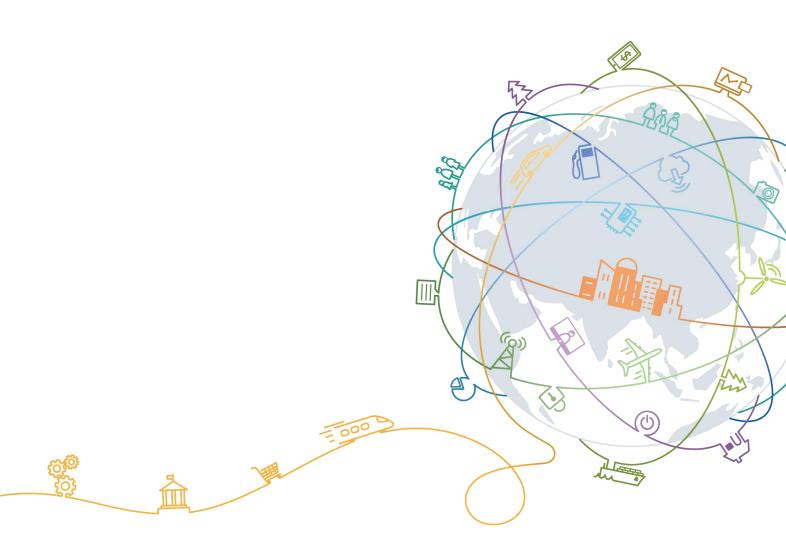
IoT Device SDK API 参考(C)

文档版本 01

发布日期 2020-02-27





版权所有 © 华为技术有限公司 2020。 保留一切权利。

非经本公司书面许可,任何单位和个人不得擅自摘抄、复制本文档内容的部分或全部,并不得以任何形式传播。

商标声明

HUAWEI 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

1	使用前必读	1
2	开发说明······	······3
2.1	数据类型说明	3
3	直连设备接入	······ <i>5</i>
3.1	初始化 SDK 资源	5
3.2	2 释放 SDK 资源	6
3.3	3 绑定配置	6
3.4	4 设置日志打印回调函数	8
3.5	5 设置回调函数	10
3.6	5 设备连接	13
3.7	7 设备断开链路	14
4	直连设备上报数据************************************	15
4.1	直连设备上报自定义 topic 消息	15
4.2	2 直连设备上报属性	16
4.3	3 直连设备上报命令执行结果	18
4.4	4 直连设备上报设备属性设置结果	19
4.5	5 直连设备上报设备属性查询结果	20
5	直连设备命令接收	22
5.1	接收消息下发	22
5.2	2 接收自定义 topic 消息	24
5.2	2.1 订阅自定义 topic	24
5.2	2.2 接收自定义 topic 消息	24
5.3	3 接收命令下发	26
5.4	4 接收平台设置设备属性	27
5.5	5 接收平台查询设备属性	29
5.6	5 接收新增子设备通知	30
5.7	7 接收删除子设备通知	36

6	子设备上报数据************************************	39
6.1	子设备上报消息	39
6.2	批量上报设备属性	39
7	OTA 升级······	42
	平台下发获取版本信息通知	
	设备上报软固件版本	
7.3	平台下发升级通知	45
7.4	设备请求下载包	46
7.5	设备上报升级状态	47
8	设备影子	49
8.1	设备请求获取平台的设备影子数据	49
8.2	设备接收平台返回的设备影子数据	50

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

功能	接口	说明
		行结果
	IOTA_PropertiesSetResponse	4.4 直连设备上报设备属 性设置结果
	IOTA_PropertiesGetResponse	4.5 直连设备上报设备属 性查询结果
直连设备命令接收	HW_VOID	5.1 接收消息下发
(相关回调接口) 	(*PFN_CALLBACK_HANDLER)/ HW VOID	5.3 接收命令下发
	(*PFN_CALLBACK_HANDLER_ WITH_TOPIC)	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

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。

返回值名称	值	类型原型
IOTA_FAILURE	-1	执行错误。
IOTA_PARAMETER_EMP TY	-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_ENCRYP T_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_NO T_FOUND	-108	找不到证书。
IOTA_MQTT_DISCONNE CT_FAILED	-109	断链失败。
IOTA_PARSE_JSON_FAI LED	-110	解析字符串失败。
IOTA_PARAMETER_ERR OR	-111	参数错误。
IOTA_NUMBER_EXCEE DS	-112	数字超过最大值。

ろ 直连设备接入

- 3.1 初始化 SDK 资源
- 3.2 释放 SDK 资源
- 3.3 绑定配置
- 3.4 设置日志打印回调函数
- 3.5 设置回调函数
- 3.6 设备连接
- 3.7 设备断开链路

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)

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)

字段	必选/可选	类型	描述
iItem (key)	必选	HW_UINT	设备绑定的配置项。

字段	必选/可选	类型	描述
			 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_NUMBE R(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)
pValue/uiV alue (value)	必选	HW_CHAR */HW_UIN T	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

					•	_
02	设备接入	IoT Device	SDK	API 参考	((7)

字段	必选/可选	类型	类型 描述	
			• 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_SECRE T 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);

字段	必选/可 选	类型	描述
	Ų		

字段	必选/可 选	类型	描述
pfnLogCallbac kHandler	必选	PFN_LOG_C ALLBACK_ 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。
- 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:

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)

字段	必选/可	类型	描述
	选		

字段	必选/可 选	类型	描述
iltem	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_CONNE CT_SUCCESS 2. 鉴权失败的通知: EN_IOTA_CALLBACK_CONNE CT_FAILURE 3. 链接断开的通知: EN_IOTA_CALLBACK_CONNE CTION_LOST 4. 设备主动断链成功的通知: EN_IOTA_CALLBACK_DISCO NNECT_SUCCESS 5. 设备主动断链失败的通知: EN_IOTA_CALLBACK_DISCO NNECT_FAILURE 6. 设备订阅成功的通知: EN_IOTA_CALLBACK_SUBSC RIBE_SUCCESS 7. 设备订阅失败的通知: EN_IOTA_CALLBACK_SUBSC RIBE_FAILURE 8. 设备发布数据成功的通知: EN_IOTA_CALLBACK_PUBLIS H_SUCCESS 9. 设备发布数据失败的通知: EN_IOTA_CALLBACK_PUBLIS H_SUCCESS 9. 设备发布数据失败的通知: EN_IOTA_CALLBACK_PUBLIS H_FAILURE ● 业务层 1. 设备收到下发的透传消息: EN_IOTA_CALLBACK_MESSA GE_DOWN 2. 设备收到下发的命令: EN_IOTA_CALLBACK_COMM AND_REQUES 3. 设备收到属性设置命令: EN_IOTA_CALLBACK_PROPE RTIES_SET 4. 设备收到属性查询命令:
			EN_IOTA_CALLBACK_PROPE RTIES_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_DEVIC E_SHADOW
pfnCallbackHa ndler	必选	PFN_CALLB ACK_HAND LER	入参为自定义函数名称
pfnCallbackHa ndler	必选	PFN_CALLB ACK_HAND LER_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

```
//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

```
// 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
- 4.2 直连设备上报属性
- 4.3 直连设备上报命令执行结果
- 4.4 直连设备上报设备属性设置结果
- 4.5 直连设备上报设备属性查询结果

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)

字段名	必选/可 选	类型	参数描述
object_device_i	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)

字段	必选/可选	类型	描述
pServiceData[必选	ST_IOTA_SE RVICE_DATA _INFO	上报的属性数据结构体

字段	必选/可选	类型	描述
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

```
// 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 *pcCommandResponse)

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_nam e	可选	HW_CHAR*	命令的响应名称,在设备关联的 产品模型中定义。
pcCommandR esponse	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

```
// 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

```
// 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_SE RVICE_DAT A_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

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

```
// 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);
```

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
- 5.2 接收自定义 topic 消息
- 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

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_i	可选	String	消息对应的最终目标设备,没有携带则

字段名	必选/可 选	类型	参数描述
d			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。

```
//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
   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);
   printfLog(EN LOG LEVEL INFO, "AgentLiteDemo: handleMessageDown(), name %s\n",
name);
   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

命今描述

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_i	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

字段名	必选/可 选	类型	参数描述
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。

```
//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
   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);
   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_devic e_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_n ame	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。

```
//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
```

```
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_devi ce_id	可选	String	属性设置对应的目标设备 ID
services	必选	List <serviced ata=""></serviced>	设备服务数据列表。

ServiceData 结构定义:

字段名	必选/可	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。

```
//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_dev ice_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。

```
//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
services	可选	List <serviceeve nt></serviceeve 	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></deviceinfo>	设备列表

字段名	必选/可选	类型	参数描述
version	必选	Long	子设备信息版本

DeviceInfo 定义表

字段名	必选/可	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

```
//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", "-
           //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",
       //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",
       //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,
                    //get value of node id
"node id", NULL);
                          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);
```

□ 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

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 <serviceeve nt=""></serviceeve>	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></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	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

}]

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
- 6.2 批量上报设备属性

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[])

字段	必选/可选	类型	描述
pServiceData[必选	ST_IOTA_DE VICE_DATA_ INFO	上报的设备数据结构体数组
serviceNum	必选	HW_INT	上报的子设备个数
serviceLenLis	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
t[]			

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[Max ServiceRepo rtNum]	必选	ST_IOTA_S ERVICE_DA TA_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。

```
"{\"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
- 7.2 设备上报软固件版本
- 7.3 平台下发升级通知
- 7.4 设备请求下载包
- 7.5 设备上报升级状态

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 <serviceeve nt></serviceeve 	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	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)

字段	必选/可选	类型	描述
----	-------	----	----

02	设备接入	IoT Device	SDK	API 参考(C)

字段	必选/可选	类型	描述
otaVersionInf o	必选	ST_IOTA_OT A_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_devic e_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

```
// 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 <serviceeve nt=""></serviceeve>	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/cald954771ae61e5098c7f83",
            "file_size": 81362928,
            "access_token": "595124473f866b033dfalf",
            "expires": 86400,
            "sign":
"595124473f866b033dfalf7e831c8c99al2f6143f392dfa996a819010842c99d"
            }
        }]
    }
}
```

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)

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_UP GRADE_STA TUS_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

字段	必选/可选	类型	描述
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_devic e_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。

设备当前版本号

HW_CHAR*

接口返回值

参见函数标准返回值

必选

version

```
// 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
- 8.2 设备接收平台返回的设备影子数据

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)

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_devic e_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	类型	参数描述
object_dev ice_id	必选	String	设备影子的目标设备 ID。
shadow	可选	List <shadowd ata></shadowd 	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	PropertiesDat a	设备影子 desired 区的属性列表。
reported	可选	PropertiesDat a	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	类型	参数描述
	选		

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_tim e	可选	String	设备属性数据的 UTC 时间(格式: yyyyMMddTHHmmssZ),如: 20161219T114920Z。

```
//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);
```