```
// 开发者调用该接口进行bootstrap topic订阅
IOTA_SubscribeBoostrap();
```

5.3 接收命令下发

接口描述

用于平台向设备下发设备控制命令。

消息体

EN_IOTA_COMMAND结构体：

字段名	必选/可选	类型	参数描述
mqtt_msg_info	必选	EN_IOTA_MQTT_MSG_INFO*	mqtt协议层信息
object_device_id	可选	HW_CHAR*	命令对应的目标设备ID
service_id	可选	HW_CHAR*	设备的服务ID。
command_name	可选	HW_CHAR*	设备命令名称，在设备关联的产品模型中定义。
paras	必选	HW_CHAR*	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。
request_id	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。

EN_IOTA_MQTT_MSG_INFO类型：

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。当前预留，默认为NULL。
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）

示例

```
static void Test_CommandResponse(char *requestId)
{
    char *pcCommandRespense = "{\"SupWh\": \"aaa\"}"; // in service accumulator

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

    int messageId = IOTA_CommandResponse(requestId, result_code, response_name, pcCommandRespense,
NULL);

    if (messageId != 0) {
        PrintfLog(EN_LOG_LEVEL_ERROR, "device_demo: Test_CommandResponse() failed, messageId %d\n",
messageId);
    }
}

//开发者实现命令下发处理
void HandleCommandRequest(EN_IOTA_COMMAND *command) {

    if (command == NULL) {
        return;
```



```
    }

    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleCommandRequest(), messageId %d\n", command->mqtt_msg_info->messageId);

    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleCommandRequest(), object_device_id %s\n", command->object_device_id);
    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleCommandRequest(), service_id %s\n", command->service_id);
    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleCommandRequest(), command_name %s\n", command->command_name);
    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleCommandRequest(), paras %s\n", command->paras);
    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleCommandRequest(), request_id %s\n", command->request_id);

    Test_CommandResponse(command->request_id); //response command
}

//设置回调函数
IOTA_SetCmdCallback(HandleCommandRequest);
```

说明

Test_commandResponse(requestId)函数里实现了命令响应逻辑，详细请参考**4.3 直连设备上报命令执行结果**。

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

表 5-1 EN_IOTA_PROPERTY_SET 结构体

字段名	必选/可选	类型	参数描述
mqtt_msg_info	必选	EN_IOTA_MQTT_MSG_INFO*	mqtt协议层信息
object_device_id	可选	HW_CHAR*	属性设置对应的目标设备ID
request_id	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。
services	必选	EN_IOTA_SERVICE_PROPERTY*	设备服务数据列表。
services_count	必选	HW_INT	设备服务数据列表的个数

表 5-2 EN_IOTA_SERVICE_PROPERTY 类型

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

表 5-3 EN_IOTA_MQTT_MSG_INFO 类型

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。当前预留，默认为NULL。
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）

示例

```
static void Test_PropSetResponse(char *requestId)
{
    int messageId = IOTA_PropertiesSetResponse(requestId, 0, "success", NULL);
    if (messageId != 0) {
        PrintfLog(EN_LOG_LEVEL_ERROR, "device_demo: Test_PropSetResponse() failed, messageId %d\n",
messageId);
    }
}

//开发者实现设置设备属性命令处理
void HandlePropertiesSet (EN_IOTA_PROPERTY_SET *rsp) {
    if (rsp == NULL) {
        return;
    }
    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandlePropertiesSet(), messageId %d \n", rsp-
>mqtt_msg_info->messageId);
    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandlePropertiesSet(), request_id %s \n", rsp-
>request_id);
    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandlePropertiesSet(), object_device_id %s \n", rsp-
>object_device_id);

    int i = 0;
    while (rsp->services_count > 0) {
        PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandlePropertiesSet(), service_id %s \n", rsp-
>services[i].service_id);
        PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandlePropertiesSet(), properties %s \n", rsp-
>services[i].properties);
        rsp->services_count--;
        i++;
    }

    Test_PropSetResponse(rsp->request_id); //response
}
```



```
//设置回调函数
IOTA_SetPropSetCallback(HandlePropertiesSet);
```

说明

Test_propSetResponse(requestId)函数里实现了命令响应逻辑，详细请参考**4.4 直连设备上报设备属性设置结果**。

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

表 5-4 EN_IOTA_PROPERTY_GET 结构体

字段名	必选/可选	类型	参数描述
mqtt_msg_info	必选	EN_IOTA_MQTT_MSG_INFO*	mqtt协议层信息
object_device_id	可选	HW_CHAR*	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	HW_CHAR*	设备的服务ID，不设置默认查询全部service。
request_id	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。

表 5-5 EN_IOTA_MQTT_MSG_INFO 类型

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。当前预留，默认为NULL。
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）

示例

```
static void Test_PropGetResponse(char *requestId)
{
    const int serviceNum = 2;
```

```
    ST_IOTA_SERVICE_DATA_INFO serviceProp[serviceNum];

    char *property = "{\"Load\":\"5\",\"ImbA_strVal\":\"6\"}";

    serviceProp[0].event_time = GetEventTimesStamp();
    serviceProp[0].service_id = "parameter";
    serviceProp[0].properties = property;

    char *property2 = "{\"PhV_phsA\":\"2\",\"PhV_phsB\":\"4\"}";

    serviceProp[1].event_time = GetEventTimesStamp();
    serviceProp[1].service_id = "analog";
    serviceProp[1].properties = property2;

    int messageId = IOTA_PropertiesGetResponse(requestId, serviceProp, serviceNum, NULL);
    if (messageId != 0) {
        PrintfLog(EN_LOG_LEVEL_ERROR, "device_demo: Test_PropGetResponse() failed, messageId %d\n",
messageId);
    }

    MemFree(&serviceProp[0].event_time);
    MemFree(&serviceProp[1].event_time);
}

//开发者实现查询设备属性命令处理
void HandlePropertiesGet (EN_IOTA_PROPERTY_GET *rsp) {
    if (rsp == NULL) {
        return;
    }
    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandlePropertiesSet(), messageId %d \n", rsp-
>mqtt_msg_info->messageId);
    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandlePropertiesSet(), request_id %s \n", rsp-
>request_id);
    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandlePropertiesSet(), object_device_id %s \n", rsp-
>object_device_id);

    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandlePropertiesSet(), service_id %s \n", rsp->service_id);

    Test_PropGetResponse(rsp->request_id); //response
}

//设置回调函数
IOTA_SetPropGetCallback(HandlePropertiesGet);
```

说明

Test_propGetResponse(requestId)函数里实现了命令响应逻辑，详细请参考**4.5 直连设备上报设备属性查询结果**。

5.6 接收平台新增子设备通知

命令描述

平台将该新增的子设备列表信息通知给设备。

消息体

表 5-6 EN_IOTA_EVENT 结构体

字段名	必选/可选	类型	参数描述
mqtt_msg_info	必选	EN_IOTA_MQTT_MSG_INFO*	mqtt协议层信息
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网关设备
services	可选	EN_IOTA_SERVICE_EVENT*	事件服务列表
services_count	可选	int	事件服务列表的个数

表 5-7 EN_IOTA_MQTT_MSG_INFO 类型

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。当前预留，默认为NULL。
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）

表 5-8 EN_IOTA_SERVICE_EVENT 类型

字段名	必选/可选	类型	参数描述
service_id	必选	int	EN_IOTA_EVENT_SUB_DEVICE_MANAGER（枚举值为0）
event_type	必选	int	EN_IOTA_EVENT_ADD_SUB_DEVICE_NOTIFY（枚举值为0）
event_time	可选	HW_CHAR*	事件时间
paras	必选	EN_IOTA_DEVICE_PARAS*	子设备事件参数
ota_paras	可选	EN_IOTA_OTA_PARAS*	ota事件参数，此时为NULL
ntp_paras	可选	EN_IOTA_NTP_PARAS*	NTP事件参数，此时为NULL

字段名	必选/可选	类型	参数描述
gtw_add_device_paras	可选	EN_IOTA_GTW_ADD_DEVICE_PARAS	网关新增子设备事件，此时为NULL
gtw_del_device_paras	可选	EN_IOTA_GTW_DEL_DEVICE_PARAS	网关删除子设备事件，此时为NULL

表 5-9 EN_IOTA_DEVICE_PARAS 结构体

字段名	必选/可选	类型	参数描述
devices	必选	EN_IOTA_DEVICE_INFO*	设备列表
devices_count	必选	int	设备列表个数
version	必选	long long	子设备信息版本

表 5-10 EN_IOTA_DEVICE_INFO 定义表

字段名	必选/可选	类型	参数描述
parent_device_id	必选	HW_CHAR*	父节点设备ID
node_id	必选	HW_CHAR*	设备标识。
device_id	必选	HW_CHAR*	设备ID
name	可选	HW_CHAR*	设备名称
description	可选	HW_CHAR*	设备描述
manufacturer_id	可选	HW_CHAR*	厂商ID
model	可选	HW_CHAR*	设备型号
product_id	可选	HW_CHAR*	产品ID
fw_version	可选	HW_CHAR*	固件版本
sw_version	可选	HW_CHAR*	软件版本
status	可选	HW_CHAR*	设备在线状态 ONLINE：设备在线 OFFLINE：设备离线

字段名	必选/可选	类型	参数描述
extension_info	可选	HW_CHAR*	设备扩展信息。用户可以自定义任何想要的扩展信息

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(EN_IOTA_EVENT *message){

    if (message == NULL) {
        return;
    }

    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleEventsDown(), messageId %d\n", message->mqtt_msg_info->messageId);
    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleEventsDown(), services_count %d\n", message->services_count);
    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleEventsDown(), object_device_id %s\n", message->object_device_id);
    int i = 0;
    while (message->services_count > 0) {
        printf("servie_id: %d \n", message->services[i].servie_id);
        printf("event_time: %s \n", message->services[i].event_time);
        printf("event_type: %d \n", message->services[i].event_type);

        //sub device manager
        if (message->services[i].servie_id == EN_IOTA_EVENT_SUB_DEVICE_MANAGER) {

            //if it is the platform inform the gateway to add or delete the sub device
            if(message->services[i].event_type == EN_IOTA_EVENT_ADD_SUB_DEVICE_NOTIFY || message->services[i].event_type == EN_IOTA_EVENT_DELETE_SUB_DEVICE_NOTIFY) {
                printf("version: %lld \n", message->services[i].paras->version);

                int j = 0;
                while(message->services[i].paras->devices_count > 0) {

                    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleEventsDown(), parent_device_id: %s \n", message->services[i].paras->devices[j].parent_device_id);
                    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleEventsDown(), device_id: %s \n", message->services[i].paras->devices[j].device_id);
                    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleEventsDown(), node_id: %s \n", message->services[i].paras->devices[j].node_id);

                    //add a sub device
                    if (message->services[i].event_type == EN_IOTA_EVENT_ADD_SUB_DEVICE_NOTIFY) {
                        PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleEventsDown(), name: %s \n", message->services[i].paras->devices[j].name);
                        PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleEventsDown(), manufacturer_id: %s \n", message->services[i].paras->devices[j].manufacturer_id);
                        PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleEventsDown(), product_id: %s \n", message->services[i].paras->devices[j].product_id);

                        Test_UpdateSubDeviceStatus(message->services[i].paras->devices[j].device_id); //report status of the sub device
                        Test_BatchPropertiesReport(message->services[i].paras->devices[j].device_id); //report data of the sub device
                    } else if (message->services[i].event_type == EN_IOTA_EVENT_DELETE_SUB_DEVICE_NOTIFY) {  //delete a sub device
                        PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleEventsDown(), the sub device is deleted: %s\n", message->services[i].paras->devices[j].device_id);
                    }
```

```
                j++;
                message->services[i].paras->devices_count--;
            }
        } else if (message->services[i].event_type == EN_IOTA_EVENT_ADD_SUB_DEVICE_RESPONSE) {
            int j = 0;
            while(message->services[i].gtw_add_device_paras->successful_devices_count > 0) {
                PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleEventsDown(), device_id: %s \n",
message->services[i].gtw_add_device_paras->successful_devices[j].device_id);
                PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleEventsDown(), name: %s \n", message-
>services[i].gtw_add_device_paras->successful_devices[j].name);
                j++;
                message->services[i].gtw_add_device_paras->successful_devices_count--;
            }
            j = 0;
            while(message->services[i].gtw_add_device_paras->failed_devices_count > 0) {
                PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleEventsDown(), error_code: %s \n",
message->services[i].gtw_add_device_paras->failed_devices[j].error_code);
                PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleEventsDown(), error_msg: %s \n",
message->services[i].gtw_add_device_paras->failed_devices[j].error_msg);
                PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleEventsDown(), node_id: %s \n",
message->services[i].gtw_add_device_paras->failed_devices[j].node_id);
                PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleEventsDown(), product_id: %s \n",
message->services[i].gtw_add_device_paras->failed_devices[j].product_id);

                j++;
                message->services[i].gtw_add_device_paras->failed_devices_count--;
            }
        } else if (message->services[i].event_type == EN_IOTA_EVENT_DEL_SUB_DEVICE_RESPONSE) {
            int j = 0;
            while(message->services[i].gtw_del_device_paras->successful_devices_count > 0) {
                PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleEventsDown(), device_id: %s \n",
message->services[i].gtw_del_device_paras->successful_devices[j]);
                j++;
                message->services[i].gtw_del_device_paras->successful_devices_count--;
            }
            j = 0;
            while(message->services[i].gtw_del_device_paras->failed_devices_count > 0) {
                PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleEventsDown(), error_code: %s \n",
message->services[i].gtw_del_device_paras->failed_devices[j].error_code);
                PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleEventsDown(), error_msg: %s \n",
message->services[i].gtw_del_device_paras->failed_devices[j].error_msg);
                PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleEventsDown(), device_id: %s \n",
message->services[i].gtw_del_device_paras->failed_devices[j].device_id);

                j++;
                message->services[i].gtw_del_device_paras->failed_devices_count--;
            }
        }

    } else if (message->services[i].servie_id == EN_IOTA_EVENT_OTA) {
        if (message->services[i].event_type == EN_IOTA_EVENT_VERSION_QUERY) {
            //report OTA version
            Test_ReportOTAVersion();

        }

        if (message->services[i].event_type == EN_IOTA_EVENT_FIRMWARE_UPGRADE || message-
>services[i].event_type == EN_IOTA_EVENT_SOFTWARE_UPGRADE) {

            //check md5
            char pkg_md5 = "yourMd5"; //the md5 value of your ota package
            if (strcmp(pkg_md5, message->services[i].ota_paras->sign)) {
                //report failed status
                Test_ReportUpgradeStatus(-1, message->services[i].ota_paras->version);
            }

            //start to receive packages and firmware_upgrade or software_upgrade
            if (IOTA_GetOTAPackages(message->services[i].ota_paras->url, message->services[i].ota_paras-
>access_token, 1000) == 0) {
```

```
                usleep(3000 * 1000);
                //report successful upgrade status
                Test_ReportUpgradeStatus(0, message->services[i].ota_paras->version);
            } else {
                //report failed status
                Test_ReportUpgradeStatus(-1, message->services[i].ota_paras->version);
            }
        }

    } else if (message->services[i].servie_id == EN_IOTA_EVENT_TIME_SYNC) {
        if(message->services[i].event_type == EN_IOTA_EVENT_GET_TIME_SYNC_RESPONSE) {
            PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleEventsDown(), device_real_time: %lld \n",
message->services[i].ntp_paras->device_real_time);
        }
    } else if (message->services[i].servie_id == EN_IOTA_EVENT_DEVICE_LOG) {
        if(message->services[i].event_type == EN_IOTA_EVENT_LOG_CONFIG) {
            PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleEventsDown(), log_switch: %s \n",
message->services[i].device_log_paras->log_switch);
            PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleEventsDown(), end_time: %s \n", message-
>services[i].device_log_paras->end_time);
        }
    }
    i++;
    message->services_count--;
  }
}
```

说明

- if (message->services[i].event_type == EN_IOTA_EVENT_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。
- if (message->services[i].event_type == EN_IOTA_EVENT_DELETE_SUB_DEVICE_NOTIFY)判断是删除子设备事件通知。
- if (message->services[i].event_type == EN_IOTA_EVENT_ADD_SUB_DEVICE_RESPONSE)判断是网关主动添加子设备响应
- if (message->services[i].event_type == EN_IOTA_EVENT_DEL_SUB_DEVICE_RESPONSE)判断是网关主动删除子设备响应
- Test_UpdateSubDeviceStatus函数实现了子设备状态上报功能，详细请参考**4.6 直连设备上报子设备状态**。
- Test_BatchPropertiesReport函数里实现了子设备数据上报功能，详细请参考**6.2 批量上报设备属性**。

5.7 接收平台删除子设备通知

命令描述

平台将该删除的子设备信息通知给设备。

消息体

表 5-11 EN_IOTA_EVENT 结构体

字段名	必选/可选	类型	参数描述
mqtt_msg_info	必选	EN_IOTA_MQTT_MSG_INFO*	mqtt协议层信息

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	EN_IOTA_SERVICE_EVENT*	事件服务列表
services_count	可选	int	事件服务列表的个数

表 5-12 EN_IOTA_MQTT_MSG_INFO 类型

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。当前预留，默认为NULL。
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）

表 5-13 EN_IOTA_SERVICE_EVENT 类型

字段名	必选/可选	类型	参数描述
service_id	必选	int	EN_IOTA_EVENT_SUB_DEVICE_MANAGER（枚举值为0）
event_type	必选	int	EN_IOTA_EVENT_DELETE_SUB_DEVICE_NOTIFY（枚举值为1）
event_time	可选	String	事件时间
paras	必选	EN_IOTA_DEVICE_PARAS*	子设备事件参数
ota_paras	可选	EN_IOTA_OTA_PARAS*	ota事件参数，此时为NULL
ntp_paras	可选	EN_IOTA_NTP_PARAS*	NTP事件参数，此时为NULL
gtw_add_device_paras	可选	EN_IOTA_GTW_ADD_DEVICE_PARAS	网关新增子设备事件，此时为NULL
gtw_del_device_paras	可选	EN_IOTA_GTW_DEL_DEVICE_PARAS	网关删除子设备事件，此时为NULL

表 5-14 EN_IOTA_DEVICE_PARAS 结构体

字段名	必选/可选	类型	参数描述
devices	必选	EN_IOTA_DEVICE_INFO*	设备列表
devices_count	必选	int	设备列表个数
version	必选	long long	子设备信息版本

表 5-15 EN_IOTA_DEVICE_PARAS 结构体

字段名	必选/可选	类型	参数描述
devices	必选	EN_IOTA_DEVICE_INFO*	设备列表
devices_count	必选	int	设备列表个数
version	必选	long long	子设备信息版本

表 5-16 EN_IOTA_DEVICE_INFO 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

示例

请参考**5.6 接收平台新增子设备通知**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。

5.8 接收网关新增子设备响应

命令描述

平台下发网关主动添加的子设备响应。

消息体

EN_IOTA_EVENT结构体：

字段名	必选/可选	类型	参数描述
mqtt_msg_info	必选	EN_IOTA_MQTT_MSG_INFO*	mqtt协议层信息
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	EN_IOTA_SERVICE_EVENT*	事件服务列表
services_count	可选	int	事件服务列表的个数

EN_IOTA_MQTT_MSG_INFO类型：

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。当前预留，默认为NULL。
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）

EN_IOTA_SERVICE_EVENT类型：

字段名	必选/可选	类型	参数描述
service_id	必选	int	EN_IOTA_EVENT_SUB_DEVICE_MANAGER（枚举值为0）
event_type	必选	int	EN_IOTA_EVENT_ADD_SUB_DEVICE_RESPONSE（枚举值为6）
event_time	可选	String	事件时间
paras	必选	EN_IOTA_DEVICE_PARAS*	子设备事件参数，此时为NULL
ota_paras	可选	EN_IOTA_OTA_PARAS*	ota事件参数，此时为NULL
ntp_paras	可选	EN_IOTA_NTP_PARAS*	NTP事件参数，此时为NULL

字段名	必选/可选	类型	参数描述
gtw_add_device_paras	必选	EN_IOTA_GTW_ADD_DEVICE_PARAS	网关新增子设备事件
gtw_del_device_paras	可选	EN_IOTA_GTW_DEL_DEVICE_PARAS	网关删除子设备事件，此时为NULL
device_log_paras	可选	EN_IOTA_DEVICE_LOG_PARAS*	设备日志事件参数，此时为NULL

EN_IOTA_GTW_ADD_DEVICE_PARAS结构体：

字段名	必选/可选	类型	参数描述
successful_devices	必选	EN_IOTA_DEVICE_INFO*	成功新增的子设备列表详情
successful_devices_count	必选	int	成功添加的子设备个数
failed_devices	必选	EN_IOTA_ADD_DEVICE_FAILED_REASON*	新增子设备失败的原因
failed_devices_count	必选	int	添加失败的子设备个数

EN_IOTA_DEVICE_INFO 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

字段名	必选/可选	类型	参数描述
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
extension_info	可选	Object	设备扩展信息。用户可以自定义任何想要的扩展信息

EN_IOTA_ADD_DEVICE_FAILED_REASON定义表

字段名	必选/可选	类型	参数描述
node_id	必选	String	对应请求中指定的设备的node_id
product_id	必选	String	对应请求中指定的设备的product_id
error_code	必选	String	新增失败错误原因码
error_msg	必选	String	新增失败原因描述

示例

请参考**5.6 接收平台新增子设备通知**示例。

说明

平台新增/删除子设备子设备通知Topic与网关新增/删除子设备响应的TOPIC一致，可通过消息体中的event_type字段来进行判断。

5.9 接收网关删除子设备响应

命令描述

平台下发网关主动删除的子设备响应。

消息体

EN_IOTA_EVENT结构体：

字段名	必选/可选	类型	参数描述
mqtt_msg_info	必选	EN_IOTA_MQTT_MSG_INFO*	mqtt协议层信息。

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	EN_IOTA_SERVICE_EVENT*	事件服务列表。
services_count	可选	int	事件服务列表的个数。

EN_IOTA_MQTT_MSG_INFO类型：

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。当前预留，默认为NULL。
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）。

EN_IOTA_SERVICE_EVENT类型：

字段名	必选/可选	类型	参数描述
service_id	必选	int	EN_IOTA_EVENT_SUB_DEVICE_MANAGER（枚举值为0）。
event_type	必选	int	EN_IOTA_EVENT_DEL_SUB_DEVICE_RESPONSE（枚举值为7）。
event_time	可选	String	事件时间。
paras	必选	EN_IOTA_DEVICE_PARAS*	子设备事件参数，此时为NULL。
ota_paras	可选	EN_IOTA_OTA_PARAS*	ota事件参数，此时为NULL。
ntp_paras	可选	EN_IOTA_NTP_PARAS*	NTP事件参数，此时为NULL。
gtw_add_device_paras	必选	EN_IOTA_GTW_ADD_DEVICE_PARAS	网关新增子设备事件，此时为NULL。

字段名	必选/可选	类型	参数描述
gtw_del_device_paras	可选	EN_IOTA_GTW_DEL_DEVICE_PARAS	网关删除子设备事件。
device_log_paras	可选	EN_IOTA_DEVICE_LOG_PARAS*	设备日志事件参数，此时为NULL。

EN_IOTA_GTW_ADD_DEVICE_PARAS结构体：

字段名	必选/可选	类型	参数描述
successful_devices	必选	EN_IOTA_DEVICE_INFO *	成功删除的子设备列表详情。
successful_devices_count	必选	int	成功删除的子设备个数。
failed_devices	必选	EN_IOTA_DEL_DEVICE_FAILED_REASON*	删除子设备失败的原因。
failed_devices_count	必选	int	删除失败的子设备个数。

EN_IOTA_DEL_DEVICE_FAILED_REASON定义表

字段名	必选/可选	类型	参数描述
device_id	必选	String	对应请求中指定的设备的device_id。
error_code	必选	String	删除失败错误原因码。
error_msg	必选	String	删除失败原因描述。

EN_IOTA_DEVICE_INFO 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。

示例

请参考**5.6 接收平台新增子设备通知**示例。

说明

平台新增/删除子设备子设备通知Topic与网关新增/删除子设备响应的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 直连设备上报自定义topic消息**，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[], HW_INT compressFlag, void *context)
```

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

字段	必选/可选	类型	描述
compressFlag	必选	HW_INT	是否压缩上报，0代表非压缩上报，1代表压缩上报。压缩上报会消耗一定的内存，如果没有特定流量要求的话，建议使用非压缩上报。
context	可选	void *	指向任何数据体的上下文指针，该指针会在成功或者失败的回调函数中返回。不使用该指针可以传递NULL。

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, 1, NULL);
    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

EN_IOTA_EVENT结构体：

字段名	必选/可选	类型	参数描述
mqtt_msg_info	必选	EN_IOTA_MQTT_MSG_INFO*	mqtt协议层信息
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网关设备
services	可选	EN_IOTA_SERVICE_EVENT*	事件服务列表
services_count	可选	int	事件服务列表的个数

EN_IOTA_MQTT_MSG_INFO类型：

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。当前预留，默认为NULL。
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）

EN_IOTA_SERVICE_EVENT类型：

字段名	必选/可选	类型	参数描述
service_id	必选	int	EN_IOTA_EVENT_OTA（枚举值为1）
event_type	必选	int	EN_IOTA_EVENT_VERSION_QUERY（枚举值为2）
event_time	可选	String	事件时间
paras	必选	EN_IOTA_DEVICE_PARAS*	子设备事件参数，此时为NULL
ota_paras	可选	EN_IOTA_OTA_PARAS*	ota事件参数
ntp_paras	可选	EN_IOTA_NTP_PARAS*	NTP事件参数，此时为NULL
gtw_add_device_paras	可选	EN_IOTA_GTW_ADD_DEVICE_PARAS	网关新增子设备事件，此时为NULL
gtw_del_device_paras	可选	EN_IOTA_GTW_DEL_DEVICE_PARAS	网关删除子设备事件，此时为NULL
device_log_paras	可选	EN_IOTA_DEVICE_LOG_PARAS*	设备日志事件参数，此时为NULL

ota_paras参数列表

字段名	必选/可选	类型	参数描述
-	-	-	-

示例

请参考**5.6 接收平台新增子设备通知**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, void *context)
```

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
context	可选	void *	指向任何数据体的上下文指针，该指针会在成功或者失败的回调函数中返回。不使用该指针可以传递NULL。

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

EN_IOTA_EVENT结构体：

字段名	必选/可选	类型	参数描述
mqtt_msg_info	必选	EN_IOTA_MQTT_MSG_INFO*	mqtt协议层信息
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	EN_IOTA_SERVICE_EVENT*	事件服务列表
services_count	可选	int	事件服务列表的个数

EN_IOTA_MQTT_MSG_INFO类型：

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。当前预留，默认为NULL。
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）

EN_IOTA_SERVICE_EVENT类型：

字段名	必选/可选	类型	参数描述
service_id	必选	int	EN_IOTA_EVENT_OTA（枚举值为1）
event_type	必选	int	• EN_IOTA_EVENT_FIRMWARE_UPGRADE（枚举值为3，固件升级） • EN_IOTA_EVENT_SOFTWARE_UPGRADE（枚举值为4，软件升级）
event_time	可选	String	事件时间
paras	必选	EN_IOTA_DEVICE_PARAS*	子设备事件参数，此时为NULL
ota_paras	可选	EN_IOTA_OTA_PARAS*	ota事件参数
ntp_paras	可选	EN_IOTA_NTP_PARAS*	NTP事件参数，此时为NULL
gtw_add_device_paras	可选	EN_IOTA_GTW_ADD_DEVICE_PARAS	网关新增子设备事件，此时为NULL
gtw_del_device_paras	可选	EN_IOTA_GTW_DEL_DEVICE_PARAS	网关删除子设备事件，此时为NULL
device_log_paras	可选	EN_IOTA_DEVICE_LOG_PARAS*	设备日志事件参数，此时为NULL

ota_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

示例

请参考**5.6 接收平台新增子设备通知**示例。

7.4 设备请求下载包

7.4.1 当前路径存储 OTA 请求下载包

接口功能

网关可以通过该接口请求下载软/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 接收平台新增子设备通知**示例。

7.4.2 指定路径存储 OTA 请求下载包

接口功能

网关可以通过该接口请求下载软/固件包，下载完毕后会在指定目录下生成一个软/固件包，同时可以获取该文件名称。

接口描述

```
HW_API_FUNC HW_INT IOTA_GetOTAPackages_Ext(HW_CHAR *url, HW_CHAR *token, HW_INT timeout,
const HW_CHAR * otaFilePath, HW_CHAR *otaFilenameOut)
```


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。
otaFilePath	必选	HW_CHAR*	软固件包存储路径
otaFilenameOut	必选	HW_CHAR*	软固件包文件名称

接口返回值

参见**函数标准返回值**

示例

```
static void HandleEventOta(EN_IOTA_EVENT *message, int i)
{
    if (message->services[i].event_type == EN_IOTA_EVENT_VERSION_QUERY) {
        // report OTA version
        Test_ReportOTAVersion();
    }

    if (message->services[i].event_type == EN_IOTA_EVENT_FIRMWARE_UPGRADE ||
        message->services[i].event_type == EN_IOTA_EVENT_SOFTWARE_UPGRADE ||
        message->services[i].event_type == EN_IOTA_EVENT_FIRMWARE_UPGRADE_V2 ||
        message->services[i].event_type == EN_IOTA_EVENT_SOFTWARE_UPGRADE_V2) {
        // check sha256
        const char *pkg_sha256 = "your package sha256"; // the sha256 value of your ota package
        if (message->services[i].ota_paras->sign != NULL) {
            if (strcmp(pkg_sha256, message->services[i].ota_paras->sign)) { //V1 only
                // report failed status
                Test_ReportUpgradeStatus(-1, message->services[i].ota_paras->version);
            }
        }

        char filename[PKGNAME_MAX + 1];
        // start to receive packages and firmware_upgrade or software_upgrade
        if (IOTA_GetOTAPackages_Ext(message->services[i].ota_paras->url, message->services[i].ota_paras-
>access_token, 1000, ".", filename) == 0) {
            usleep(3000 * 1000);
            PrintfLog(EN_LOG_LEVEL_DEBUG, "the filename is %s\n", filename);
            // report successful upgrade status
            Test_ReportUpgradeStatus(0, message->services[i].ota_paras->version);
        } else {
            // report failed status
            Test_ReportUpgradeStatus(-1, message->services[i].ota_paras->version);
        }
    }
}
static void HandleEventsDown(EN_IOTA_EVENT *message)
{
    if (message == NULL) {
        return;
    }

    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleEventsDown(), messageId %d\n", message-
```

```
>mqtt_msg_info->messageId);
    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleEventsDown(), services_count %d\n", message-
>services_count);
    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleEventsDown(), object_device_id %s\n", message-
>object_device_id);
    int i = 0;
    while (message->services_count > 0) {
        // sub device manager
        if (message->services[i].servie_id == EN_IOTA_EVENT_SUB_DEVICE_MANAGER) {
            HandleSubDeviceManager(message, i);
        } else if (message->services[i].servie_id == EN_IOTA_EVENT_OTA) {
            HandleEventOta(message, i);
        } else if (message->services[i].servie_id == EN_IOTA_EVENT_TIME_SYNC) {
            HandleTimeSync(message, i);
        } else if (message->services[i].servie_id == EN_IOTA_EVENT_DEVICE_LOG) {
            HandleDeviceLog(message, i);
        } else if (message->services[i].servie_id == EN_IOTA_EVENT_TUNNEL_MANAGER) {
#ifdef SSH_SWITCH
            HandleTunnelMgr(message, i);
#endif
        } else if (message->services[i].servie_id == EN_IOTA_EVENT_SOFT_BUS) {
            PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleEventsDown(), event_id: %s \n", message-
>services[i].event_id);
            PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleEventsDown(), soft_bus_info: %s \n",
message->services[i].soft_bus_paras->bus_infos);
        }
        i++;
        message->services_count--;
    }
}
```

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, void *context)
```

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
context	可选	void *	指向任何数据体的上下文指针，该指针会在成功或者失败的回调函数中返回。不使用该指针可以传递NULL。

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, void *context)
```

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
context	可选	void *	指向任何数据体的上下文指针，该指针会在成功或者失败的回调函数中返回。不使用该指针可以传递NULL。

接口返回值

参见**函数标准返回值**

示例

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

8.2 设备接收平台返回的设备影子数据

接口描述

用于接收平台返回的设备影子数据。

消息体

EN_IOTA_EVENT结构体：

字段名	必选/可选	类型	参数描述
mqtt_msg_info	必选	EN_IOTA_MQTT_MSG_INFO*	mqtt协议层信息
object_device_id	可选	HW_CHAR*	设备影子的目标设备ID。
request_id	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。
shadow	可选	EN_IOTA_SHADOW_DATA*	服务影子数据列表
shadow_data_count	可选	int	服务影子列表的个数（默认为0）

EN_IOTA_MQTT_MSG_INFO类型：

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。当前预留，默认为NULL。
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）

EN_IOTA_SHADOW_DATA类型：

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_properties	可选	String	设备影子desired区的属性列表，具体字段在设备关联的产品模型里定义。
desired_event_time	可选	String	desired区设备属性数据的UTC时间（格式：yyyyMMddTHHmmssZ），如：20161219T114920Z。
reported_properties	可选	String	设备影子reported区的属性列表，具体字段在设备关联的产品模型里定义。
reported_event_time	可选	String	reported区设备属性数据的UTC时间（格式：yyyyMMddTHHmmssZ），如：20161219T114920Z。
version	可选	Integer	设备影子版本信息

示例

```
//开发者实现影子数据处理
void HandleDeviceShadowRsp(EN_IOTA_DEVICE_SHADOW *rsp) {
    if (rsp == NULL) {
        return;
    }
    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleDeviceShadowRsp(), messageId %d \n", rsp->mqtt_msg_info->messageId);
    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleDeviceShadowRsp(), request_id %s \n", rsp->request_id);
    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleDeviceShadowRsp(), object_device_id %s \n", rsp->object_device_id);

    int i = 0;
    while (rsp->shadow_data_count > 0) {
        PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleDeviceShadowRsp(), service_id %s \n", rsp->shadow[i].service_id);
        PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleDeviceShadowRsp(), desired properties %s \n", rsp->shadow[i].desired_properties);
        PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleDeviceShadowRsp(), reported properties %s \n", rsp->shadow[i].reported_properties);
        PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleDeviceShadowRsp(), version    %d \n", rsp->shadow[i].version);
        rsp->shadow_data_count--;
        i++;
    }
}

//设置回调函数
IOTA_SetShadowGetCallback(HandleDeviceShadowRsp);
```


9 设备时间同步

9.1 设备上报时间同步请求

接口功能

设备向平台发起时间同步请求。

接口描述

```
HW_API_FUNC HW_INT IOTA_GetNTPTime(void *context)
```

参数说明

字段	必选/可选	类型	描述
context	可选	void *	指向任何数据体的上下文指针，该指针会在成功或者失败的回调函数中返回。不使用该指针可以传递NULL。

接口返回值

参见**函数标准返回值**

示例

```
// 开发者调用该接口进行时间同步请求
//NTP
IOTA_GetNTPTime(NULL);
```


9.2 设备接收时间同步响应

命令描述

设备接收平台下发的时间同步响应，获取设备准确时间。

消息体

EN_IOTA_EVENT结构体：

字段名	必选/可选	类型	参数描述
mqtt_msg_info	必选	EN_IOTA_MQTT_MSG_INFO*	mqtt协议层信息
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网关设备
services	可选	EN_IOTA_SERVICE_EVENT*	事件服务列表
services_count	可选	int	事件服务列表的个数

EN_IOTA_MQTT_MSG_INFO类型：

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。当前预留，默认为NULL。
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）

EN_IOTA_SERVICE_EVENT类型：

字段名	必选/可选	类型	参数描述
service_id	必选	int	EN_IOTA_EVENT_TIME_SYNC（枚举值为2）
event_type	必选	int	EN_IOTA_EVENT_GET_TIME_SYNC_RESPONSE（枚举值为5）
event_time	可选	HW_CHAR*	事件时间

字段名	必选/可选	类型	参数描述
paras	必选	EN_IOTA_DEVICE_PARAS*	子设备事件参数，此时为NULL
ota_paras	可选	EN_IOTA_OTA_PARAS*	ota事件参数，此时为NULL
ntp_paras	必选	EN_IOTA_NTP_PARAS*	NTP事件参数
gtw_add_device_paras	可选	EN_IOTA_GTW_ADD_DEVICE_PARAS	网关新增子设备事件，此时为NULL
gtw_del_device_paras	可选	EN_IOTA_GTW_DEL_DEVICE_PARAS	网关删除子设备事件，此时为NULL
device_log_paras	可选	EN_IOTA_DEVICE_LOG_PARAS*	设备日志事件参数，此时为NULL

EN_IOTA_NTP_PARAS结构体：

字段名	必选/可选	类型	参数描述
device_real_time	必选	HW_LLONG	设备真实时间

示例

请参考**5.6 接收平台新增子设备通知**示例。

10 设备信息上报

10.1 设备信息上报

接口功能

设备向平台上报设备信息（设备连接上平台后，会自动上报一次携带sdk版本号的消息）。

接口描述

```
HW_API_FUNC HW_INT IOTA_ReportDeviceInfo(ST_IOTA_DEVICE_INFO_REPORT *device_info_report, void *context)
```

参数说明

字段	必选/可选	类型	描述
device_info_report	可选	ST_IOTA_DEVICE_INFO_REPORT *	设备相关信息结构体
context	可选	void *	指向任何数据体的上下文指针，该指针会在成功或者失败的回调函数中返回。不使用该指针可以传递NULL。

ST_IOTA_DEVICE_INFO_REPORT 类型：

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

字段名	必选/可选	类型	参数描述
event_time	可选	String	事件时间
device_sdk_version	可选	String	格式为：接入方式_版本号，例如C_v0.5.0, JAVA_v0.5.0，Tiny SDK_v1.0.0等
sw_version	可选	String	软件版本
fw_version	可选	String	固件版本

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设备信息
ST_IOTA_DEVICE_INFO_REPORT deviceInfo;
deviceInfo.device_sdk_version = SDK_VERSION;
deviceInfo.sw_version = "v0.0.1";
deviceInfo.fw_version = "v0.0.2";
deviceInfo.event_time = NULL;
deviceInfo.object_device_id = NULL;
IOTA_ReportDeviceInfo(&deviceInfo, NULL);
```


11 设备日志上报

11.1 平台下发日志收集通知

命令描述

用于平台下发日志收集通知给设备。

消息体

EN_IOTA_EVENT结构体：

字段名	必选/可选	类型	参数描述
mqtt_msg_info	必选	EN_IOTA_MQTT_MSG_INFO*	mqtt协议层信息
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网关设备
services	可选	EN_IOTA_SERVICE_EVENT*	事件服务列表
services_count	可选	HW_INT	事件服务列表的个数

EN_IOTA_MQTT_MSG_INFO类型：

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。当前预留，默认为NULL。
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）

EN_IOTA_SERVICE_EVENT类型：

字段名	必选/可选	类型	参数描述
service_id	必选	HW_INT	EN_IOTA_EVENT_DEVICE_LOG（枚举值为3）
event_type	必选	HW_INT	EN_IOTA_EVENT_LOG_CONFIG（枚举值为8）
event_time	可选	HW_CHAR*	事件时间
paras	可选	EN_IOTA_DEVICE_PARAS*	子设备事件参数，此时为NULL
ota_paras	可选	EN_IOTA_OTA_PARAS*	ota事件参数，此时为NULL
ntp_paras	可选	EN_IOTA_NTP_PARAS*	NTP事件参数，此时为NULL
gtw_add_device_paras	可选	EN_IOTA_GTW_ADD_DEVICE_PARAS*	网关新增子设备事件，此时为NULL
gtw_del_device_paras	可选	EN_IOTA_GTW_DEL_DEVICE_PARAS*	网关删除子设备事件，此时为NULL
device_log_paras	必选	EN_IOTA_DEVICE_LOG_PARAS*	设备日志事件参数

EN_IOTA_DEVICE_LOG_PARAS结构体：

字段名	必选/可选	类型	参数描述
log_switch	可选	HW_CHAR*	设备侧日志收集开关 on：开启设备侧日志收集功能 off：关闭设备侧日志收集开关
end_time	可选	HW_CHAR*	日志收集结束时间 yyyy-MM-dd'T'HH:mm:ss'Z'

示例

请参考**5.6 接收平台新增子设备通知**示例。

11.2 设备上报日志内容到平台

接口功能

日志收集开关开启时设备使用该接口向平台上报日志内容（在设备连接上平台后，SDK 默认自动上报一条登录成功日志到平台），最大不超过1MB。

接口描述

```
HW_API_FUNC HW_INT IOTA_ReportDeviceLog(HW_CHAR *type, HW_CHAR *content, HW_CHAR *timestamp, void *context)
```

字段	必选/可选	类型	描述
type	必选	HW_CHAR*	日志类型（字符串）： "DEVICE_STATUS"： 设备状态 "DEVICE_PROPERTY"： 设备属性 "DEVICE_MESSAGE"： 设备消息 "DEVICE_COMMAND"：设备命令
content	必选	HW_CHAR*	日志内容。
timestamp	可选	HW_CHAR*	日志产生时间戳，精确到毫秒
context	可选	void *	指向任何数据体的上下文指针，该指针会在成功或者失败的回调函数中返回。不使用该指针可以传递NULL。

接口返回值

参见**函数标准返回值**

示例

```
// 开发者调用该接口进行时间同步请求
//report device log
```



```
long long timestamp = getTime();
char timeStampStr[14];
sprintf(timeStampStr,"%lld",timestamp);
IOTA_ReportDeviceLog("DEVICE_STATUS", "device log", timeStampStr, NULL);
```


12 远程 SSH 登录

12.1 上报远程 SSH 配置结果

接口功能

上报设备远程SSH配置的结果。

接口描述

```
HW_API_FUNC HW_INT IOTA_RptDeviceConfigRst(const ST_IOTA_DEVICE_CONFIG_RESULT
*device_config_report, void *context)
```

参数说明

字段	必选/可选	类型	描述
device_config_report	必选	const ST_IOTA_DEVICE_CONFIG_RESULT *	设备远程配置结果结构体。
context	可选	void *	指向任何数据体的上下文指针，该指针会在成功或者失败的回调函数中返回。不使用该指针可以传递NULL。

ST_IOTA_DEVICE_CONFIG_RESULT 结构体：

字段名	必选/可选	类型	参数描述
object_device_id	可选	HW_CHAR*	对应的最终目标设备，传NULL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
result_code	必选	HW_INT	标识设备配置结果，0表示成功，其他表示失败。不带默认认为成功。
description	可选	HW_CHAR[]	上报设备远程配置的结果。

接口返回值

参见**函数标准返回值**

示例

```
static int OnEventsDownRemoteCfgArrived(EN_IOTA_SERVICE_EVENT *services, const char *event_type,
JSON *paras, char *object_device_id)
{
    ST_IOTA_DEVICE_CONFIG_RESULT deviceCfgRpt = {0};
    JSON *cfg = NULL;
    if (!strcmp(event_type, DEVICE_CONFIG_UPDATE)) {
        services->event_type = EN_IOTA_EVENT_DEVICE_CONFIG_UPDATE;
        cfg = JSON_GetObjectFromObject(paras, DEVICE_CONFIG_CONTENT);
        if (cfg == NULL) {
            PrintfLog(EN_LOG_LEVEL_ERROR, "OnEventsDownRemoteCfgArrived(): paras parse failed.\n");
            return -1;
        }
        deviceCfgRpt.object_device_id = object_device_id;
        if (onDeviceConfig) {
            deviceCfgRpt.result_code = onDeviceConfig(cfg, deviceCfgRpt.description);
        }
        IOTA_RptDeviceConfigRst(&deviceCfgRpt, NULL);
    }
    return 1;
}
函数OnEventsDownRemoteCfgArrived在OnEventsDownArrived进行调用处理
```

12.2 设置设备远程配置结果

接口描述

用于接收平台返回的设备远程配置结果，用户自行实现HandleDeviceConfig函数。

示例

```
static int HandleDeviceConfig(JSON *cfg, char *description)
{
    char *cfgstr = cJSON_Print(cfg);
    PrintfLog(EN_LOG_LEVEL_INFO, "HandleDeviceConfig config content: %s\n", cfgstr);
    (void)strcpy_s(description, MaxDescriptionLen, "update config success");
    MemFree(&cfgstr);
    return 0;
}
//设置回调函数
IOTA_SetDeviceConfigCallback(HandleDeviceConfig);
```


13 端侧规则引擎

13.1 端侧规则本地存储

接口功能

端侧规则本地存储，防止设备断线后断网无法实现连接。

接口描述

```
HW_VOID IOTA_EnableDeviceRuleStorage(const char *filepath)
```

参数说明

字段	必选/可选	类型	描述
filepath	必选	const char *	用户存储规则的文件路径。

示例

```
#define DEVICE_RULE_FILE_PATH       "testdata.txt"
IOTA_EnableDeviceRuleStorage(DEVICE_RULE_FILE_PATH);
```

13.2 跨设备场景发送消息函数回调

接口描述

规则引擎中跨设备场景发送消息函数回调，用户自行实现函数HandleDeviceRuleSendMsg，进行消息通信。

示例

```
static int HandleDeviceRuleSendMsg(char *deviceId, char *message)
{
```



```
    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleDeviceRuleSendMsg(), deviceId is %s, the
message is %s",
        deviceId, message);
    return 0;
}
//设置回调函数
IOTA_SetDeviceRuleSendMsgCallback(HandleDeviceRuleSendMsg);
```


14 IoTEdge M2M 通信

目前支持使用SDK对接边缘IoTEdge，边缘节点完成消息的中转到目标设备，从而实现M2M的功能。

14.1 M2M 通信发送消息

接口功能

M2M通信发送消息。

接口描述

```
HW_API_FUNC HW_INT IOTA_M2MSendMsg(HW_CHAR *to, HW_CHAR *from, HW_CHAR *content,
HW_CHAR *requestId, void *context)
```

参数说明

字段	必选/可选	类型	描述
to	必选	HW_CHAR*	目标设备的设备id。
from	必选	HW_CHAR*	源设备的设备id。
content	必选	HW_CHAR*	发送的消息内容。
requestId	可选	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。
context	可选	void *	指向任何数据体的上下文指针，该指针会在成功或者失败的回调函数中返回。不使用该指针可以传递NULL。

接口返回值

参见**函数标准返回值**

示例

```
char *to = "deviceA";
char *from = username_;
char *content = "hello deviceB";
char *requestId = "demoIdToDeviceB";
IOTA_M2MSendMsg(to, from, content, requestId, NULL);
```

14.2 接收 M2M 消息下发

接口描述

用于接收平台返回的设备远程配置结果。

消息体

EN_IOTA_M2M_MESSAGE结构体：

字段名	必选/可选	类型	参数描述
mqtt_msg_info	必选	EN_IOTA_MQTT_MSG_INFO*	mqtt协议层信息
request_id	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。
to	必选	HW_CHAR *	目标设备设备id
from	必选	HW_CHAR *	源设备设备id
content	必选	HW_CHAR *	消息内容

EN_IOTA_MQTT_MSG_INFO类型：

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。当前预留，默认为NULL。
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）

示例

```
static void HandleM2mMessageDown(EN_IOTA_M2M_MESSAGE *rsp)
{
    if (rsp == NULL) {
        return;
    }

    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleM2mMessageDown(), requestId: %s\n", rsp-
>request_id);
    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleM2mMessageDown(), to:         %s\n", rsp->to);
    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleM2mMessageDown(), from:       %s\n", rsp->from);
    PrintfLog(EN_LOG_LEVEL_INFO, "device_demo: HandleM2mMessageDown(), content:    %s\n", rsp-
>content);

    // do sth
}
//设置回调函数
IOTA_SetM2mCallback(HandleM2mMessageDown);
```


15 端云安全通信

15.1 获取云平台配置的最新软总线信息

接口功能

主动获取IoT云平台配置的最新软总线信息。

接口描述

```
HW_API_FUNC HW_INT IOTA_GetLatestSoftBusInfo(HW_CHAR *busId, HW_CHAR *eventId, void *context)
```

参数说明

字段	必选/可选	类型	描述
busId	可选	HW_CHAR*	云平台生成软总线id，唯一标识。用户填写，如果为NULL，则查询当前设备所有的软总线最新信息；如果不为NULL，则查具体的软总线最新信息
eventId	可选	HW_CHAR*	事件ID，用来标识请求。
context	可选	void *	指向任何数据体的上下文指针，该指针会在成功或者失败的回调函数中返回。不使用该指针可以传递NULL。

接口返回值

参见**函数标准返回值**

示例

```
IOTA_GetLatestSoftBusInfo(NULL，NULL, NULL);
```

